Enhancing Salted Password Hashing Technique Using Swapping Elements in an Array Algorithm

[1]Dr. Abdelrahman Karrar, [2]Talal Almutiri, [3]Sultan Algrafi, [4]Naif Alalwi, [5]Ammar Alharbi
[1,2,3,4,5]Taibah University, Saudi Arabia

Abstract
The importance of internet has become highly valid environment for organizations and governments due to providing services and easily dealing with e-commerce and government services. All of these services are provided to registered users which an organization store their profiles in its databases. User's profile may contain sensitive information such as passwords, credit card numbers, and personal data. One of most concerned issues is how to protect this sensitive information. However, a dictionary attack, brute force attack and rainbow table are the most common ways of guessing passwords in cryptanalysis. As a result, salted password hashing technique one of most efficient ways to protect user's passwords. In cryptography, a salt is random string appending or prepending to original user's password before enter it hash function. This is paper will provide guidelines to use this technique to increase efficiency for preventing dictionary attack, brute force attacks and rainbow table from guessing users' passwords. Also, this paper will provide an algorithm to improve salted password hashing technique by swapping elements in array which work to rearrange the user's password and salt before send it to hash function. For example, suppose the user' password is "123" and the salt is "abc". The common way is use hash(user' password, salt) - hash(123abc). This algorithm is to reorder user's password and salt to become like "a1bc23" and then send it to hash function. Also in common using of salted password hashing technique the salt is storing in database without any changes, therefore this algorithm we will rearrange the salt before storing it in database. This algorithm will make guessing the password more difficult because isolating the password and the salt from each other through the final hashes will become very difficult.

Keywords
Password Security, Hash Algorithm, Encyption, Array Algorithm.

I. Introduction
Hashing algorithms are used to convert passwords into string called hash values, hash codes, digests, or simply hashes. A hash is also a one-way function which theoretically cannot reverse or get a plain text from hash. Salted password hashing means appending or prepending a random string to the user's password before hashing.

To make hashes more secure, salt can be added to the hash. This means that, a random string of characters is either prefixed or postfixed to the password before hashing it. Every password has a different salt. Even if the salts are stored on the database, it will be very complicated cracking the passwords using a rainbow table as the salted passwords are long, complex and unique. Salted hashes can be brutally forced but the time taken is significantly longer. Using two salts, one public and one private can also protect the password against offline attacks [1].

To store password in database there is four ways. The first way: storing password in a plain text that mean the password does not need any process to discover it can be known just by looking. The second way: Using symmetric encryption to convert password from the plain text – the original string- into cipher text by using the same cryptographic key which could lead to possibility of decryption the cipher text and stolen the password by knowing the cryptographic key. The third way: Using password hashing algorithm to generate non-reversible hashes but dictionary attack, brute force attacks and rainbow table can guess the password because the hashes will be same for all the same plain text. The fourth way: Using salted password hashing to generate different hashes for the same plain text that make guessing too difficult.

Most database applications usually store their passwords in the database as plain text which is not a sufficient method for protection, especially for applications that hold sensitive data of user [2]. MD5 (Merkl-Damagerd), SHA1 (Secure Hash Algorithm) and RIPEMD (RACE Integrity Primitives Evaluation Message Digest) algorithm are considered as broken algorithms and they should not be used in the new applications [2].

Swapping Elements in an array algorithm focuses on rearrange hash input (password and salt) before send to hashing function that arrangement is flexible where we relied on the even elements - positions - of the array to store the password and then reverse the salt and store it in the remaining places. This algorithm, also, rearrange salt before store it in database based on a predefined array.

II. Related Work
Nicholas Oluwole Ogini and Noah Oghenefego Ogwara focused on database passwords security, using a strong hashing algorithm and salting. They provided a platform to eliminate the need to ever persist user password in plain text or easy to know any users' password. This system is implemented using the message digest 5 (MD5) algorithms for hashing and salt pattern. They, also, provided some challenges of the existing such as storing passwords in a plain text and encrypting passwords before store it [2]. They used username as salt, but this is a contrary to the principle of salt where it should be random and preferable not to create an algorithm for generating random number.

Prathamesh Churi, Medha Kalelkar and Bhavin Save had provided a new algorithm for improvising password encryption using Jumbling-Salting-Hashing technique. JSH algorithm consists of three techniques namely jumbling, salting and hashing. In the jumbling part, the password undergoes "addition", "selection" and "reverse" processes. Addition process is responsible for generating a value required for determining the number of characters to be added to the password [4]. Selection deals with selecting characters to be added to the password from predefined character set. Reverse process is responsible for reversing the output of selection process on some predefined condition. In salting part, random salt is added to the jumbled password. Selection of salt is based on timestamp value which is determined when the user creates an account. Finally, jumbled and salted password is given to the hashing procedure where predefined hashing algorithm such as SHA algorithm is implemented [5]. They added a number of characters to the password from predefined character set which is similar to the principle of salt by having an additional string to the password.

III. Guidelines for Improving Password Security

There are many aspects to password security that must be considered. These include the manner in which passwords are chosen. Also, salt reuse, short salt, hash algorithm should be used and Cryptographically Secure Pseudo-Random Number Generator (CSPRNG).

A. Choosing a Password

The password must be difficult to guess and also should be easy to remember. Personal information or common words such as names, special dates, places, and dictionary words should be avoided when choosing a password. Using a variation of capital letters, small letters, numbers, and symbols is highly recommended as well as the length of a password should be considered at least 9 letters.

B. Salt Reuse

Using the same salt in each hash is ineffective because if two users have the same password, they will still have the same hash which an attacker can still use a reverse lookup table attack to run a dictionary attack on every hash at the same time. As a result, a new random salt must be generated each time a user creates an account or changes their password.

C. Short Salt

Using a short salt also ineffective because attackers may be able to guess user' password by generating a lookup table for every possible salt. A salt recommended being at least 32 random bytes.

D. Hash Algorithm

A fast cryptographic hash functions such as MD5 and SHA1 this algorithm are considered as broken algorithm. MD5 is a fast hashing function, that is, it is computationally fast to calculate. Iterative hashing makes the calculation slower, hence computationally slower and more difficult to crack. The number of iterations can typically be made to be equal to 1000 [5].

However, a key derivation functions such as bcrypt, crypt, PBKDF2, and scrypt are slower than common functions like MD5 and SHA1 but is hard to break it.

E. Cryptographically Secure Pseudo-Random Number Generator (CSPRNG).

Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) should be used to generate salt. It is designed to be cryptographically secure. However, using simple Pseudo-Random Number Generator like rand() function is not recommended. It is designed for generating random number only without any consideration of cryptography.

IV. Swapping Elements in an Array Algorithm

Swapping Elements in an array algorithm consists of two major modules, Hash input rearrangement module and Salt rearrangement module. The diagram of swapping Elements in an array algorithm is given in fig. 1.

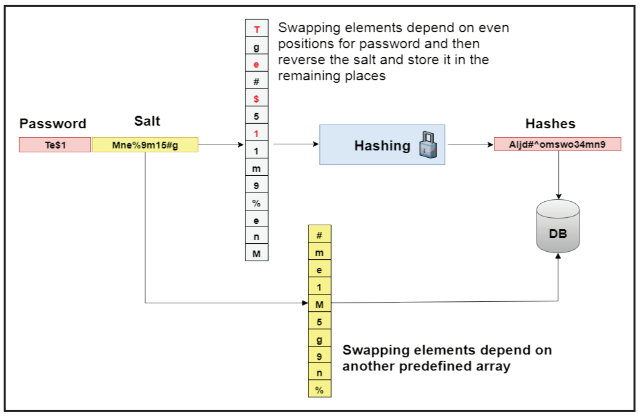
Swapping elements depend on even positions for password and then reverse the salt and store it in the remaining places
Password
Te$1
Salt
Mne%9m15#g
T g e # $ 5 1 1 m 9 % e n M
Hashing
Hashes
Aljd#^omswo34mn9
DB
# m e 1 M 5 g 9 n %
Swapping elements depend on another predefined array

Fig. 1: Architecture Diagram of Swapping Elements in an Array Algorithm

A. Hash Input Rearrangement Module

This module is responsible for rearranging hash input (password and salt). It is consisted of two processes, Reversion Process which is receiving the salt then reversing its elements and preparing the salt for next process. Rearrangement Process, which gets the reversed salt and original user's password then it, will be placing the first letter of password in the first even position in an array and inputting the next letter in the next even position. After finishing the password, it will add the reversed salt sequentially to the remaining positions. Then, the rearranged hash input will be sending to hash function to generate hashes. Fig. 2 shows Hash Input Rearrangement Module.

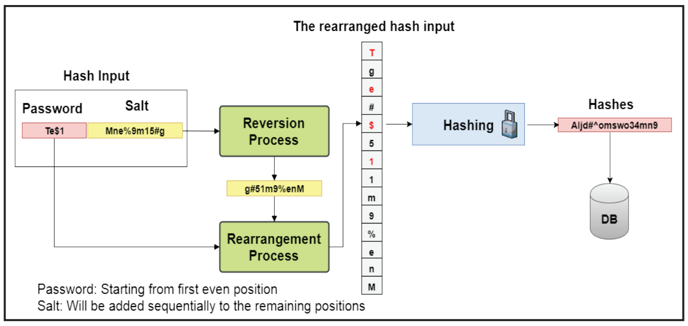
The rearranged hash input
Hash Input
Password
Salt
Te$1
Mne%9m15#g
Reversion Process
g#51m9%enM
Rearrangement Process
T g e # $ 5 1 1 m 9 % e n M
Hashing
Hashes
Aljd#^omswo34mn9
DB
Password: Starting from first even position
Salt: Will be added sequentially to the remaining positions

Fig. 2: The Diagram of Hash Input Rearrangement Module

B. Salt Rearrangement Module

This module is responsible for rearranging the salt before storing it in database. Salt rearrangement process is based on predefined array, which contains the new positions of the salt. The purpose of rearranging salt is to reduce the possibility of salt detection that mean guessing of the password will be very difficult. Fig. 3 shows Salt Rearrangement Module.

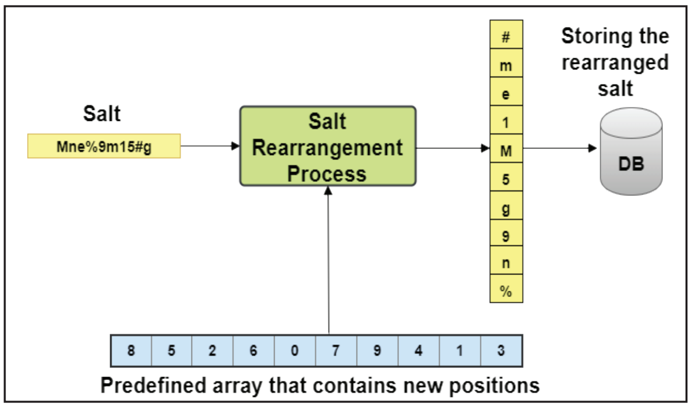
Salt
Mne%9m15#g
Salt Rearrangement Process
# m e 1 M 5 g 9 n %
Storing the rearranged salt
DB
8 5 2 6 0 7 9 4 1 3
Predefined array that contains new positions

Fig. 3: The Diagram of Salt Rearrangement Module

V. Algorithm Steps

The detailed steps of Swapping Elements in an array algorithm will be:

1. Receiving user' password from user.
2. Generating the salt.
3. Reversing the salt by Reversion Process.
4. Rearranging hash input (user's password and salt) by Rearrangement Process.
5. Sending hash input to hash function to generate hashes.
6. Rearranging the salt by Salt Rearrangement Process
7. Storing hashes and the rearranged salt in database.

VI. Algorithm Implementation

The algorithm will be implemented using SHA2 with .Net framework using C#.Net 2013. Two pages will be implemented, Registration page, and Login page. The main class is Swapping Elements Algorithm which contains this methods.

```
using System;
using System. Security. Cryptography;
using System.Text;

public class SwappingElementsAlgorithm
{
            public SwappingElementsAlgorithm()
            {
            }
   private const int saltSize = 24;
   public static string GenerateSalt()
   {
      RNGCryptoServiceProvider rng = new
RNGCryptoServiceProvider();
      byte[] salt = new byte[saltSize];
      rng.GetBytes(salt);
      return Convert.ToBase64String(salt);
   }

   // Reversion Process
   public static string ReversingSalt(string Salt)
   {
      char[] reversedSalt = Salt.ToCharArray();
      Array.Reverse(reversedSalt);
      return new string(reversedSalt);
   }

   // Rearrangement Process
    public static string RearrangingHashInput(string Paasword,
string ReversedSalt)
   {
      char[] password = Paasword.ToCharArray();
      char[] reversedSalt = ReversedSalt.ToCharArray();
      char[] rearrangedHashInput = new char[password.Length +
reversedSalt.Length];

      int pIndex = 0 ;
      int rsIndex = 0;

      for (int i = 0; i < rearrangedHashInput.Length; i++ )
      {
         if(i % 2 == 0) // even number
         {
            if (pIndex < password.Length)
            {
               rearrangedHashInput[i] = password[pIndex];
               pIndex++;
            }
            else
            {
               rearrangedHashInput[i] = reversedSalt[rsIndex];
               rsIndex++;
            }
         }
else
         {
            rearrangedHashInput[i] = reversedSalt[rsIndex];
            rsIndex++;
         }
      }

      return new string(rearrangedHashInput);
   }

//Salt Rearrangement Process
   public static string RearrangingSalt(string Salt)
   {
      // new  positions 32 elements depend on salt length
      int[] predefinedArray = { 3,0,31,24,5,9,17,29,13,1,
        6,10,18,2,4,8,7,15,21,19,11,20,27,23,14,12,22,16,25,30,
         26,28};

         char[] rearrangedSalt = new char[predefinedArray.
Length];
      for (int i = 0; i < predefinedArray.Length; i++)
      {
                  rearrangedSalt[i] = Salt.ToCharArray()
[predefinedArray[i]];
      }

      return new string(rearrangedSalt);
   }
   //SHA384 hash function
      public static string HashPasswordUsingSHA384(string
HashInput)
   {
      SHA384 sha384 = new SHA384CryptoServiceProvider();
         byte[] hash = sha384.ComputeHash(Encoding.UTF8.
GetBytes(HashInput));
      return Convert.ToBase64String(hash);



   }
    //Verifying user's password. Calling from login page
   public static bool VerifyPassword(string UserPassword, string
Salt,string HashedPassword)
   {
         string reversedSalt = SwappingElementsAlgorithm.
ReversingSalt(Salt);
      string rearrangedHashInput = SwappingElementsAlgorithm.
RearrangingHashInput(UserPassword, reversedSalt);
        string hash = HashPasswordUsingSHA384(rearrangedHa
shInput);

      if (String.Compare(hash, HashedPassword, false) == 0 )
         return true; // password is correct
    else
      return false;

  }
}
```

Registration page is shown in fig. 4.

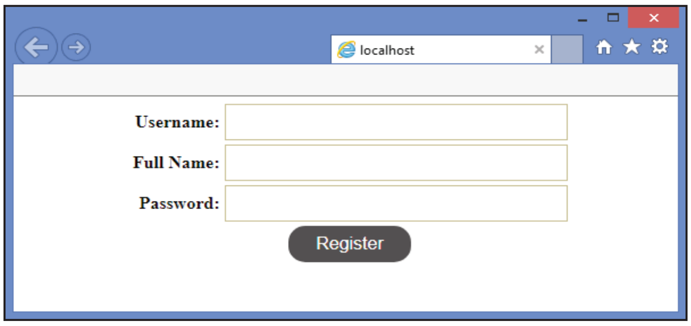

Fig. 4: The Design of Registration Page

```
protected void btnRegister_Click(object sender, EventArgs e)
  {
    string salt = SwappingElementsAlgorithm.GenerateSalt();
       string reversedSalt = SwappingElementsAlgorithm.
ReversingSalt(salt);
   string rearrangedHashInput = SwappingElementsAlgorithm.
RearrangingHashInput(txtPassword.Value, reversedSalt);
    string hashes = SwappingElementsAlgorithm.HashPasswo
rdUsingSHA384(rearrangedHashInput);
    // Storing user info in database also hashes and salt
    SqlParameter[] parameters = new SqlParameter[]{
            new SqlParameter("@Username",txtUsername.
Value),
        new SqlParameter("@FullName",txtName.Value),
        new SqlParameter("@Salt",salt),
        new SqlParameter("@Hashes",hashes),
        };
        if (SQLHelper.ExecuteNonQuery("INSERT INTO
UsersAccounts VALUES(@Username,@FullName,@Salt,@
Hashes)", CommandType.Text, parameters))
      lbMessage.Text = "Successfully saved";
    else
      lbMessage.Text = "Failed to save data";
  }
```

Login page is shown in Fig 5.

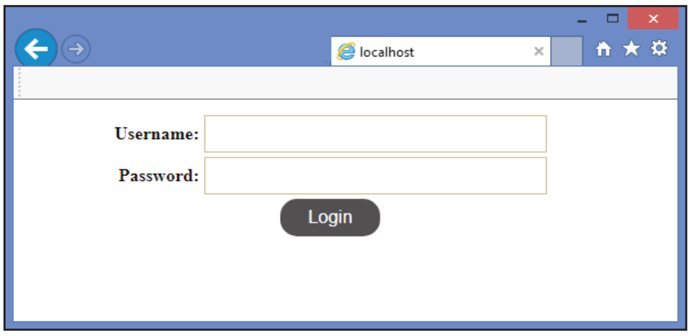

Fig. 5: The Design of Registration Page

```
protected void btnLogin_Click(object sender, EventArgs e)
  {
      SqlParameter[] parameters = new SqlParameter[]{
            new SqlParameter("@Username",txtUsername.
Value),
        };

      // Getting data from database
    DataTable userInfo = SQLHelper.ExecuteQuery("SELECT
* FROM UsersAccounts WHERE Username=@Username",
CommandType.Text, parameters);
      if (userInfo == null || userInfo.Rows.Count <= 0)
            lbMessage.Text = "Username or password is
incorrect";
      else
      {
        string salt = userInfo.Rows[0]["Salt"].ToString();
              string hashedPassword = userInfo.Rows[0]
["HashedPassword"].ToString();
   if(SwappingElementsAlgorithm.VerifyPassword(txtPassword.
Value, salt, hashedPassword))
          lbMessage.Text = "successfully logged in";
        else
              lbMessage.Text = "Username or password is
incorrect";
      }
    }
  }
```

VII. Future Scope

The future scope of algorithm will include some enhancements and modifications that can be implemented using the other hashing algorithm and key derivation functions such as PBKDF2. A new method can be implemented is using the salt not, only, as an additional string to the password before rearrangement, but also as an input to hashing function which means adding salt to the rearranged hash input.

VIII. Conclusion

In cryptanalysis, a dictionary attack, brute force attack and rainbow table are the most common ways of guessing passwords. However, there are many aspects to password security that must be considered which some of them should be avoided to increase security ratio such as selecting a strong password consists of variations of capital letters, small letters, numbers, and symbols, as well as voiding using short salt and using the same salt in each hash. Also, using a fast cryptographic hash functions such as MD5 and SHA1 which are considered as broken algorithm is not highly recommended. Therefore, salted password hashing technique is one of most efficient ways to protect user's passwords. In addition, swapping Elements in an array algorithm consists of two major modules, Hash input rearrangement module and Salt rearrangement module which try to enhancing salted password hashing technique by rearranging hash input before hashing it and rearranging salt before storing it in database. This algorithm is recommended to reduce the possibility of guessing the password and make it more difficult.

References

[1] K. Chanda,"Password Security: An Analysis of Password Strengths and Vulnerabilities", I. J. Computer Network and Information Security, 2016, 7, pp. 23-30.

[2] N. O. Ogini, N. O. Ogwara,"Securing Database passwords using a combination of hashing and salting techniques", IPASJ International Journal of Computer Science (IIJCS), Vol. 2, No. 8, pp. 52-58, 2014.

[3] P. Sriramya, R. A. Karthika,"Providing password security by salted password hashing using bcrypt algorithm", ARPN Journal of Engineering and Applied Sciences, Vol. 10, No. 13, pp. 5551-5556, 2015.
[4] Secure password methods(salting and hashing methods) , https://crackstation.net/hashing-security.htm#salt
[5] Prathamesh Churi, Medha Kalelkar, Bhavin Save, "JSH Algorithm: A Password Encryption Technique using JumblingSalting-Hashing", International Journal of Computer Applications, Vol. 92, No. 02, pp. 26-29, 2014.
[6] M. C. Kioon, Z. Wang, S. D. Das,"Security Analysis of MD5 algorithm in Password Storage", Proceedings of the 2nd International Symposium on Computer, Communication, Control and Automation (ISCCCA-13), pp. 706-709, 2013.