International Journal of Electrical and Computer Engineering (IJECE)
Vol. 10, No. 6, December 2020, pp. 6582~6591
ISSN: 2088-8708, DOI: 10.11591/ijece.v10i6.pp6582-6591

Authentication and password storing improvement using SXR algorithm with a hash function

Jakkapong Polpong[1], Pongpisit Wuttidittachotti[2]
[1]Faculty of Information Technology and Digital Innovation,
King Mongkut's University of Technology North Bangkok, Thailand
[2]Department of Data Communication and Networking, Faculty of Information Technology and Digital Innovation,
King Mongkut's University of Technology North Bangkok, Thailand

Article Info

Article history:

Received Apr 13, 2020
Revised Jun 2, 2020
Accepted Jun 15, 2020

Keywords:

Dictionary attacks
Hash function
Password security
Secure password storage

ABSTRACT

Secure password storing is essential in systems working based on password authentication. In this paper, SXR algorithm (Split, Exclusive OR, and Replace) was proposed to improve secure password storing and could also be applied to current authentication systems. SXR algorithm consisted of four steps. First, the received password from users was hashed through a general hash function. Second, the ratio and the number of iterations from the secret key (username and password) were calculated. Third, the hashed password and ratio were computed, and the hashed password was divided based on the ratio (Split) into two values. Both the values were applied to XOR equation according to the number of iterations, resulting in two new values. Last, the obtained values were concatenated and stored in the database (Replace). On evaluating, complexity analyses and comparisons has shown that SXR algorithm could provide attack resistance with a stronger hashed password against the aforementioned attacks. Consequently, even if the hackers hacked the hashed password, it would be challenging and would consume more time to decrypt the actual one, because the pattern of the stored password is the same as the one that has been hashed through the general hash function.

Corresponding Author:

Pongpisit Wuttidittachotti,
Department of Data Communication and Networking,
King Mongkut's University of Technology North Bangkok,
1518 Pracharat Sai 1 Rd. Bangsue, Bangkok 10800, Thailand.
Email: pongpisit.w@it.kmutnb.ac.th

1. INTRODUCTION

In the 21st century, Information Technology plays a significant role in our daily activities. Many agencies rely on information technology for accessing various services on the internet such as financial transactions, e-mail, social networks, remote desktop, access via command line, etc. The usage of various services consists of three parts: Confidentiality, Integrity, and Availability [1]. Confidentiality means storing the information or keeping the information confidential, which only allows access to those who have the right to the information. Integrity means protecting the information to remain complete and accurate. There should be a verification mechanism to check for alterations that may occur without permission. Availability means the ability to access the network data or resources whenever necessary. In other words, a person with the right or permission to access the network information or data source can access it at all times without any latency [2].

Secure Password Storing is a critical issue [3, 4], and there are many methods to choose from cryptography to the usage of a hash function. These can change the messages so that they could not be read as actual messages anymore or in some cases, even if they could be read, they could not be understood.

The main goal of storing passwords securely is to protect confidential information from users with malicious intention, prevent data leakage, and to prevent data from being changed during transmission. The system should able to detect whether the data has changed during any transmission. Nowadays, websites with many users are hacked, member lists containing Username, E-mail, and Password, as well as other information, such as addresses or credit card information. Oftenly, the trap from the website such as "have i been pwned?" revealed that many famous websites, such as Adobe, Apollo, Avast, BitTorrent, LinkedIn, and Yahoo had been hacked to reveal membership information [5].

In summary, this research aims to design and develop an algorithm to increase the security of store user data in web applications relying on working with hash functions in various algorithms such as MD5, SHA1, SHA256, and SHA512. Currently, the security of the MD5 algorithm has been significantly reduced because a group of malicious users can attack it. They develop password comparison tables of MD5 called Rainbow Tables, which are large tables created from the hash function [6]. These tables consist of general basic vocabulary that is commonly used as a password.

In order to improve security and to make it difficult to decrypt the secured data from malicious use, the password is obtained from the hash function by splitting the data into 2 parts. The ratio can be calculated from the (1) called "Split". Next step is to compute the number of working iterations from (2) (Iteration), and then taking these two values through a mathematical process from (3) called "XOR" (Exclusive - OR), the result is called "Replace". Then, the two values are arranged together, which is called the SXR algorithm (Split, Exclusive - OR, Replace). Finally, we compare attack resistance and processing time between traditional hash functions and enhanced security hash functions with the SXR algorithm.

2. LITERATURE REVIEWS

This section describes existing secure password storing methods. A hash function is similar to encrypting by a technique of changing the format of the message. The hash function will change the password into random characters and random numbers. Popular hash functions are MD5 and SHA-1 [7, 8]. The two features of the hash function to increase password security are as follows: the hash function cannot be reversed, and then the possibility of two different passwords generates the same hash value by random characters and numbers which is very hard method.

A method used to increase stronger stored passwords can be done by using a salted password. It can increase the length of passwords by adding a random number of data sets into the password that the user enters before going through the hash function. However, there is a limitation that the salt value must be stored in the database. If attackers can access a database with salt values, they can easily decrypt to get the real password [9, 10]. Mariam and Sujitha conducted a study on security analysis of salt and password hashes. This research was focused on evaluating the effectiveness of salt in passwords, such as prefix or suffix which can make attackers more difficult to predict. If the attackers try many experiments until they find the fixpoint, the password then could not be protected [11].

Prathamesh et al. presented a technique to increase the security of plain text passwords in the server database. The research offered methods of encryption by process Jumbling-Salting (JS) which can prevent dictionary attacks and brute force attacks by increasing the length of the ciphertext. The ciphertext is chosen from a predefined character set with modulus mathematical functions. When experimenting with AES encryption, it was found that this method made deciphering more time and more secure [12]. In the previous methods, although salted randomly assigned to each user, it was kept in the database as a constant. The vulnerability is that if the database is hacked, it can easily identify and use salt values to decrypt passwords. Therefore, if the use of Access-based Salts that changes continuously, it will make the password more secure [13].

In addition to increasing the efficiency of the hash function, Chawdhury and Habib proposed methods to increase the efficiency of MD5 in password security using six reserve bits, which is usually not used in the TCP Header. Encrypting password using hash should be performed prior transmitting it out through the network. Server side then uses the values from the TCP Header to decrypt the hash password. This technique makes it harder for those who want to attack MD5 with a Rainbow Table [14]. The SHA-1 algorithm has been improved to increase security when applied to web applications, especially for password hashing. A simple quadratic function and salt values has been implemented to increase the complexity of creating a hash table which makes it difficult to attack [15, 16]. Seong and Kim had adopted hash algorithms, such as SHA-1, SHA-256, and MD5, to increase the efficiency of transmitting long-distance communications between the sea and the ground to prevent information loss [17]. Gurpreet et al. presented the use of Message Authentication Code (MAC) using secret codes created by Deoxyribonucleic Acid (DNA) and Linear Random Number Generator (LCG) to work with the hash function [18].

From the related work discussed in this paper, most of the existing techniques usually had focused on improving the hash function. Although the salted password has can improve the efficiency of store passwords, it may require stores in the database. The proposed SXR in this paper, on the other hand, focuses on manipulating the original hash value by Split, XOR, and Replace based on username and password.

3. PROPOSED METHOD

At present, almost all websites only use one-factor authentication. The user just enters username and password to access the information on that website. Especially, medium and small websites do not have a budget to use TLS or multi-factor authentication services [19], such as OTP, biometric, token, etc. This makes the websites possibly insecure, and there is a high risk of being hacked [20, 21]. Most users prefer to use the same username and password on all websites, and this is their weakness. When an attacker decrypts this information into a plaintext, the attacker can use that username and password on other websites, thereby impersonating the rightful person. The attacker may gain access to important and confidential information from these websites if they do not have professional data protection standards [22, 23].

This research has divided the experimental method into two parts: the first part is the speed test, and the second part is a security test for attack from malicious users. In this experiment, we applied the operating principle of mathematical equations, called XOR (Exclusive - OR), which was used as the key to the design and development of algorithms. XOR was used to design the equations for the calculation of the 3 equations to increase password security. The first equation was the equation to find the ratio (Split), the second equation was the equation for finding the number of iterations (Iteration). The third equation was a replacement equation (Replace). The working of the whole system is shown in Figure 1.

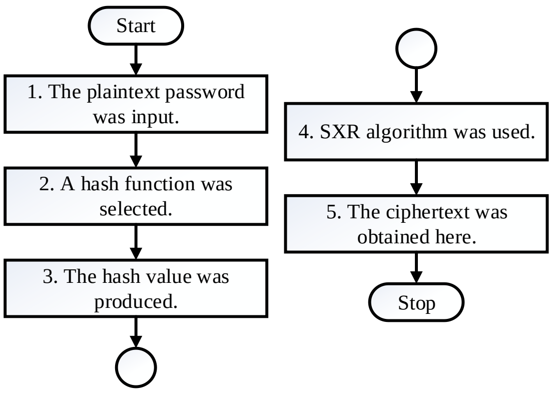
Start
1. The plaintext password was input.
2. A hash function was selected.
3. The hash value was produced.
4. SXR algorithm was used.
5. The ciphertext was obtained here.
Stop

Figure 1. System workflows

The workflow of systems consisted of five steps, see Figure 1. The first step was to use the password in the plain text format that the general public could understand. The second step was to select a hash function that was used for the transformation of the password that made it human-readable, consisting of popular algorithms, such as MD5, SHA1, SHA2, SHA224, SHA256, and SHA512. The third step was the result of the hash function format that humans do not understand. With each data passing through the hash function, it must have been unequal and had specific characteristics. The results of this process were called hash value. The fourth step was the introduction of the SXR algorithm. The algorithm has been proposed to increase the efficiency of the hash values. They were obtained from the previous steps to make it more complex. The detail of the work process is explained in Figure 2. The last step was the results obtained through the SXR algorithm. This information was unique and could not be calculated backward.

The SXR algorithm was used to increase the efficiency of the hash values, see Figure 2. First step, the password was used through the hash function. For example, the username was “Jakka1b2” and the password was “Polpong” after passing the hash function with the MD5 algorithm, the result was “f7540c62489302e-375e48c6e6670f6f2”. To create the secret key, the first 4 characters of the password “Polp” and the last 4 characters of the username “a1b2” were used to generate the secret key “a1b2Polp”. Second step, the secret key was configured to be used to calculate the ratio (section 3.1), and the number of iterations (section 3.2) was used in the processing. Third step, the first equation was used to calculate the ratio (90%) and the second equation to calculate the number of iterations (27,548).

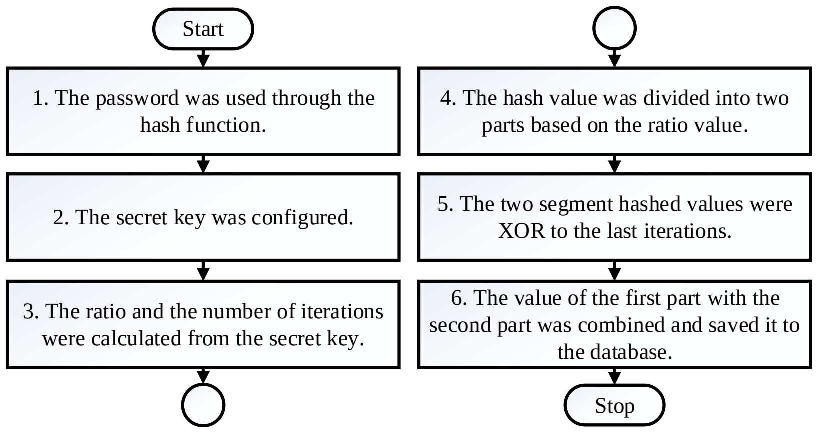
Start
1. The password was used through the hash function.
2. The secret key was configured.
3. The ratio and the number of iterations were calculated from the secret key.
4. The hash value was divided into two parts based on the ratio value.
5. The two segment hashed values were XOR to the last iterations.
6. The value of the first part with the second part was combined and saved it to the database.
Stop

Figure 2. SXR algorithm workflow

In the fourth step, the hash value was divided into two parts based on the values obtained in the third step. For example, the calculated ratio was 90% of the hash value "f7540c62489302e375e48c6e6670f6f2". The 90% ratio would divide "F7540c62489302e375e48c6e6670 | f6f2" and then bring the divided values to move the bit in the middle by moving from right to left. The result would be "02e375e48c6e6670 | f6f2f7540c624893" and then "02e375e48c6e6670". This result was assigned as the first part, while "f6f2f7540c624893" was the second part to be use in the next step. To create the new password, the authors used both values from the fourth step to calculate by using the SXR equation (section 3.3) which performed all calculations according to the number of iterations obtained from the iterations equation in the fifth step. The last step was to collate the results of the calculations from the fifth step by combining the value of the first part with the second part and saved it to the database.

3.1. Ratio equation

The ratio (1) is the equation used to determine the ratio of the password obtained from the calculation with the hash function. The result was divided into two parts as follows**:**

$$[\ ((\text{index}[0] \oplus \text{index}[4]) + 128) * ((\text{index}[1] \oplus \text{index}[5]) + 128)\ ] \bmod 99 = \text{ratio} \quad (1)$$

where

Index [0]	**:** the position of the first character of the secret key
Index [4]	**:** the position of the 5th character of the secret key
Index [1]	**:** the position of the 2nd character of the secret key
Index [5]	**:** the position of the 6th character of the secret key
128	**:** to increase the size of the bits in 1 character to the size in the binary number range 00000000- 11111111 (0-255)
Mod 99	: to determine the ratio in the group to be in the range 0 – 99%

To calculate ratio, the authors used the secret key (**"**a1b2Polp"**)** of username and password. Thus, the ratio equation was $[((a \oplus P) + 128) * ((1 \oplus o) + 128))] \bmod 99 = \text{ratio}$, the ratio of this equation was 90%.

3.2. Iterations equation

The equation for finding the number of iterations (2) is the equation used to determine the number of iterations. Calculating the password generation of new hash values could be performed as follows:

$$[\ ((\text{index}[2] \oplus \text{index}[6]) + 128) * ((\text{index}[3] \oplus \text{index}[7]) + 128)\ ] = \text{iteration} \quad (2)$$

where

Index [2]	: the position of the 3rd character of the secret key
Index [6]	: the position of the 7th character of the secret key
Index [3]	: the position of the 4th character of the secret key
Index [7]	: the position of the 8th character of the secret key
128	: to increase the size of the bits in 1 character to the size in the binary number 00000000 - 11111111 (0-255)

With the number of iterations, it was very likely that the ratioof the iteration equation was $[((b \oplus l) + 128) * ((2 \oplus p) + 128))]$ = iteration, the iteration of this equation was 27,548.

3.3. SXR equation

The equation for SXR (3) is the equation used to calculate hash values with the XOR operator combined with moving the bit to the right as follows:

$$
\begin{array}{lclcl}
X_{i1} & \oplus & X_{j1} & = & X_{ij1} \\
X_{ij1} \text{ Shift left i \% max index } X_{ij1} & * & i_1 & = & X_{y1} \\
X_{i1} & \oplus & X_{y1} & = & X_{i2} \\
X_{j1} & \oplus & X_{y1} & = & X_{j2} \\
[\,(X_{i2} \oplus X_{y1}) & \oplus & (X_{j2} \oplus X_{y1})\,] * i_2 & = & X_{ij2} \\
X_{ij2} \text{ Shift left i \% max index } X_{ij2} & * & i_2 & = & X_{y2} \\
X_{i2} & \oplus & X_{y2} & = & X_{i3} \\
X_{j2} & \oplus & X_{y2} & = & X_{j3} \\
[\,(X_{i3} \oplus X_{y2}) & \oplus & (X_{j3} \oplus X_{y2})\,] * i_3 & = & X_{ij3} \\
X_{ij3} \text{ Shift left i \% max index } X_{ij3} & * & i_3 & = & X_{y3} \\
 & & \vdots & & \\
[\,(X_{in-1} \oplus X_{yn-1}) & \oplus & (X_{jn-1} \oplus X_{yn-1})\,] * i_{n-1} & = & X_{ijn} \\
X_{ijn} \text{ Shift left i \% max index } X_{ijn} & * & i_n & = & X_{yn}
\end{array} \qquad (3)
$$

where

X_i : the value obtained from the division of half from the first part of the (1)

X_j : the value obtained from the division of half from the second part of the (1)

i : the value obtained from calculating the number of iterations from the (2)

The SXR equation for calculating the new password with the ratio and the number of iterations were devided into two equations. The X_{i1} was the first part, the X_{j1} was the second part and the i was defined by the number of iterations. For example, if the hash value that divided by the ratio was "02e375e48c6e6670-f6f2f7540c624893", the X_{i1} of this hash value was "02e375e48c6e6670", the X_{j1} of this hash value was "f6f2f7540c624893" and the i was "27,548". Then, the two values were arranged together (X_{yn} and X_{in-1}), which result in the new password of this hash value which was "ec962b4538afab7c66971839e6629ef9" and it was kept in the database as depicted in (4).

$$
\begin{array}{lclcl}
\text{02e375e48c6e6670} & \oplus & \text{f6f2f7540c624893} & = & \text{f41182b0800c2ee3} \\
\text{f41182b0800c2ee3 Shift left 1\% max index 16 * 1} & & & = & \text{41182b0800c2ee3f} \\
\text{02e375e48c6e6670} & \oplus & \text{41182b0800c2ee3f} & = & \text{43fb5eec8cac884f} \\
\text{f6f2f7540c624893} & \oplus & \text{41182b0800c2ee3f} & = & \text{b7eadc5c0ca0a6ac} \\
[\,(\text{43fb5eec8cac884f} & \oplus & \text{41182b0800c2ee3f}) \oplus (\text{f6f2f7540c624893} \oplus \text{41182b0800c2ee3f})\,] * 2 & = & \text{16a135371019d81b} \\
\text{16a135371019d81b Shift left 2\% max index 16 * 2} & & & = & \text{1426a6e2033b0362} \\
 & & \vdots & & \\
[\,(X_{in-1} \oplus X_{yn-1}) & \oplus & (X_{jn-1} \oplus X_{yn-1})\,] * i_{n-1} & = & X_{ijn} \\
X_{ijn} \text{ Shift left i \% max index } X_{ijn} & * & 27{,}548 & = & X_{yn}
\end{array} \qquad (4)
$$

4. RESULTS AND DISCUSSION

The proposed SXR algorithm with hash function was evaluated in two aspects: processing time and attack resistance. For the experiment, the popular passwords of 2018 and the test set 4 forms were utilized. The evaluation was divided into five parts. The first part was data used in performance testing. The second part was performance evaluation. The third part was comparison of the effectiveness of resistance to attack. The fourth part was collision efficiency comparison. The fifth part was application of the SXR algorithm.

4.1. Dataset and experiment setting

For the experiment, the data consisted of five forms referenced from the research on "Password Entropy and Password Quality" [24] and the other four forms which consisted of the first form (test set 1) was a popular year 2018 [25], second form (test set 2) all lowercase characters (abcdefghijklmnopqr-stuvwxyz), third form (test set 3) all uppercase characters (ABCDEFGHIJKLMNOPQRSTUVWXYZ), fourth form (test set 4) all alphanumeric characters (01234567890), and fifth form (test set 5), characters mixed between form 1-4 together with special characters (! @ # $ % ^ & * () _ + []: <>?), as shown in Table 1. All the experiments were carried out using a desktop computer. The specification was Intel Core i7 3.4 GHz with 32 GB RAM. The proposed technique was implemented in Python3. The efficiency of the algorithm was tested by comparing between the traditional hash functions, including MD5, SHA1, SHA256 and SHA512, and the hash function that used the SXR algorithm.

Table 1. Examples of passwords for test set 1-5

Test set 1	Test set 2	Test set 3	Test set 4	Test set 5
123456	vyoniurxhsld	EWHLSBRWG	164739363421	iqC-NXrQ
password	orpctieqjfwc	KUOHDLHMF	57289228	srvYoC;bX[=R
123456789	eamwyuwwt	IGRVYEPTAK	556130746974	0|O=-=M4f
12345678	lcqwjjlhkljb	NXKIVHIXH	15539653860	#m#y-?^6O4
12345	ufshblrhbf	STTYWUZQ	45677928	gP5/*OJGyU

4.2. Performance evaluation

The performance evaluation was divided into two parts. Thefirst part was the size, the number of bits that have come out from the normal process and the SXR algorithm implementation. Second part was a comparison of the time spent in the process between normal and through the SXR algorithm.

Examples used in performance testing include

Username	: ChanGa1b2
Password	: Jakkapong

Ramark: The results were in different font sizes according to each algorithm.

The second method was a method of SXR algorithm that used the a1b2Jakk as a secret code to increase the efficiency of the password. It can be seen that the hash values obtained from both forms generated different font sizes. The output value of each character, thus, is different. It was found that the average time of the traditional hash function was 0.00004001-0.000156 seconds from the MD5, SHA1, SHA256 and SHA512 algorithms. As for the hash function that has been enhanced with the SXR algorithm, it was found that the average time taken from four algorithms is 0.002308 to 0.015632 seconds. Experimental results showed that the traditional hash function took less time to perform than the hash function that has been enhanced by the SXR algorithm, which spent approximately 50 times longer. This means traditional hash functions have good performance in terms of operational time. Moreover, in the test of the time spent in processing, results are less than 0.1 seconds [26]. In terms of actual usage, humans will not realize the difference in the processing time. The results are shown in Table 2.

Table 2. Examples of hash values obtained from traditional methods and the SXR algorithm

Algorithms	Hash Original	Hash Original + SXR	Hash Original (s)	Hash Original + SXR
MD5	ab524a4deecaa19f5499454c9d221f29	5d729251dc6ddcef3493f382c2c7fc2e	00004001.0	0.002308
SHA1	da8ead9e6f7fea38ccbbaa62af26260e2a1fb4ca	c532a07f0a05349c31a4cb49e78275590ebbd84f	0000.076	0.002764
SHA256	19a9d72db170c35aaf955ccf8e27b52130909b9df4300de9293ab1f4a7bd45b3	5bc7c22c0fec2ab895aaf7a6f1f81ea83088d7c763f68604cf31de9756a05c32	00.001	0.005514
SHA512	0e7262e20976cc756646abcad13089d5c3405244bf01dd1ed61ce35224b9d8e104c8663fd211f1327738edc818f9d188a8cc7f2dcaecbe43839066c4c827b88f	cf917f3df41f612a67c2a4d636bb08104c0ddaab02e5812298463bce2b4806993f9619769e4eb2b66e5e191615577dea9da175232634abdccf678203c8753560	00.00156	0.015632

4.3. Comparison of the effectiveness of resistance to attack

This research compared the resistance against attack between traditional hash algorithms and hash algorithms that enhanced security with the SXR algorithm by comparing the resistance to attack with Dictionary attack, Brute-force and Birthday attack. This attack-resistance performance experiment was tested by using the 100 passwords that were encrypted with traditional hash functions and hash functions that have been enhanced with the SXR algorithm is shown in Table 3.

Table 3. Comparison of the efficiency of resistance against Brute-force and Dictionary attack in test set 1–5

Algorithms	Hash Original + SXR (test set 1)	Hash Original + SXR (test set 2)	Hash Original + SXR (test set 3)	Hash Original + SXR (test set 4)	Hash Original + SXR (test set 5)
MD5	0%	0%	0%	0%	0%
SHA1	0%	0%	0%	0%	0%
SHA256	0%	0%	0%	0%	0%
SHA512	0%	0%	0%	0%	0%

The experiment result in Table 3 shows the comparison of the effectiveness of resistance to attack with Brute-force and Dictionary attack in traditional hash algorithms, and hash algorithms that enhanced security with the SXR algorithm of test set 1 to 5. The result showed 0% in all experiments because the usage of passwords and secret key to make the attack techniques and methods currently unable to calculate the decoding.

In addition, the results from the SXR algorithm had the same size and format as the normal encryption methods. It was even difficult to calculate the hash value of that data. Thus, the decrypting method of the SXR algorithm could be achieved by creating a new algorithm. In order to enter the information used to find the hash value, it must consist of two variables, passwords and secret codes which is the requirement to be searched using Brute-force technique. The efficiency of decrypting using the Brute-force attack can be presented as the Big O notation as follows:

$$O(n^2) \tag{5}$$

From the (5), the value $O(n^2)$ is derived from the Brute-force of the SXR algorithm which has the following steps:

Notation:

hash P	: Hash value of the password that the attacker desires to search from the SXR algorithm
P_{bf}	: The password assigned by the attacker to the initial value of the Brute-force
P_{ld}	: Password list that the attacker assigns as a search value with the Dictionary attack
SC_{bf}	: The secret code assigned by the attacker to the initial value of the Brute-force
SXR_h	: The results from applying the P_{bf} and SC_{bf} variables are calculated through (1)-(3).

Algorithm 1: Brute-force SXR Verification

```
Input:  a hashed password hash P
        a start password brute-force $P_{bf}$
        a start secret key $SC_{bf}$
Output:  true or false
Start
1: $P_{bf}$ ← char(00000000)
2: $SC_{bf}$ ← char(0000)
3: while (hash P =! $SXR_h$) do
4:      $P_{bf}$ ←  $P_{bf}$ + 1 bit
5:      for (i ←  0; i <= $127^8$; i++){
6:             $SC_{bf}$ ← $SC_{bf}$ + 1 bit
7:             $SXR_h$ ← SXRfunction ($P_{bf}$ , $SC_{bf}$)
8:             if ($SXR_h$ = hash P) then
9:                    return true
10:            end if
11:     end for
12: end while
Stop
```

The Brute-force SXR Verification was an algorithm for calculating the plaintext of passwords and secret codes. The attacker would bring the hash value from the attack (hash P) to find the plaintext of the password and secret code in the Brute-force method, by assigning the password from 8 to 12 digits. The initial value of the first loop is 00000000 and plus 1 bit each for the next loop. Then set the value of the secret code to 4 to 8 digits, starting at 0000 in the first loop, and plus 1 bit each of the next loop. The password and secret codes consisted of numbers, characters, special characters that are all possible in the range of 4 to 8 characters. The result was the password (P_{bf}) and the secret code (SC_{bf}). The dictionary attack method was the default word list to search for the hash value to find the secret code results from the SXR algorithm.

Algorithm 2: Dictionary attack SXR Verification
```
Input:   a hashed password hash P
         a start password lists of dictionary attack $P_{ld}$
         a start secret key $SC_{bf}$
Output: true or false
Start
1: $P_{ld}$   ← password list
2: $SC_{bf}$  ← char(0000)
3: i    ← 0
4: while (i < $P_{ld}$.size) do
5:       for (j ← 0; j <= $127^8$; j++) do
6:              $SC_{bf}$    ←  $SC_{bf}$ + 1 bit
7:              $SXR_h$    ←  SXRfunction ($P_{ld}$.[i], $SC_{bf}$)
8:              if ($SXR_h$ = hash P) then
9:                     return true
10:             end if
11:      i += 1
12:      end for
13: end while
14: return false
Stop
```

From the Dictionary attack SXR Verification algorithm, calculating the plaintext value of the secret code by the attacker could bring the hash value from the attack (hash P) to find the plaintext value of the secret code using dictionary attack by comparing the value from the password list by password, find the secret code in the same way as Brute-force method. The result was a password (P_{ld}) and a secret code (SC_{bf}). Then, the processing time during the attack had been analyzed as adopted from the method of Wenjian Luo [27], as shown in Table 4.

Notation:

• N_d	: the number of elements in a password list;
• N_{sc}	: the number of elements in a secret key list;
• N_p	: the number of passwords to be cracked;
• T_h	: the time spent on executing a cryptographic hash function;
• T_{ks}	: the time spent on executing a key stretching algorithm;
• T_{sxr}	: the time spent on executing a SXR algorithm;
• T_{m_hash}	: the time spent on determining whether two hash values match;
• T_{m_ks}	: the time spent on determining whether two passwords enhanced by a key stretching algorithm match;

Table 4. The Comparison of attack complexity

Schemes	Time complexity
Hashed password	$O(N_d * N_p * (T_h + T_{m\_hash}))$
Salted password	$O(N_d * N_p * (T_h + T_{m\_hash}))$
Key stretching	$O(N_d * N_p * (T_{ks} + T_{m\_ks}))$
SXR (*proposed algorithm)	$O(N_d * N_{sc} * N_p * (T_h + T_{sxr} + T_{m\_hash}))$

Table 4 hightlighted that the SXR method is more secure in preventing dictionary attack than other methods. Inclusively, the equation by adding 1 more secret code is considered the strength of this research. While existing techniques usually try to improve the hash function [18, 28], the proposed SXR technique focuses on the password storing. Enhancing security could be done by calculating the ratio that divides the hash value and the number of iterations. Unlike existing salted password approaches, they need to add character set to the password to increase strength, but also need to store this character set in the database [10, 11]. Our SXR is to use the username and password as the basis for improving the efficiency to increase the complexity of password, so the decrypting must consist of two parts.

Our technique neither intervenes nor modifies the hash algorithm. It strengthens the password after the hash function and keeps the password in the database. Thus, the proposed technique is applicable to any existing hash algorithm. The only disadvantage is that the encryption time is increased because the username and password are used to create a secret key. However, users will not be aware of the encryption time. The performance of the proposed technique is considered by using the processing time for a hash password. As per the experiment, the average time of the hash function with the SXR algorithm is 0. 002308 to 0. 015632 seconds. Although result has taken more time than the traditional hash function by 50 times, in actual use, the processing time is less than 0. 1 seconds. Humans will not realize the difference in the processing time [26]. As for the resistance against attack, the proposed technique can prevent 100% from Dictionary attack, Brute-force and Birthday attack.

5. CONCLUSIONS AND FUTURE WORK

At present, database threats come in many ways, and there are many reports of database attacks occurring. The severity of these attacks seems to be becoming more dangerous, whether it is the loss of data, property or distrust in the security criteria of the system or website. Consequently, the criteria or methods for storing passwords are considered to be a priority in system development. The method of storing passwords has many forms, such as encryption using the hash function. The main objective of the above methods are maintaining the security of a database so that the security system remains stable and reliable, and can verify the accuracy of the information and be able to prevent or report when there has been an unauthorized modification of data.

In this paper, we proposed SXR algorithm by calculating the username and password to get a secret key, then bringing them through the SXR algorithm and store the password into the database. Although the password attacker will get the password or may get into the database; it would not help to make it easier to randomly find the correct password, because the form of the password stored is the same as the password through the general hash function.

Experimental results have shown that the popular password that is introduced through the algorithm proposed with the MD5 algorithm can increase the security of the database. Even if the database is attacked, password attacker cannot decrypt the data, thus the data stored in the database is secure. This research can help in building credibility for web application developers or various identity authentication software among users. In future work, the authors plan to utilize in investment transactions for increasing security of the data with multi-factor authentication such as IP address, OTP, biometrics, etc., as well as to improve the SXR algorithm scheme with proper encryption in order to increase the attack resistance.

REFERENCES

[1] M. Abadi, “Security Protocols and Specifications,” Berlin, Heidelberg, pp. 1-13, 1999.

[2] B. A. Forouzan, “Cryptography and Network Security,” McGraw-Hill, Inc., pp. 1-480, 2007.

[3] B. Savege, “A Guide to Hash Algorithms,” GIAC Practical Repository, SANS Institute, 2003.

[4] A. Singh and S. Raj, “Securing password using dynamic password policy generator algorithm,” *Journal of King Saud University - Computer and Information Sciences,* 2019.

[5] T. Hunt, “Have I Been Pwned?” 2013. [Online]. Available: https://haveibeenpwned.com/PwnedWebsites.

[6] H. Kumar, et al., “Rainbow table to crack password using MD5 hashing algorithm,” in *2013 IEEE Conference on Information & Communication Technologies*, pp. 433-439, 2013.

[7] R. Rivest, “The MD5 Message-Digest Algorithm,” MIT Laboratory for Computer Science and RSA Data Security, 1992.

[8] U. S. Department of Commerce, “FIPS PUB 180-4, Federal Information Processing Standards Publication: Secure Hash Standard (SHS),” National Institute of Standards and Technology, CreateSpace Independent Publishing Platform, pp. 1-36, 2012.

[9] R. Morris and K. Thompson, “Password security: a case history,” *Communications of the ACM,* vol. 22, no. 11, pp. 594-597, 1979.

[10] S. Kharod, et al., “An improved hashing based password security scheme using salting and differential masking,” in *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, pp. 1-5, 2015.

[11] M. M. Kassim and A. Sujitha, “ProcurePass: A User Authentication Protocol to Resist Password Stealing and Password Reuse Attack,” in *2013 International Symposium on Computational and Business Intelligence*, pp. 31-34, 2013.

[12] P. P. Churi, et al., “Jumbling-Salting: An improvised approach for password encryption,” in *2015 International Conference on Science and Technology (TICST)*, pp. 236-242, 2015.

[13] J. Jeong, et al., “Enhancement of Website Password Security by Using Access Log-based Salt,” in *2019 International Conference on Systems of Collaboration Big Data, Internet of Things & Security (SysCoBIoTS),* pp. 1-3, 2019.

[14] M. D. A. Chawdhury and A. H. M. A. Habib, "Security enhancement of MD5 hashed passwords by using the unused bits of TCP header," in *2008 11th International Conference on Computer and Information Technology*, pp. 714-717, 2008.
[15] F. E. D. Guzman, et al., "Enhanced Secure Hash Algorithm-512 based on Quadratic Function," in *2018 IEEE 10th International Conference on Humanoid, Nanotechnology, Information Technology,Communication and Control, Environment and Management (HNICEM)*, pp. 1-6, 2018.
[16] F. E. D. Guzman, et al., "Implementation of Enhanced Secure Hash Algorithm Towards a Secured Web Portal," in *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*, pp. 189-192, 2019.
[17] K. T. Seong and G. H. Kim, "Implementation of voyage data recording device using a digital forensics-based hash algorithm," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 6, pp. 5412-5419, 2019.
[18] G. Sodhi, et al., "Implementation of message authentication code using DNA-LCG key and a novel hash algorithm," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 1, pp. 352-358, 2019.
[19] D. Dasgupta, et al., "Multi-Factor Authentication," in D. Dasgupta, et al. (eds), Advances in User Authentication, Cham, *Springer International Publishing*, pp. 185-233, 2017.
[20] R. D. Pietro, et al., "A two-factor mobile authentication scheme for secure financial transactions," in *International Conference on Mobile Business (ICMB'05)*, pp. 28-34, 2005.
[21] S. Subrayan, et al., "Multi-factor Authentication Scheme for Shadow Attacks in Social Network," in *2017 International Conference on Technical Advancements in Computers and Communications (ICTACC)*, pp. 36-40, 2017.
[22] R. G. Rittenhouse and J. A. Chaudhry, "A Survey of Alternative Authentication Methods," *International Conference on Recent Advances in Computer Systems (RACS 2015)*, pp. 179-182, 2015.
[23] D. Dasgupta, et al., "Adaptive Multi-factor Authentication," in D. Dasgupta, et al. (eds), Advances in User Authentication, Cham, *Springer International Publishing*, pp. 281-355, 2017.
[24] W. Ma, et al., "Password Entropy and Password Quality," in *2010 Fourth International Conference on Network and System Security*, pp. 583-587, 2010.
[25] John Hall, "SplashData's Top 100 Worst Passwords of 2018," 2018. [Online]. Available: https://www.teamsid.com/splashdatas-top-100-worst-passwords-of-2018/.
[26] R. B. Miller, "Response time in man-computer conversational transactions," *The Proceedings of the fall joint computer conference, part I*, San Francisco, California, pp. 267-277, 1968.
[27] W. Luo, et al., "Authentication by Encrypted Negative Password," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 1, pp. 114-128, 2019.
[28] A. Abouchouar, et al., "New concept for cryptographic construction design based on noniterative behavior," *International Journal of Artificial Intelligence (IJ-AI)*, vol. 9, no. 2, pp. 229-235, 2020.

BIOGRAPHIES OF AUTHORS

Jakkapong Polpong was born in Yala, in 1988. He received a Bachelor of Science in Computer science from faculty of Applied Science, King Mongkut's University of Technology North Bangkok, in 2009. He recently received a Master of Science in Information Technology, King Mongkut's University of Technology North Bangkok, in 2014. At present, he is a Ph.D. Candidate in Information Technology and Digital Innovation at KMUTNB.

Dr. Pongpisit Wuttidittachotti is currently an associate professor and head of the Department of Data Communication and Networking at the Faculty of Information Technology and Digital Innovation, King Mongkut's University of Technology North Bangkok (KMUTNB), Thailand. He received his Ph.D. in Networks, Telecommunications, Systems and Architectures from INPT-ENSEEIHT, in France. He received an outstanding employee award in social service at the university level in 2019, an outstanding employee award at the faculty level and the university level in 2020. He owns more than 30 recognized certifications, for example, CISSP, CISM, CISA, CRISC, CGEIT, IRCA ISO/IEC 27001:2013 Lead Auditor, COBIT 5 Foundation, COBIT 2019 Foundation, COBIT 2019 Design & Implementation, Data Protection Officer (DPO) etc. So far, Wuttidittachotti has over ten years of working experience covering software development, network, security, audit, risk management, IT governance, and standard, and compliance. His expertise has shown out as a member of the ISACA Bangkok Chapter committee since 2015, and an Accredited Trainer - COBIT® 2019 Foundation for ISACA Bangkok Chapter. He has conducted and published many research articles continually in information security and related topics.