IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 19, 2020, accepted December 29, 2020, date of publication January 4, 2021, date of current version January 22, 2021.

Digital Object Identifier 10.1109/ACCESS.2020.3049052

A Simple and Secure Reformation-Based Password Scheme

MUSHTAQ ALI[1], AMANULLAH BALOCH[1], ABDUL WAHEED[1,2], MAHDI ZAREEI[3], (Senior Member, IEEE), RIMSHA MANZOOR[1], HASSAM SAJID[1], AND FAISAL ALANAZI[4]

[1]Department of Information Technology, Hazara University Mansehra, Mansehra 21120, Pakistan
[2]School of Electrical and Computer Engineering, Seoul National University, Seoul 08826, South Korea
[3]Tecnologico de Monterrey, School of Engineering and Sciences, Zapopan 45201, Mexico
[4]Department of Electrical Engineering, College of Engineering, Prince Sattam Bin Abdulaziz University, Al Kharj 16278, Saudi Arabia

Corresponding author: Faisal Alanazi (faisal.alanazi@psau.edu.sa)

This publication was supported by the Deanship of Scientific Research at Prince Sattam bin Abdulaziz University, Alkharj, Saudi Arabia.

ABSTRACT The electronic applications of financial institutions like banks and insurance companies use either token-based, biometric-based, or knowledge-based password scheme to keep the confidential information of their customers safe from hackers. The knowledge-based password scheme's resistance, particularly its reformation-based password scheme against shoulder surfing attacks, is comparatively better than the other two because its password can be entered in crowded places without fear of shoulder surfers. However, the available reformation based passwords involve mental computation making their usability difficult. Furthermore, they also need an extra device like earphones during password entry causing to create a gap for information leakage. Moreover, most of the passwords store passwords' actual content on a server database that causes penetration in the financial institutions' database. In this article, a reformation-based password scheme involving no mental computation and using no extra device is proposed. The proposed scheme works on the password characters' indices, which change dynamically after each login process. It gets the password characters' indices from the end-user and obtains his password characters' indices from the database. Next, the textual passwords are formed from the user-provided indices and those obtained from the database. The textual passwords are then compared, and if found match, then login is succeeded, otherwise failed. Our proposed password scheme's experimental results on the password data set showed better security and usability compared to state-of-art password schemes.

INDEX TERMS Password, brute force attack, shoulders surfing attack, authentication system, usability, security.

I. INTRODUCTION

The financial institutions like banks and insurance companies need security methods like a password for keeping their information secure from hackers [1], [2]. The available password methods are broadly categorized into three types [3], [4]: (i) token-based/possession (ii) biometric-based, and (iii) knowledge/text-based. Diagrammatically they are shown in Fig. 1. The token-based scheme is not reliable because the security key information printed over the smart card can be stolen and misused [10]. The biometric authentication is also unreliable because dummy thumbprints can be formed from different objects like glass and metal when touched and misused by adversaries [10], [19]. Moreover, both the token and biometric-based schemes require additional hardware(s), making them expensive and complicating their usability [14]. The most popular and widely used password scheme is the knowledge-based scheme [19], which is further subdivided into two subcategories, graphical-based and alphanumeric character set based (text-based). The graphical-based password scheme (subdivided into recognition based and recall based) involves a mouse password entry. The mouse's use in the recognition-based schemes causes the vulnerability of shoulder surfing (SS) attack [5], [20], [21]. In contrast, recall-based schemes are a little secure due to pixel selection in an image rather than objects. However, memorization and selection of the exact position on the image are difficult in these schemes [25].

The associate editor coordinating the review of this manuscript and approving it for publication was Chien-Ming Chen.

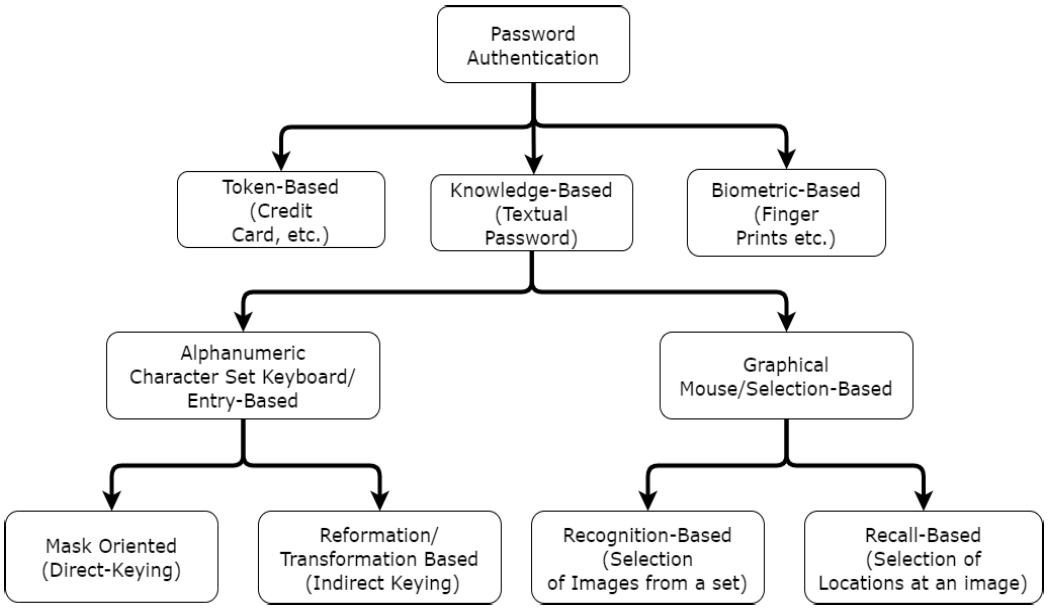

FIGURE 1. Types of password schemes.

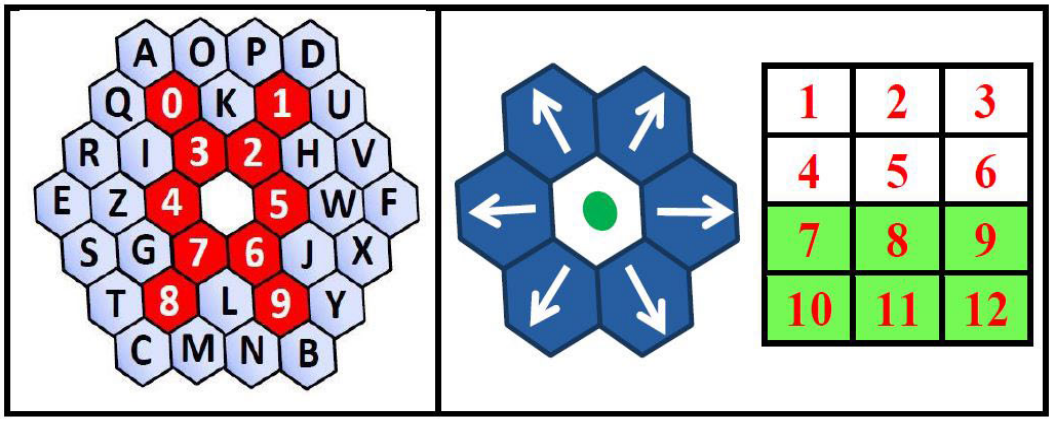

FIGURE 2. Login interface of reformation-based password scheme [5].

The scheme based on the alphanumeric character set is further subdivided into two types: (i) direct keying password scheme (mask-oriented conventional password scheme); and (ii) reformation based password scheme(Indirect keying password scheme or transformation-based schemes). The direct keying is vulnerable to shoulder surfing, and brute force attacks [19], while the reformation based password scheme proposed in [9], [11]–[13], [37], [39] show comparatively better resistance to these attacks. However, these reformation based password schemes achieved client-side security at the cost of losing usability. For instance, the extra keystrokes, i.e., several keys are pressed for entering single characters [5], [9], [11], user computation [5], [13], entering metadata [12], [13], using large character set [5], [10], [37], [39], and use of additional device like earphone [5], [21] are required for completing the login process. The use of the extra device(s) complicates usability and acts as a threat to client-side security if used in crowded places. For example, the reformation based password scheme's user interface [5] is shown in Fig. 2. This scheme needs an extra earphone device to communicate the challenge password's characters during the login process. Suppose the shoulder surfer succeeds to hack each of the challenge characters via sensors (i.e., eavesdropping microphone listener [26]) along with the entered directions and distances. In that case, the original password characters can be traced using the password entry process's reverse actions. For instance, 'D' is the password character stored in the database, and 'S' is its respective challenge character given to the user via earphone. During the login process, the user needs to press $\leftarrow$ 2 $\swarrow$ 4 characters in place of entering 'D', and the shoulder surfer traces 'D' by pressing $\nearrow$ 4 $\rightarrow$ 2, which is the reverse action starting from

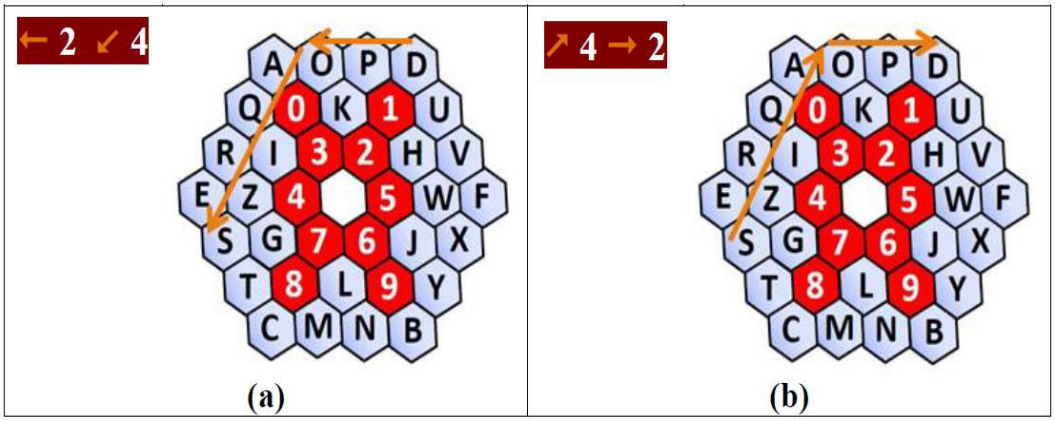

FIGURE 3. Vulnerability of [5] to shoulder surfing attack a) entry of original password character 'D', with challenge character 'S' b) tracing of original password character 'D' using the leaked challenge character 'S'.

the challenge character 'S' as has shown in Fig. 3. The same procedure is applied to finding the other password characters. Once the shoulder surfer traces the password characters, he/she can login to the system [5] by using the earphone as an original user. Moreover, its password entry time was four times the cost of a traditional text-based password scheme. The user must also estimate the distance between the password character and the challenging character, which is not convenient for the user and slows down the system efficiency. Additionally, this scheme is not beneficial for the deaf person. Besides usability and client-side security problems in the reformation based password scheme, no server-side security technique is applied in these schemes [5], [9]–[13], [25], [39].

To overcome the limitations as mentioned above, i.e., complex usability, weak client-side security, and lack of server-side security of the state-of-art reformation based password scheme [5], [21], a password scheme providing strong client and server-side security with better usability without using the extra device is proposed in this research work. The proposed scheme works as follows: The proposed scheme displays a matrix containing 42 password characters. The user needs to enter the columns' numbers containing password characters instead of the actual password characters during the login process. The transpose of each of the entered columns' numbers is then computed and placed into the transpose matrix in order. The order of the columns in the transposed matrix is changed after each login process. The transpose matrix is then displayed, and the user is asked to enter the columns' numbers containing password characters. Next, the proposed scheme picks the original password characters based on the previously entered columns' numbers. After this, the numeric password (NP) representing the password characters is picked from the password database for the given user Id. The picked numeric password is then decomposed into two equal parts. The first and second parts of the numeric password denote the rows and columns the indices of the password characters stored in the database. The corresponding digits in both parts of the numeric password are then used for constructing the textual password. The textual passwords formed on both client and server sides are then matched. If both match, then the login process complete successfully; otherwise, the user is asked to enter the password again.

A. CONTRIBUTIONS

Our contributions in this research work are as follows:

1) We identified two research gaps in the available reformation based password schemes: i) they improved client-side security at the cost of usability, ii) their server-side security is not satisfactory.
2) We introduced a reformation-based password scheme that improves client security without losing usability and improves the server-side database security.
3) We introduced a brute force algorithm for evaluating our proposed password scheme's security strength against the brute force attack both on the client and server sides.

B. ARTICLE ORGANIZATION

The rest of this article is organized as follows. Section 2 is about the literature review, section 3 explains our proposed password scheme, section 4 presents the experimental results, and section 5 describes the conclusion and future work.

II. LITERATURE REVIEW

This section presents the available reformation based password schemes [5], [6], [9]–[13], [21], [37]–[39], graphical password schemes [7], [23]–[25], and their pros and cons. Besides, it also presents the brute force algorithms [15]–[17] used for cracking the passwords.

A. REFORMATION BASED PASSWORD SCHEMES

In [5], a knowledge-based password authentication scheme was proposed. The proposed scheme needs the password characters with their respective challenge characters at the

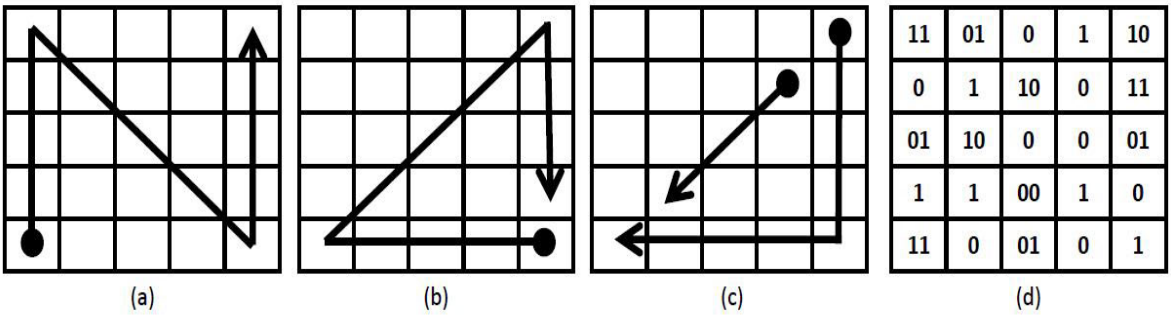

FIGURE 4. Registration and login interface [9]: (a-b) login pattern of user with one stroke (c) login pattern of user with two strokes (d) grid with input characters.

time of login. An extra earphone device is needed for communicating the challenge characters to the user. If the challenge characters leak out, then the original password characters can be traced by applying the reverse action of password entry, as discussed in the introduction section. Moreover, its password entry time is four times the cost of a traditional text-based password scheme. The user also needs to estimate the distance between the password character and challenging character during the login process, which is not convenient for the user and slows down the system efficiency. This scheme performs a reformation-based method only on the client-side, while on the server-side, the original contents of the password are stored.

In [6], a new PIN entry scheme was introduced. The scheme was designed to protect the user against shoulder surfing attacks. The scheme consists of 36 locations in a 6×6 matrix form. The user enters the positions/locations instead of a textual password. The user is required to memorize the static coordinates of the security PIN code. The four static positions PIN code of each user is stored in the database with eight digits (i.e., each position is represented with one row and one column coordinate digit). The required position is selected by entering the number present at the first location of the PIN code. To mislead the shoulder surfer, the same number is placed at six different locations of the interface. This scheme showed promising results against weak shoulder surfing, but it is vulnerable to a strong shoulder surfing attack. Moreover, its server-side security is not protected. Additionally, the user needs to interact with four user interfaces, which is tedious and time-consuming.

In [9], a stroke-based textual password authentication scheme was proposed. For logup, the user draws pattern like geometrical shape as shown in Fig. 4(b-c) or pattern like a letter, a symbol, or a digit as shown in Fig. 4(a) by selecting the order sequence of locations on a 5×5 grid and is stored as his/her password. For login, the user inputs the characters present at the grid locations assigned to him/her as his/her password pattern. For example, a user with the pattern shown in Fig. 4(b) using characters of Fig. 4(d) will enter his/her password string like "10100111001011010" while the user with the pattern shown in Fig. 4(c) using characters of Fig. 4(d) will enter his/her password string like "10110101001011001". If the input characters match with characters present in the user password pattern, access to the system is granted. This scheme is resilient to brute force attacks. However, it is vulnerable to shoulder surfing and hidden camera attacks due to the grid's fixed location and similarity in the input string pattern. Moreover, the login process is complicated for the user to remember the pattern, particularly when the strokes are included in the password pattern. It also needs a long password space, which creates difficulty for the user during the login process. The sequence of the same consecutive characters can confuse and mislead the user during password login entry. Furthermore, if the database is hacked, the intruder can login as an actual user because the password pattern positions remain static on the server-side.

In [10], a text-based authentication scheme was introduced. Nine to ten different characters are attached to each numeric digit. The user presses the digits against the corresponding password characters. The scheme confuses the different attacks due to multiple options available against each digit. It also creates difficulty for the user to search his/her character and press the current digit assigned to that password character. The login process becomes complicated for a long password. Moreover, the scheme is vulnerable to a strong shoulder surfing attack provided that recording of two times the login process is available to the shoulder surfer. For example, the user password is "bdg", the user enters digit 7 for his/her first password character b. The digit 7 is assigned to the list of ten characters in the first session interface: $b, z, E, J, U, 9, :, [, \{$. In the next session, users enter the digit 8 for their first password character b. The digit 8 is assigned to list of ten characters in the second session interface that are: $b, m, P, D, H, L, M, @, \_, \}$. The character b is common in both interfaces and could be extracted for breaking the password. In this way, other password characters could also be extracted by constructing each password character's two lists. This scheme secures the server-side database password by storing the encrypted password, but it is vulnerable to recording-based shoulder surfing. Furthermore, its usability is difficult because the user needs to search for a password character in an interface containing many characters.

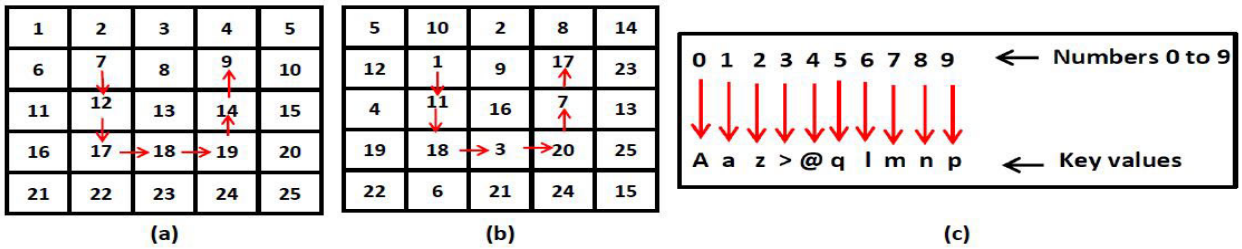

FIGURE 5. Registration and login interface of article [11]: (a) login pattern of user with cell locations (b) login entry of user password against his/her pattern locations (c) list of transformation characters against input digits.

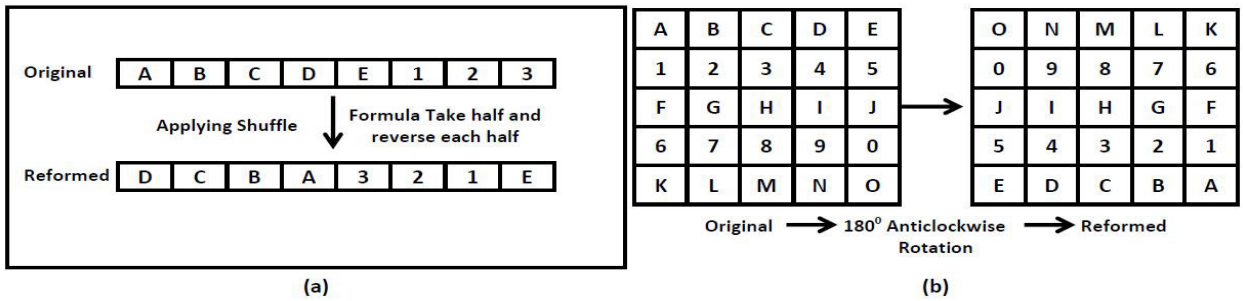

FIGURE 6. Password Reformation process of article [12]: (a) one-dimensional password reformed with shuffling (b) two-dimensional password reformed at 180° rotation.

In [11], a 5 × 5 grid is the interface of user password entry, as shown in Fig. 5(a). In this grid, the user registered by selecting an order sequence of grid points and making a pattern as his/her password. For login, the user has to make a string of numeric numbers located at the user password grid points in the proper order of password pattern, as shown in Fig. 5(b). Digits of this numeric string are further mapped with ten digits key, as shown in Fig. 5(c), and another string of letters, digits, and symbols is constructed, which is considered a user password. However, the usability of this scheme is better and resists well against brute force attacks. However, this scheme is vulnerable to weak and strong shoulder surfing attacks because the grid locations of user passwords are static, and after two or more sessions, they can be traced.

In [12], a reformable one-dimensional password scheme and two-dimensional password scheme were proposed, which are shown in Fig. 6(a) and in Fig. 6(b), respectively. This scheme registers the user by storing his/her (RP) and reformable information, i.e., shuffling technique or rotation angle in one-or two-dimensional password, respectively. The server-side password is retransformed into its original shape at the login time using metadata (shuffling/rotation angle information) and compared with the user-entered password for granting the authentication. A one-dimensional password string has only one variation at 180° for shuffling the password string. In contrast, the two-dimensional password has three possible rotations angles 90°, 180°, and 270°, which is also easy for the intruder to try the password with three angles one by one. This scheme is resilient to different attacks (i.e., brute force attack, key logger) in restricted login entry (i.e., using one of the three possible rotations). Moreover, it also resists shoulder surfing attacks. However, if someone hacks the server-side database and shuffling/rotation angle information, the password can be easily retransformed to its original shape, and security could be a break.

In [13], the *S3TFPAS* scheme was introduced. In this scheme, the user registered himself with three kinds of information: a) Three letters password, b) An arithmetic expression for calculating the token value and c) Select an arithmetic operator ($*$, $+$, $-$) for operating the password numeric value and token value. This scheme is resistant to brute force, hidden camera, shoulder surfing, and a dictionary attack on the client-side. However, this scheme's usability factor is deficient because of the high cognitive burden in login entry. This is because the user needs to memorize each password character's numeric number and memorize the arithmetic operation equivalent to each character mentioned in the token. Additionally, the user also needs to perform an arithmetic operation in mind for making login value. Furthermore, if the password letter with the additional metadata information is hacked from the server-side, it can easily be used from the client-side to break the security. In [21], a scheme for mobile security against shoulder surfing attack was introduced. In this scheme, an additional device (earphone) is required for communicating the challenge character. The additional device causes a) security risk, i.e., it may increase the chances of vulnerability, b) affect a large deaf community, i.e., the latest report of deafness [26] shows that roundabout 466 million persons in the world are deaf, and they will directly affect due to this limitation of the scheme. The user

counts clockwise and then responds to the target character against his/her corresponding password character. If the user missed his/her track and counted the characters in the wrong circle after the wrong entry, he can press the BACK button and respond to the target character in the right track. This facilitation of the BACK button helps the user correct his/her entry. However, it also shows the scheme's usability factor's weakness: i) the scheme is not user-friendly ii) re-entry of the target character increases the user entry time. Thus the scheme loses the usability factor by introducing the additional device and putting the extra burden of counting for each character entry.

The password of the scheme proposed in [37] is composed of two parts, i.e., static part (text) and dynamic part (system time in the form of hour and minutes). The combination of static and dynamic part is then used as a user password. The machine extracts both the static and dynamic parts from the password. The dynamic part of the entered password is then compared with the hours and minutes section of the current system time, and its static part is compared with the encrypted static part of the user password in the password database for password verification. The login process is succeeded if both the static and dynamic parts of the entered password match those present in the password database.

This scheme provides strong client-side password security if the static part of the password is designed using the whole character set, i.e., 95 printable characters. However, the use of the whole character set makes the usability of the scheme difficult. Besides complex usability, memorization of the locations of the dynamic part of the password is also needed. Furthermore, the password containing less than 10 characters is vulnerable to brute force attacks and shoulder surfing attacks. In addition to the limitations mentioned above, the user often needs to wait during the login process if the minutes of system time is shifting to the next minute, i.e., 12:32:58. For instance, if the user enters Bot12net32 as his/her password at system time 12:32:58, this scheme extracts the dynamic part from the user password and gets the system time, which has been changed 12:33:02. In this specific instance, the dynamic part of the user-entered password does not match the system time, and the system will display the message of the invalid password.

In [38], a mutual user authentication scheme is introduced for communicating information securely between client and server in specific application domain like healthcare centers so that their patient's information could not be leaked or tampered.

In [39], two types of authentication mechanism were introduced, i.e., i) Matrix-based authentication and ii) Sector-based authentication. i) In matrix-based authentication mechanism, database authentication table contains the three information of each user: i) user identity, ii) actual password, and iii) password character selection order, i.e., whether the user has to enter the row index or columns index for specific password character. A character matrix with a dimension 10×10 having all capital and small alphabets, numeral digits, and special symbols are regenerated after each password character entry during a login session. The specific password characters are placed in the character matrix in the particular row or column as per character selection order, and the system notes its corresponding row or column; the rest of the characters are placed randomly in the character matrix now if the user enters the correct index of that password character then the second screen with second password character is given to the user and so on till the entry of last password character. This password entry procedure will affect the usability factor, and multiple screens will engage the user's mental attention. The system is given the client-side security to some extent, at the cost of increasing usability. Moreover, the system stores the actual password and its metadata, i.e., the password character selection method in the database. This storing of actual data insecure the database on the server-side because, if it is hacked and information is leaked, the whole database will become insecure. ii) In sector-based authentication mechanism, database authentication table contains the two information of each user i) user identity, and ii) actual password. Character matrix with 5 sectors having all capital and small alphabets, numeral digits, and special symbols is regenerated after each password character entry during a login session. The specific password characters are placed in the character matrix in the particular sectors, and the system notes its corresponding sector; the rest of the characters are placed randomly in the character matrix, now; if the user enters the correct sector of that password character, then the second screen with second password character is given to the user and so on till the entry of last password character. This password entry procedure will also affect the usability factor, and multiple screens will engage the user's mental attention. Moreover, this system improved the client-side security at the cost of an increase in usability, but it stores the password database's actual password, making its server-side security low.

B. GRAPHICAL PASSWORD SCHEMES

In [7], a graphical password authentication system was proposed. This system was designed to protect users against weak as well as strong shoulder surfing attacks. The proposed system uses a database of different icons, where the user selects a few icons as a password. The user also selects a ball of specific color out of five balls of different colors. The chosen color ball is used to authenticate a password icon as and when the ball moves over the icon. Although the system resists against both weak and strong shoulder surfing attacks, it takes high time cost during the authentication process, and the user has to wait for a dynamic moving ball to coincide over his/her required password icon so that the user confirms his/her password icon by pressing the space key or any other key of the keyboard. It shows that the scheme is not suitable for a real-time system. A graphical-based textual password scheme was introduced in [8]. For each password character entry, the user has to click inside the area of an imaginary

triangle made up of three vertices of the user's password characters. The scheme is resilient to shoulder surfing attacks, but the login process is lengthy, complicated, and difficult for successful login entry. The scheme is not suitable when the user has to enter the password frequently after every short interval, as in a mobile phone.

In [20], a two-steps login entry scheme was introduced. This scheme is vulnerable to shoulder surfing attacks and brute force attacks.

A unique idea based on camouflage strategy is introduced in [22]. This is a good idea to deceive the shoulder surfer, but it also engages the user on the screen and waits to activate his/her required key and audio input to confirm this key as a login entry.

In [23], a uniform security metric for measuring the security of the cognition-based graphical password scheme was introduced. The security attacks can be categorized into guess ability (social engineering attack), observability (shoulder surfing and intersection attacks), and recordability (replay or man-in-the-middle attack). The evaluation approach maintains four tuples of information for a scheme against different attacks and analyses its security level based on the collected information. The score level is maintained for a scheme against four tuples of attacks, i.e., distractor, shoulder surfing, intersection, and replay.

In [24], a hybrid graphical authentication scheme was introduced. This scheme combines the choice-based, click-based, and draw-based input responses as password entry. It gives the user options to select two images from a set of images, click some grid points in 3×3 grids as a password, and finally draw a grid pattern. All these three inputs validate a user. Although the scheme provides security against brute force and dictionary attack, it is vulnerable to shoulder surfer due to mouse clicks for image and points selection and to draw the pattern. In [25], the first recall-based graphical password scheme was introduced by Greg. E. Blonder. In this scheme, the user selects some position points on an image for password entry. This scheme was a step from a text-based password to a graphical password. It gave protection against brute force and dictionary attack, but the scheme's usability is difficult due to memorization and selection of exact positions on the image.

C. BRUTE FORCE ALGORITHMS

In [15], a brute force algorithm for searching a string pattern was introduced. It was an excellent attempt to search a pattern efficiently by ignoring the middle section of the string, but it will give a better result if the pattern length is more significant than four characters. The algorithm will compare the first and last characters for four characters, but the time saved against the rest of the two characters is less than the time consumed for multiple if-else statements. Moreover, if the first and last character matched for a long string, the rest does not need to match. Instead, they will require further comparison for the rest of the characters. Suppose any mismatch occurred in the central characters. In that case, the substring will be ignored, and the algorithm will move onward, e.g., the first and last characters of the strings ''introduced'' and ''interested'' are matched, but the central characters mismatch.

The method introduced in [16] is an advanced version of [15], which compares the first, last, and middle characters of the pattern in the text.

In [17], different techniques have been proposed to avoid the brute force attack. For instance, the user account should be locked after a specified number of attempts. The password authentication should be allowed only for one time; a time-bound should be set for each user to login, a query should be asked from the user for providing access, the unique IP address should be used for login entry, the *CAPTCHA* testing layer should be fixed before granting access. An appropriate block method can be applied to provide security to the user account against the brute force attack. The interested readers are referred to see [28]–[34] for more details about brute force algorithms.

The aforementioned reformable approaches improved the security using extra devices [5], [21], high cognitive overhead [5], [9], [12], [13], multiple interfaces for each character of password [6], focusing on the client-side and not providing server-side database security [5]–[7], [10]–[13], [21]. However, they are losing usability due to complex interfaces as in [5], [9], [10] and extra keys entry against each password character [5], [9], [11], [12]. Additionally, the use of different devices (earphones) in [5], [21] acts as a threat to security if its sound is leaked out.

In contrast to available approaches, our proposed scheme does not use any additional device and puts a low cognitive burden on the user. Additionally, the textual passwords need not be stored in the password database, ensuring server-side database security. The proposed scheme uses dynamic interfaces for ensuring high security without losing usability.

III. PROPOSED PASSWORD SCHEME AND BRUTE FORCE ALGORITHM

In this section, we present our proposed password scheme. The proposed scheme performs two major processes: a) logup and b) login. In each of these two processes, two queues are created for the same purpose, i.e., for holding the row and column indices of the password characters. However, in each process, different sources are used for populating these queues. The contents of these two queues are then combined into the third queue. The corresponding indices in the first and second half of the third queue represent the row and column indices of the password characters. Besides our proposed password scheme, the working of our proposed brute force algorithm used to evaluate our proposed password scheme's security is also presented in this section.

A. PROPOSED PASSWORD SCHEME

The architecture of our proposed password scheme is comprised of five main modules: 1) logup, 2) login, 3) password verification, 4) update password, and 5) forgot password. The flow diagram of the first three modules is shown in Fig. 7, and

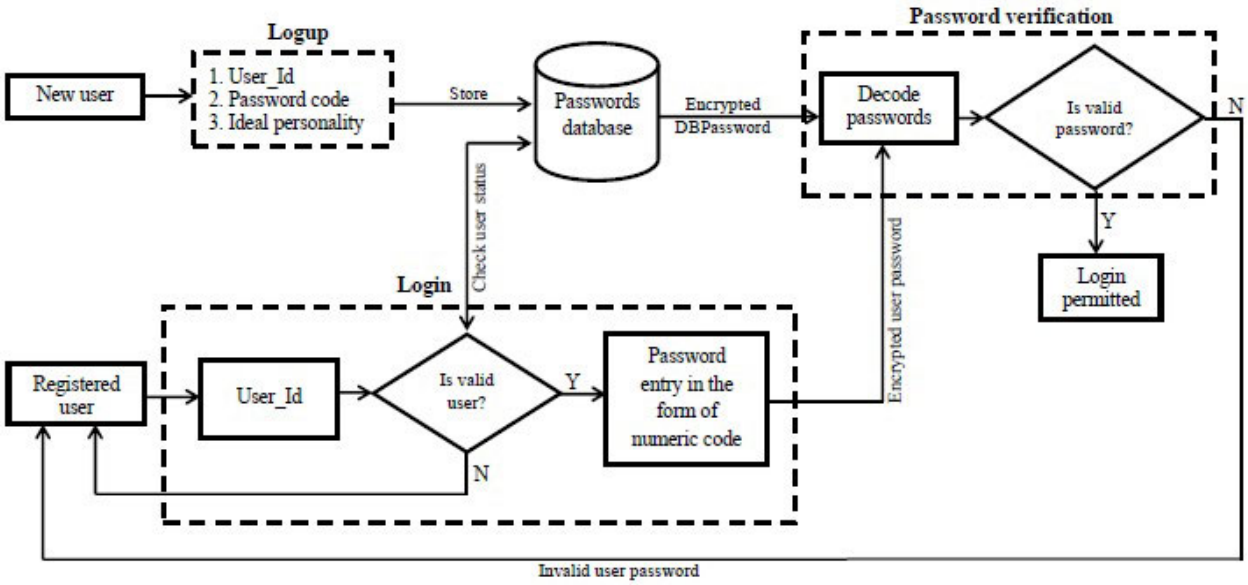

FIGURE 7. Flow diagram of the first three modules of our proposed password scheme.

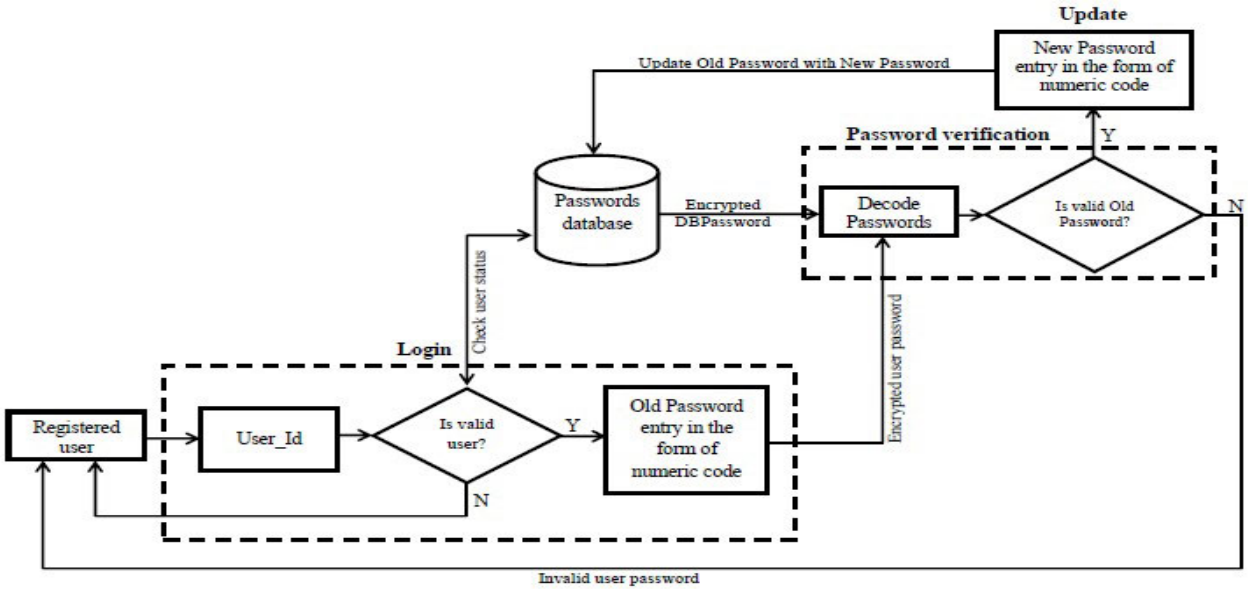

FIGURE 8. Flow diagram of password update.

that of the other two modules are shown in Fig. 8, and Fig.9, respectively. In these figures, each module is represented in the form of dashed rectangles.

1) LOGUP

This module aims to generate a numeric password containing the row and column indices of the user password for the given user id and stores both the user id and password numeric code for each user in the password database. Storing the numeric codes of passwords instead of textual passwords in the password database can protect the passwords from different attacks. This is because the attackers can easily steal the stored textual password, while in the case of stored numeric code of a password, even if the attackers steal the numeric code of a password, they cannot restore the textual password from it. For generating this kind of numeric code, this module uses a matrix containing password characters. We named this matrix a character-set matrix, which is shown below.

$$\begin{bmatrix} A & B & C & D & E & F \\ G & H & I & J & K & L \\ M & N & O & P & Q & R \\ S & T & U & V & W & X \\ Y & Z & 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 & 8 & 9 \\ @ & \# & \$ & \& & \_ & ! \end{bmatrix}$$

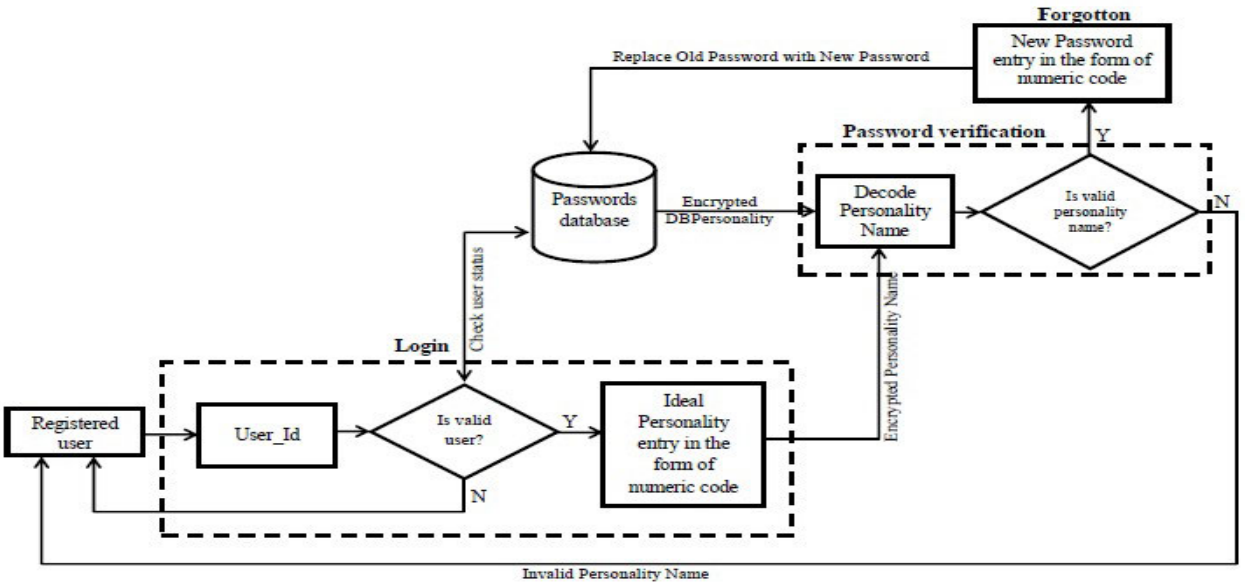

FIGURE 9. Flow diagram of forgotten password.

Most of the passwords contain some alphabetic characters, both in small and capital letters, numeric characters, and special characters. Thus, based on our observation, we organized all alphabetic characters, all numeric characters, and some of the special characters used most often in forming a password in a character-set matrix.

After getting this matrix, its columns are then labeled with column numbers. The user is asked to enter the column numbers containing password characters from the labeled character set matrix.

$$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \end{matrix}$$
$$\begin{bmatrix} A & B & C & D & E & F \\ G & H & I & J & K & L \\ M & N & O & P & Q & R \\ S & T & U & V & W & X \\ Y & Z & 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 & 8 & 9 \\ @ & \# & \$ & \& & \_ & ! \end{bmatrix}$$

The entered column numbers are then stored in the first queue. After this, the column-wise transpose of the character set matrix is taken using Eq. (1), and the resultant values are stored in the completely transposed matrix shown below.

$$\begin{bmatrix} A & G & M & S & Y & 4 & @ \\ B & H & N & T & Z & 5 & \# \\ C & I & O & U & 0 & 6 & \$ \\ D & J & P & V & 1 & 7 & \& \\ E & K & Q & W & 2 & 8 & \_ \\ F & L & R & X & 3 & 9 & ! \end{bmatrix}$$
$$B_{k,r} = \text{Transpose}(A_{r,c}) \tag{1}$$

where, $r = 1, 2, 3, \cdots, m$, $k = 1, \cdots, L$ (length of password), and c refers to selected column number.

The column number in the queue becomes the row indices of the required password characters in the complete transposed matrix due to transpose operation. The proposed password scheme has information regarding the row indices of the required password characters in the first queue. However, it still needs information about the column indices of the required password characters. Therefore, for getting column indices of the required password characters, the labeled version of the complete transposed matrix is created, shown below. The user is asked to enter the column numbers containing password characters from it.

$$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix}$$
$$\begin{bmatrix} A & G & M & S & Y & 4 & @ \\ B & H & N & T & Z & 5 & \# \\ C & I & O & U & 0 & 6 & \$ \\ D & J & P & V & 1 & 7 & \& \\ E & K & Q & W & 2 & 8 & \_ \\ F & L & R & X & 3 & 9 & ! \end{bmatrix}$$

The entered column numbers are then stored in the second queue. These column numbers in the second queue represent the column indices of the required password characters. Next, the row and column indices stored in the first and second queues are combined into the third queue. This numeric code in the third queue refers to both the row and column indices of each character of the required/user password. The registered users' database is then opened, and the user id of the new user and his/her password in the form of numeric code is stored in it. Thus, the users' passwords are stored in the password database in numeric codes containing row and column indices of the password characters. This numeric code is significant for security, and the original password

characters can be quickly restored from it. Algorithm 1 is used for implementing the above-stated logup procedure.

Algorithm 1 Logup_Module(A,UI)

1) A_1 = Label Matrix-A with columns numbers
2) Q_1 = Enter columns of A_1 containing password characters:
3) B = **TransposeMatrix**(A, Q_1)
4) B = Label Matrix B with columns numbers
5) Q_2 = Enter columns of B containing password characters:
6) Q_P = Concatinate(Q_1, Q_2)
7) Q_1 = Enter columns of A_1 containing Ideal Personality Name characters:
8) B = **TransposeMatrix**(A, Q_1)
9) B = Label Matrix B with columns numbers
10) Q_2 = Enter columns of B containing Ideal Personality Name characters:
11) Q_N = Concatinate(Q_1, Q_2)
12) Pt = Open(RegistrationTable)
13) Append(Pt, UI, Q_P, Q_N)

Whereas, A, UI, A_1, Q_1, B, Q_2, Q_P, Q_N and Pt are character set matrix, User_Id, character set matrix with column labels, queue having the first half of numeric code, transpose of character set matrix with column labels, queue having the second half of numeric code, queue with the combination of two halves password code, queue with the combination of two halves ideal personality name code and pointer for password file respectively. For instance, if a user has a user id 12 and his/her password is ''CAMAL'', then using the above-stated procedure, this module will populate the queues Q_1 and Q_2 as follows:

$Q_1 = 12345$
$Q_2 = 11312$

The contents of these two queues are then combined into Q, i.e., $Q = 1234511312$. This code reflects that individual characters like C, A, M, A, L of this password are located in the following pairs of indices of the transposed matrix.

(1, 1), (2, 1), (3, 3), (4, 1), (5, 2), whereas the actual indices of the individual characters of this password in front of the user are: (1, 3), (1, 1), (3, 1), (1, 1), and (2, 6), which are different from the system computed indices. The subset of user ids and their respective numeric passwords taken from the password database created using the above specified method is shown in Table 1. The textual passwords for the numeric passwords in Table 1 are *HI*, *ALI*, *DEAR*, *HARIS*, *IMTIAZ*, and *MUSHTAQ*, respectively. However, they are not stored in the password database.

2) LOGIN

The aim of this module is two-fold. Firstly, it checks the given user id's validity, and secondly, it generates a numeric code equivalent to the user password and passes it to the password verification module. Thus, this module comprises two sub-modules: checking the user id's validity and generating a numeric code equivalent to the user password during the login attempt. These two sub-modules are explained as follows.

a) Checking the validity of user id: This sub-module loads the registered users' database and matches the user id with each user id present in the registered users' database. If no match is found, then an invalid user id message is displayed. In the case of a valid user id, the next sub-module, i.e., ''Generating numeric code equivalent to the user password'', is called. A matrix containing password characters and user id is passed. Algorithm 2 implements the method of this sub-module.

Algorithm 2 User_Status(UI)

1) status = 0
2) $A = \begin{bmatrix} A & B & C & D & E & F \\ G & H & I & J & K & L \\ M & N & O & P & Q & R \\ S & T & U & V & W & X \\ Y & Z & 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 & 8 & 9 \\ @ & \# & \$ & \& & \_ & ! \end{bmatrix}$
3) Pt = Open(Registration Table)
4) while(Pt = NULL)
5) Rec = Read(Pt)
6) if(Rec(User_Id) == UI)
7) status = 1
8) Break
9) return status

Whereas, *UI*, *status*, *A*, *Pt*, and *Rec* are *User_Id*, status of registered user, character set matrix, pointer for password file, and structure variable having user record respectively

b) Password entry in the form of numeric code

This sub-module aims to generate a numeric code containing row and column indices of each character of the entered password during login time. This module needs two inputs: 1) a character-set matrix, and 2) the user id.

After receiving the character-set-matrix, the column-wise permutation is performed, and a permutated version of this matrix is created. The permutated matrix is then labeled with column numbers, which are shown below.

$$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \end{matrix}$$
$$\begin{bmatrix} F & C & E & A & B & D \\ L & I & K & G & H & J \\ R & O & Q & M & N & p \\ X & U & W & S & T & V \\ 3 & 0 & 2 & Y & Z & 1 \\ 9 & 6 & 8 & 4 & 5 & 7 \\ ! & \$ & \_ & @ & \# & \& \end{bmatrix}$$

Next, the user is asked to enter the column numbers containing the password characters in the above-permutated matrix. The entered column numbers are then stored in a queue. Next, the transpose of those columns of the permutated

TABLE 1. Password database on server.

User_Id	Numeric Password	Ideal Personality Name
11	23 22	23315 23441
12	163 122	6314 3311
13	4516 1113	6315 3341
14	21631 21324	3116 4313
15	312312 234215	412 213
16	1312215 3442413	112 313
17	651312612332 312244314233	51256 41413
18	3245612124321 4311344131232	51256 41413

matrix whose column numbers are present in the queue is taken using Eq. (1). The transposed columns' resultant values are then stored in the partial-transpose-matrix in sequential order, i.e., the column of the permutated matrix, which is transposed first, is placed on the first row of the partial-transpose-matrix and so on.

Now, this module has partial information about the location of the password characters, i.e., the first character of the password is located in the first row of the password-matrix, 2^{nd} character is located in the 2^{nd} row, and the last character is located in the last row of the password- matrix. However, it has no information regarding column indices, i.e., in which column of the first row of the partial-transpose-matrix, the password's first character is present.

To get this information, this module labels the password-matrix with column numbers and asks for information about the column numbers of the password- matrix containing password characters. This module stores the entered column numbers of the password- matrix in the second queue. This module has complete information about each character of the password, i.e., the row index of each character of the password is stored in the first queue, and its column index is stored in the second queue. Thus, it combines both the first and second queues into the third queue for generating the numeric code for the desired password. This module's working procedure is implemented in Algorithms 3, and 4.

Whereas, A, UI, status, Columns, C, Q_1, B, D, Q_2, Q and OK are character set matrix, User_Id, the status of a registered user, number of columns in character set matrix, permutated character set matrix labeled with columns numbers, a queue of column labels containing password characters of the first interface, transpose of character set matrix permutated transpose matrix having user password, queue having columns of password characters of the second interface, queue having a resultant string of rows + columns, and password status, respectively.

Whereas, A, UI, status, Columns, C, Q_1, B, D, Q_2, Q and OK are character set matrix, User_Id, the status of a registered user, number of columns in character set matrix, permutated character set matrix labeled with columns numbers, a queue of column labels containing password characters of the first interface, transpose of character set matrix permutated transpose matrix having user password, queue having columns of password characters of the second interface, queue having a resultant string of rows + columns, and password status, respectively.

Algorithm 3 Login_Module (A, UI)

```
status = User_Status(UI)
if status == 0
    Display (''invalid user-id'')
else
    Columns = Total Number of Columns of A
    C = Permutation(A,Columns)
    C = Label matrix C with columns numbers
    Q1 = Enter columns of C containing the password characters:
    B = Transpose (A)
    D = TransposeMatrix (C, Q1)
    Q2 = Enter columns of D containing password characters:
    Q = Concatinate(Q1, Q2)
    OK = PWVerification(D, Q, B, UI)
    if (OK == 1)
        Display(''Login Permitted'')
    else
        Display(''Invalid User Password'')
```

Algorithm 4 TransposeMatrix (A, SelectedColumns)

```
row = 1
Q3 = SelectedColumns
m = Total Number of Columns of C
n = Total Number of Rows of C
D = zeroes(n, m)
While(Q3 ≠ φ)
    col = DEQUEUE(Q3)
    D(row, n) = Transpose(C(n, col))
    ENQUEUE(Q1,row)
    row = row +1
D = Permutation(D, n)
D = Label matrix D with columns numbers
return (D)
```

For example, suppose a user's password is "CAMAL". In that case, he/she will need to enter the respective column numbers containing individual characters of his/her password from the labeled version of the permutated matrix, as shown above. Since the individual characters of the user password (*CAMEL*), i.e., C, A, M, A, and L are present in columns 2, 4, 4, 4, and 1, respectively of the permutated matrix. So, the user will need to enter these columns like 24441. After this, a password-matrix and a queue for holding transpose of the entered columns (24441) of the permutated matrix and the row numbers of password-matrix, respectively, are created and shown as follows.

$$B = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\ \begin{bmatrix} C & 1 & O & U & O & 6 & \$ \\ A & G & M & S & Y & 4 & @ \\ A & G & M & S & Y & 4 & @ \\ A & G & M & S & Y & 4 & @ \\ F & L & R & X & 3 & 9 & ! \end{bmatrix} \end{array} \text{ and } Q_1 = [12345]$$

This module has information about the row index of each character of the password in the form of Q_1, i.e., the first character of the password is present in the first row of the password-matrix named as B, and so on. For obtaining information regarding each password character's column index, this module labels the password-matrix columns with column numbers and asks the user to enter column numbers of labeled password-matrix containing password characters. So, for the above-stated password, the user will need to enter column numbers like 11312, and these column numbers will be stored in queue Q_2. Next, the contents of Q_1 and Q_2 are combined into Q, so $Q = 1234511312$. The user will not need to enter the individual characters of the password "CAMAL" Instead, he/she will need to enter the numeric code stored in Q.

3) PASSWORD VERIFICATION

This module aims to verify the authenticity of the user password's numeric code during the login process. The user password's numeric code for the given user id is loaded from the registered users' database. Next, it decomposes the numeric code into two parts: (i) the left half part containing row indices, and (ii) the right half part containing column indices of the desired password. The row index and column index of the respective characters of the password are then extracted. Using these indices, all the characters of the password of the given user id are restored. The same procedure is applied for restoring the textual password from the numeric code entered during the login process. These two passwords are then compared. If both match, then the user is login successfully; otherwise, an invalid password message is displayed. Algorithms 5, 6, and 7 implement the working of this module.

Where DBPass, EntPass, and OK represent to database password, the password entered, and the status of the user password, respectively.

Where, L, *mid*, Q_r, Q_c, and *EntPass*, denote to the length of numeric code stored in a queue, middle of the length, queue containing row indices of password characters, the queue is containing column indices of password characters, and the login entry password returned by this module, respectively.

Where (Algorithm 7), Rec, n, L, *mid*, and *DBPass* refer to the structure variable containing a record of the password file, the content of password attribute of the current record read by the file, length of password attribute, middle of password length, and database password returned by this module to verification module respectively.

The logup process gets a user id and a matrix containing password characters. The first queue stores the row indices of the password characters from the user inputs, i.e., the entered column numbers containing password characters in the received matrix. While the second queue stores the

Algorithm 5 PW Verification (D, Q, B, UI)

```
1) DBPass = DBPassword (B, UI)
2) EntPass = EntryPassword (D, Q)
3) if (DBPass == EntPass)
4)     OK = 1
5) else
6)     OK = 0
7) return (OK)
```

Algorithm 6 EntryPassword (D, Q)

```
1) L = length(Q)
2) mid = L/2
3) Qr = Q(1 : mid)
4) Qc = Q(mid + 1 : L)
5) EntPass = ''
6) while(Qr ≠ ϕ)
7)     row = DEQUEUE (Qr)
8)     col = DEQUEUE(Qc)
9)     EntPass = Concatinate (EntPass,D(row, col))
10) return (EntPass);
```

Algorithm 7 DBPassword(B, UI)

```
1) Pt = Open(RegistrationTable)
2) DBPass = ''
3) While(Pt ~ = NULL)
4)     Rec = Read(Pt)
5)     if(Rec(User_Id) == U I)
6)         n = Rec(pword)
7)         L = length(n)
8)         mid = L/2
9)         Qr = n(1 : mid)
10)        Qc = n(mid + 1 : L)
11)        While(Qr ≠ ϕ)
12)            row = DEQUEUE(Qr)
13)            col = DEQUEUE(Qc)
14)            DBPass = Concatenate (DBPass,
    B(row, col))
15)        Break
16) return (DBPass)
```

column indices of the password characters that come from the user inputs but this time, by entering the column numbers of the transposed matrix containing password characters derived from the received matrix. The values in these two queues are then combined, and a numeric code equivalent to the user password is formed. The computed numeric code and user id are then stored in the password database.

Like the logup process, two queues are created in the login process to store row and column indices of the password characters. During the login process, this scheme gets a user id from the user, checks its validity; if it is valid, then the column-wise permutation of matrix containing password characters is taken, and the user is asked to type his/her password by entering the column numbers of the permutated matrix containing his/her password characters. Next, the transpose of the entered columns of the permutated matrix is taken, and the resultant values are stored in the transpose matrix. The user is then asked to enter the column numbers of the transposed matrix containing his/her password characters. The first and second time entered column numbers represent the row and column indices of the password characters in the transposed matrix. These row and column indices of the password characters are then combined, and a numeric code equivalent to the user password is formed. This numeric code and the numeric code for the given user id present in the password database are then decoded for restoring the textual passwords. If both of the restored textual passwords match, then login is succeeded. Otherwise, an invalid password message is displayed.

4) UPDATE PASSWORD

Sometimes, the end-user needs to change his/her password due to security reasons. This module's objective is to facilitate the end-user to update his/her old password with the new one. This module verifies the end-users old password first using the verification process mentioned in the verification module. If it is found correct, the user can enter his/her new password, and this module replaces it with the old password in the database. This password updating process is shown in Fig. 8 and is implemented using Algorithm 8.

5) FORGOT PASSWORD

This module aims to allow the end-user to register himself with a new password if he forgets it. This module needs the perfect personality name of the end-user as an alternate password to achieve this aim. After getting the name of the personality in the form of numeric code, it then converts that numeric code into the textual form. It compares it with the textual form obtained from its appropriate numeric code present in the password database. If both match, then the end-user can enter and register his/her new password in the form of a numeric code, and the process is terminated. On the other hand, if both are found mismatched, they are asked to enter his/her correct personality name. The above-stated process is diagrammatically shown in Fig. 9 and is implemented via Algorithm 9.

Algorithm 8 Password_Update_Module(A,UI)

```
status = User_Status(UI)
if status == 0
    Display (''invalid user id'')
else
    Columns = Total Number of Columns of A
    C = Permutation(A,Columns)
    C = Label matrix C with columns numbers
    Q1 = Enter columns of C containing the password characters :
    B = Transpose (A)
    D = TransposeMatrix (C, Q1)
    Q2 = Enter columns of D containing password characters:
    Q = Concatinate(Q1, Q2)
    OK = PWVerification(D, Q, B, UI)
    if (OK == 1)
        A1 = Label Matrix A with columns numbers
        Q1 = Enter columns of A1 containing New password characters:
        B = TransposeMatrix (A, Q1)
        B = Label Matrix B with columns numbers
        Q2 = Enter columns of B containing New password characters:
        QP = Concatinate(Q1, Q2)
        Pt = Open(RegistrationTable)
        while(Pt~ = NULL)
            Rec = Read(Pt)
            if(Rec(User_Id)== UI)
                replace(Pt, UI, QP, Rec(QN))
                Close(Pt)
                Break
    else
        Display(''Invalid Old Password'')
```

B. PROPOSED BRUTE FORCE ALGORITHM

For evaluating the security of our proposed scheme, we developed a brute force Algorithm which makes all possible numeric combinations of user password (password numeric code) using Eq. (2).

$$Z = X * Y \tag{2}$$

where X refers to the set of all combinations made from the first part of the numeric password and Y denotes the set of all combinations made from the second part of the numeric password. These two sets X and Y are described as follows.

$$X = \{X_1, X_2, X_3, \cdots, X_n\}$$
$$Y = \{Y_1, Y_2, Y_3, \cdots, Y_m\}$$

where $\{X_1, X_2, X_3, \cdots, X_n\}$ represent the n different combinations of the first part of numeric code entered in the first user interface, and $\{Y_1, Y_2, Y_3, \cdots, Y_m\}$ denote the m combinations of the second part of numeric code entered in the second user interface. The value of n, m can be computed

Algorithm 9 Password_Forgotten_Module(A,UI)

```
status = User_Status(UI)
if status == 0
    Display ("invalid user-id")
else
    Columns = Total Number of Columns of A
    C = Permutation(A,Columns)
    C = Label matrix C with columns numbers
    Q1 = Enter columns of C containing theIdeal Personality Name characters:
    B = Transpose (A)
    D = TransposeMatrix(C, Q1)
    Q2 = Enter columns of D containing Ideal Personality Name characters:
    Q = Concatinate (Q1, Q2)
    OK=PWVerification(D, Q, B, UI)
    if (OK == 1)
        A1 = Label Matrix A with columns numbers
        Q1 = Enter columns of A1 containing New password characters:
        B = TransposeMatrix (A, Q1)
        B = Label Matrix B with columns numbers
        Q2 = Enter columns of B containing New password characters:
        QP = Concatinate(Q1, Q2)
        Pt = Open(RegistrationTable)
        while(Pt~ = NULL)
            Rec = Read(Pt)
            if(Rec(User_Id) == UI)
                replace(Pt, UI, QP, Rec(QN))
                Close(Pt)
                Break
```

using Eq. (3) and Eq. (4), respectively.

$$n = (F)^T \tag{3}$$

$$m = (S)^T \tag{4}$$

F and S refer to the number of unique characters(indices) in the first and second parts of numeric code, while T represents the numeric password length. The values of n and m may or may not be equal. This is because the number of unique characters (indices), i.e., F and S values in two parts of the numeric code, may or may not be equal. The total number of combinations is then computed by multiplying n with m.

For instance, in the case of numeric password 163122, the value of F, S, and T are 3, 2, and 3, respectively. Putting these values in Eq. (3), and Eq. (4), we get $n = 27$, and $m = 8$. The total number of combination forms by brute force is $27 * 8 = 216$.

The values of X and Y are then put in Eq. (2), and all combinations of the numeric password are then formed by applying the Cartesian product given as follows.

$$Z = \{(X_1, Y_1), (X_1, Y_2), (X_1, Y_3), \cdots, (X_1, Y_8), (X_2, Y_1), (X_2, Y_2), (X_2, Y_3), \cdots, (X_2, Y_8), (X_27, Y_1), (X_27, Y_2), (X_27, Y_3), \cdots, (X_27, Y_8)\}$$

where, each element in parenthesis $(X_1, Y_1), (X_1, Y_2), (X_1, Y_3), \cdots, (X_1, Y_8)$ represents one possible numeric password equivalent to a user password.

IV. EXPERIMENTAL RESULTS

This section presents the evaluation metrics, usability analysis, security analysis, and performance comparison of our proposed password scheme with state-of-the-art password methods.

A. DATA SET

We created 150 user ids and assigned different length passwords to them. The number of users having different length password is shown in Table 2.

TABLE 2. Data set of users.

Number_of_users	Password length
20	2
16	3
16	4
16	5
16	6
16	7
10	8
10	9
10	10
10	11
05	12
05	13

B. EVALUATION METRICS

Two metrics, i.e., overall usability and the average probability of success/synchronization, are used for evaluating the usability and security of our proposed password scheme. The usability of the password scheme is affected by contextual factors like human (H), design (D), and technology (T) [18]. The value of H and D are computed using Eq. (5) and Eq. (6), respectively.

$$H = K + A + Cz \tag{5}$$

$$D = ND + SC \tag{6}$$

where K, A, and Cz refer to keystroke efforts, manual arithmetic involved in password entry, and the character set's size, respectively. While SC and ND denote the number of screens displayed during the login process and simple/complex interface, respectively. Moreover, T represents the use of different devices. Our proposed password scheme's overall usability and those present in the literature are computed using Eq. (7). This overall usability measure is used for comparison purposes.

$$OU = H + D + T \tag{7}$$

TABLE 3. Comparison of usability of knowledge-based password schemes.

Schemes/Approaches	Human Factors			Design Factors		Technology	Overall usability
	K	A	Cz	ND	SCI	T	OU
Mobile Authentication System [5]	3	1	1	1	1	1	8
Shoulder Surfing Proof [7]	0	0	2	0	1	1	4
Text-Based Authentication [10]	1	0	2	1	1	0	5
Dynamic Password Protocol [37]	0	1	2	0	1	0	4
Secure Authentication Mechanism [39]	0	1	2	1	1	0	5
Our Proposed Scheme	**1**	**0**	**1**	**1**	**0**	**0**	**3**

The probability of success of the brute force algorithm on cracking user passwords is computed using Eq. (8).

$$P^{(L)} = \frac{\frac{1}{n \times m}}{(R \times C)(L \times R)} \tag{8}$$

where $n \times m$ refers to the total number of candidate combinations formed by the brute force algorithm for a given database password. L refers to the password's length, and its value should be in the range of $1, \cdots, 12$.

The TC denotes the total number of candidate combinations formed by the brute force algorithm for a password of length l. While $(R \times C) \times (L \times R)$ refers to the number of positions of characters changed in testing each candidate numeric password.

The average probability of success of the proposed brute force algorithm is then computed and is used for comparison purposes.

C. USABILITY ANALYSIS

We evaluated our proposed scheme's usability based on contextual factors, i.e., human, design, and technology described in the evaluation criteria. The criteria used for assigning weight to each sub-factor of these three contextual factors are described as follows:

Key stroke efforts (K): If a password scheme needs one keystroke per password character, which is the minimum number, then weight 0 is assigned to this sub-factor. Similarly, for a password scheme that requires two keystrokes per password character, a weight of 2 is assigned to this sub-factor. The same rule is applied to other password schemes that need 3 or more than 3 keystrokes per character.

Arithmetic in password entry (A): The weight of this sub-factor is one if arithmetic is involved during password entry of a password scheme; otherwise, its weight is 0. This factor involves human mental engagement; therefore, the highest weight is assigned to the password scheme, which needs arithmetic during password entry.

size of the character set (Cz): The printable characters used in password entry are near about 94 comprising of four groups (upper case letters, lower case letters, numerals, and special symbols) if the size of the character set of a password scheme consists of one group or 25% of total printable characters then weight 0 is assigned to this sub-factor. If two groups or 50% of printable characters are used in the password scheme, weight 1 is assigned to this sub-factor. Moreover, if all groups or 100% of printable characters are used in the password scheme, weight 2 is assigned to this sub-factor.

Entry on single/multiple screens (ND): If the whole password is entered using a single screen, then weight 0 is assigned to this sub-factor, and in the case of multiple screens, weight 1 is assigned.

Simple/complex interface (SCI): If the interface of a password scheme is user-friendly and straightforward, i.e., the user can find and enter his password characters quickly, then weight 0 is assigned to this sub-factor, and if a complex interface, the weight 1 is assigned.

Extra device/ technology (T): If the scheme requires an extra device for its password entry, then weight 1; otherwise, weight 0 is assigned to this contextual factor.

Using the weights of these sub-factors, our proposed password scheme's overall usability and that of available password methods are computed using Eq. (4), shown in Table 3. The password scheme having overall usability in the range of $2-3$ is considered as easy to use, while the one having overall usability in the range of $6-8$ is considered as difficult to use. The password scheme, with overall usability in the range of $4-5$, is considered moderate. As shown in Table 3, our proposed scheme is easy to use according to the specified criteria.

D. SECURITY ANALYSIS

In this sub-section, the server-side security and client-side security of our proposed scheme are evaluated.

1) SERVER SIDE SECURITY ANALYSIS

The server-side security of our proposed password scheme was evaluated using a brute force attack.

Brute force attack: We randomly picked a numeric password of length 2 from the password database and gave it as input to our proposed brute force algorithm for making all its candidate password combinations. The proposed brute force algorithm tested each candidate's password combinations on our proposed scheme. The same procedure was applied on numeric passwords of length 3 and 4 characters, respectively. The value of the probability of success for each length

TABLE 4. Evaluation results of brute force attack on server side password database.

User_Id	Password length	$c1 = n * m$	$c2$ =No of permutation in FUI	$c3$ =No of permutation in SUI	$p(i) = \frac{1}{c1 \times c2 \times c3}$
11	2	4	42	14	0.000425170068027211
12	3	216	42	21	0.00005249013185521 1200
13	4	4096	42	28	0.00000207602572278 9120
14	5	1048576	42	35	0.00000000648758038 3716
15	6	11390625	42	42	0.00000000049768421 3146
16	7	268435456	42	49	0.00000000001810150 7767
20	8	656100000000	42	56	0.00000000000000064 80263
23	9	101559956668416	42	63	0.00000000000000003 7212
25	10	590490000000000	42	70	0.00000000000000000 5760
30	11	131621703842267000	42	77	0.00000000000000000 0023
40	12	4738381338321620000	42	84	0.00000000000000000 0001
					0.000039147944

TABLE 5. Evaluation results of shoulder surfing attack on our proposed scheme.

User Id	Shoulder Surfer Password		Proposed Scheme Password	
	Numeric	Reformed	Numeric	Reformed
11	54124 32536	5LU73	21231 54341	HARIS
12	531 711	$\_HL$	721 436	ALI
13	6214 5554	$\_!@N$	2136 5721	DEAR
14	6531 7726	$\_@XO$	7681 1267	BA3
15	556 774	XXA	865 356	OO7

password was 0. This is because the brute force is not providing the correct candidate numeric password at the right time to our proposed password scheme.

However, each time the brute force algorithm does not need to fail because a situation in which the required numeric password and the brute force algorithm candidate password match may occur. In this situation, the login will succeed. Thus, the brute force's success depends on synchronizing the candidate numeric password and required numeric password. The values of the probability of synchronization computed using Eq. (8). are shown in Table 4.

2) CLIENT SIDE SECURITY ANALYSIS

The shoulder surfing and brute force attacks were used to evaluate our proposed password scheme's client-side security.

a) Shoulder Surfing attack: We grabbed the user ids and numeric passwords of five users during their login sessions. The grabbed user ids and numeric passwords were tested one by one on our proposed password scheme, and none of them were found correct. The reason is that the proposed scheme does not change the password characters but changes their positions randomly after each login. Thus, the grabbed numeric code is not effective at the next login because the proposed scheme changes the numeric password after each login and leaves no clue for the shoulder surfer to guess the new numeric password.

They grabbed user ids, grabbed numeric passwords, their respective reformed passwords, the correct numeric passwords needed by our proposed scheme, and their respective reformed passwords are shown in Table 5.

As it is clear from Table 5, each of the textual passwords formed from the grabbed numeric passwords does not match their respective textual password formed from the correct numeric password. The average probability of success of the shoulder surfing attack on cracking passwords of lengths $2, 3, \cdots, 7$ characters is 0.

b) Brute force attack: For checking our proposed scheme's client-side security, the grabbed user ids and numeric passwords were given to our proposed brute force algorithm one by one. The proposed brute force algorithm formed all possible combinations of each numeric password and gave them to our proposed scheme one by one. It was observed that none of the combinations was succeeded in the login process. All the formed combinations contain a combination that matches the correct numeric password. However, password entry timing causes a mismatch between the user's numeric password combination and a correct candidate numeric password in the generated combinations of our proposed brute force algorithm. One instance of the grabbed numeric password, a subset of the numeric password

TABLE 6. Evaluation results of brute force attack on client-side.

Hacked NP	BF generated combinations	Required Corresponding NPs
531 711	531 711	513 711
	315 117	
	513 711	
	.	.
	.	.
	.	.
	.	.
	153 711	531 711

TABLE 7. Comparison of the Security.

Methods	Client-side Security		Server-side security
	Avg. prob. of SS attack	Avg. prob. of BF attack	Avg. prob.of BF attack
Mobile Authentication System [5]	1	0.1216216	Nil
Text-Based Authentication [10]	0	0.0000968	0.0000968
Dynamic Password Protocol [37]	1	0.102663	0.0000852
Secure Authentication Mechanism [39]	0	0.1550	Nil
Our Proposed Scheme	0	0.0000718	0.0000718

combinations formed by our proposed brute force algorithm, and the correct numeric password are shown in Table 6.

It is clear from Table 6 that the generated combinations of brute force algorithms contain the same combination (i.e., 513711) as the required one (i.e., 513711). However, its turn will reach at number 3, and our proposed scheme needs it at number 1. Similarly, the proposed brute force algorithm assigns a combination of 531 711 at several 1, and our proposed scheme needs it at the last number. Although in this proper scenario, the probability of synchronization between the candidate numeric password and the required numeric password is 0. However, still, the chances of success of brute exist because a situation of synchronization may occur. So, just like a brute force attack in subsection "Server-side security analysis", the probability of synchronization can be computed using Eq. (8).

3) SECURITY COMPARISON

The probabilities of success of the shoulder surfing attack on the password of lengths 2, 3, $\cdots$, 12 were manually computed for each of the available schemes [5], [10], [37], [39] and that of our proposed password scheme. The values of these probabilities were then used to compute the average probabilities of success of the shoulder surfing attack for [5], [10], [37], [39], and that of our proposed scheme, shown in Table 7.

In [5], many interfaces containing 2, 3, 4, 5 or 6 layers are used to minimize the success rate of the brute force attack. We have selected the 3-layers interface with 37 characters for comparison purposes because our proposed interface contains 42 characters. Therefore, the 3-layers interface and its brute force success probability are more suitable for comparing the brute force success probability with our proposed scheme. The probability of success of the brute force algorithm against password scheme [5] using 3-layers interface is $1/37$ for one password character; so, for comparison purposes, we multiplied the probability of guessing one character with password length to calculate the probability of the whole password.

To compare the probability of success of the brute force attack against our proposed scheme with password scheme [10], we need to find the probability of all length combinations (entire password), single length combinations (single character of password at a time) or both. The candidate combinations formed by the brute force algorithm of [10] are more extensive than our proposed scheme. However, our proposed scheme uses additional features, i.e., a permutation of characters during each login process, which contributes more towards minimizing the brute force algorithm's success rate.

The same methods were applied for computing the probability of success of the brute force attack on the server-side.

The average values of the probability of shoulder surfing attack, brute force attack on the client-side, and brute force

TABLE 8. Time required to Brute Force Algorithm for Testing all the Combinations of Multiple length Passwords.

Password Length	Number of possible Combinations	Secs	Minutes	Hours	Days	Years
3	74088	27.61	0.46	0.01	0.00	8.75507E-07
4	3111696	1387.67	23.13	0.39	0.02	4.40027E-05
5	130691232	58282.14	971.37	16.19	0.67	0.001848115
6	5489031744	2447849.9	40797.50	679.96	28.33	0.077620811
7	230539333248	102809695	1713494.92	28558.25	1189.93	3.260074041

attack on the server-side for the schemes [5], [10], [37], [39] and that of our proposed scheme are shown in Table 7.

Our proposed method needs a different numeric password at each login because the characters' positions change at each login process. Due to this vital feature of our proposed password scheme, its resistance against shoulder surfer attack is far better than [5], [10], [37], [39]. Additionally, the brute force attack's probability of success against our proposed password scheme is comparatively very low than [5], [10], [37], [39] because the brute force algorithm does not know which candidate among the available $n * m$ candidates is the need of our proposed password scheme at a specific time.

Like [10], [37], our proposed scheme does not store the original textual password, but its alternative numeric password is stored on the server-side. Thus, our proposed scheme's server-side security is comparable to [10], [37] and better than [5].

In summary, our proposed scheme provides better usability and security as compared to [5]. The security of our proposed scheme is comparable to [10], [37]. However, its usability is superior, then [10], [37].

E. TIME ANALYSIS OF BRUTE FORCE ALGORITHM

The total time required for cracking passwords generated by our proposed scheme is computed using Eq. (9)

$$T_{cr} = T_c + T_t \tag{9}$$

T_c refers to the time required for making all combinations by our proposed brute force algorithm, and T_t denotes the time required for testing all combinations. The time needed for cracking passwords of length 3, 4 and 5 characters were recorded practically, while other passwords having length 6 and 7 characters were computed relatively. The required time for cracking each length password is shown in Table 8. The time in days and years shows that the brute force will face a high challenge for breaking passwords having a length greater than or equal to 7 characters. This characteristic shows the strength of our proposed password scheme against our proposed brute algorithm.

V. CONCLUSION AND FUTURE WORK

We found some limitations of the available reformation-based password schemes, i.e., they improved the client-side security of the electronic applications by using an extra device, involving mental computation, using an extensive character set, or utilizing several user interfaces. In other words, they improved the client-side security of the electronic applications but complicated their usability. Furthermore, it is also found that most of the available reformation-based password schemes do not provide server-side security because they store the actual content of the password on the server database. In this article, we have proposed a new reformation based password scheme that improved the client and server sides' security of the electronic applications without complicating their usability. The proposed scheme uses two different numeric codes, one on the client and the other on the server-side, for the same actual textual password. Algorithm 6 deals with transforming the client-side numeric code into the actual textual password, while Algorithm 7 maps the server-side numeric code into an actual textual password. The obtained textual passwords are then compared by Algorithm 5 for password verification. Our proposed password scheme's usability and security (both client and server sides) were evaluated and compared with state-of-art reformation-based password schemes using overall usability and the average probability of success metrics. The experimental results of our proposed password scheme showed better usability (as shown in Table 3), client and server-side security (as shown in Table 7) in comparison to the state of the art reformation based password schemes. In the future, we intend to improve the usability by providing a single user interface for login entry and entering the same number of characters as contained in the password during the login process. Additionally, we also intend to improve security against a strong shoulder surfing attack.

REFERENCES

[1] F. Z. Glory, A. Ul Aftab, O. Tremblay-Savard, and N. Mohammed, "Strong password generation based on user inputs," in *Proc. IEEE 10th Annu. Inf. Technol., Electron. Mobile Commun. Conf. (IEMCON)*, Oct. 2019, pp. 0416–0423.

[2] S. Liang, Y. Zhang, B. Li, X. Guo, C. Jia, and Z. Liu, "Secureweb: Protecting sensitive information through the Web browser extension with a security token," *Tsinghua Sci. Technol.*, vol. 23, no. 5, pp. 526–538, Oct. 2018.

[3] J. Song, D. Wang, Z. Yun, and X. Han, "Alphapwd: A password generation strategy based on mnemonic shape," *IEEE Access*, vol. 7, pp. 119052–119059, 2019.

[4] S. W. Shah and S. S. Kanhere, "Recent trends in user authentication—A survey," *IEEE Access*, vol. 7, pp. 112505–112519, 2019.

[5] J.-N. Luo and M.-H. Yang, "A mobile authentication system resists to shoulder-surfing attacks," *Multimedia Tools Appl.*, vol. 75, no. 22, pp. 14075–14087, Nov. 2016.

[6] P. Shi, B. Zhu, and A. Youssef, "A PIN entry scheme resistant to recording-based shoulder-surfing," in *Proc. 3rd Int. Conf. Emerg. Secur. Inf., Syst. Technol.*, 2009, pp. 237–241.

[7] T.-S. Wu, M.-L. Lee, H.-Y. Lin, and C.-Y. Wang, "Shoulder-surfing-proof graphical password authentication scheme," *Int. J. Inf. Secur.*, vol. 13, no. 3, pp. 245–254, Jun. 2014.
[8] H. Zhao and X. Li, "S3PAS: A scalable shoulder-surfing resistant textual-graphical password authentication scheme," in *Proc. 21st Int. Conf. Adv. Inf. Netw. Appl. Workshops (AINAW)*, 2007, pp. 467–472.
[9] Z. Zheng, X. Liu, L. Yin, and Z. Liu, "A stroke-based textual password authentication scheme," in *Proc. 1st Int. Workshop Edu. Technol. Comput. Sci.*, vol. 3, 2009, pp. 90–95.
[10] S. Zaman, S. Raheel, T. Jamil, and M. Zalisham, "A text based authentication scheme for improving security of textual passwords," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 7, pp. 513–521, 2017.
[11] M. H. Zaki, A. Husain, M. S. Umar, and M. H. Khan, "Secure pattern-key based password authentication scheme," in *Proc. Int. Conf. Multimedia, Signal Process. Commun. Technol. (IMPACT)*, Nov. 2017, pp. 171–174.
[12] S. Safdar, M. Fadzil Hassan, M. Aasim Qureshi, R. Akbar, and R. Aamir, "Authentication model based on reformation mapping method," in *Proc. Int. Conf. Inf. Emerg. Technol.*, Jun. 2010, pp. 1–6.
[13] M. Shakir and A. A. Khan, "S3TFPAS: Scalable shoulder surfing resistant textual-formula base password authentication system," in *Proc. 3rd Int. Conf. Comput. Sci. Inf. Technol.*, Jul. 2010, pp. 12–14.
[14] C. Herley and P. Van Oorschot, "A research agenda acknowledging the persistence of passwords," *IEEE Secur. Privacy Mag.*, vol. 10, no. 1, pp. 28–36, Jan. 2012.
[15] R. A. Abdeen, "Start-to-end algorithm for string searching," *Int. J. Comput. Sci. Netw. Secur.*, vol. 11, no. 2, no. 2011, pp. 179–182.
[16] R. A. Abdeen, "An algorithm for string searching based on brute-force algorithm," *Int. J. Comput. Sci. Netw. Secur.*, vol. 11, no. 7, pp. 24–27, 2011.
[17] G. Sowmya, D. Jamuna, and M. Reddy, "Blocking of brute force attack," *Int. J. Eng. Res. Technol.*, vol. 1, no. 6, pp. 1–4, 2012.
[18] C. Katsini, M. Belk, C. Fidas, N. Avouris, and G. Samaras, "Security and usability in knowledge-based user authentication: A review," in *Proc. 20th Pan-Hellenic Conf. Informat.*, Nov. 2016, pp. 1–6.
[19] M. Raza, M. Iqbal, M. Sharif, and W. Haider, "A survey of password attacks and comparative analysis on methods for secure authentication," *World Appl. Sci. J.*, vol. 19, no. 4, pp. 439–444, 2012.
[20] M. A. S. Gokhale and V. S. Waghmare, "The shoulder surfing resistant graphical password authentication technique," *Procedia Comput. Sci.*, vol. 79, pp. 490–498, Dec. 2016.
[21] N. Chakraborty, G. S. Randhawa, K. Das, and S. Mondal, "MobSecure: A shoulder surfing safe login approach implemented on mobile device," *Procedia Comput. Sci.*, vol. 93, pp. 854–861, Dec. 2016.
[22] J. D. Still and J. Bell, "Incognito: Shoulder-surfing resistant selection method," *J. Inf. Secur. Appl.*, vol. 40, pp. 1–8, Jun. 2018.
[23] R. English and R. Poet, "Towards a metric for recognition-based graphical password security," in *Proc. 5th Int. Conf. Netw. Syst. Secur.*, 2011, pp. 239–243.
[24] R. Afandi and R. Bin, "ChoCD: Usable and secure graphical password authentication scheme," Ph.D. dissertation, Dept. Inf. Secur. Assurance, Univ. Sains Islam Malaysia, Nilai, Malaysia, 2016.
[25] G. E. Blonder, "Graphical password," U.S. Patent 5 559 961, Sep. 24, 1996.
[26] D. Sathyamoorthy, M. J. M. Jelas, and S. Shafii, "Wireless spy devices: A review of technologies and detection methods," *Editorial Board*, vol. 7, p. 130, May 2014.
[27] *Disabled People in the World in 2019: Facts and Figures*. Accessed: Nov. 19, 2020. [Online]. Available: https://www.inclusivecitymaker.com/disabled-people-in-the-world-in-2019-facts-and-figures/
[28] K. Sigmon, "How to kill copyright: A brute-force approach to content creation," *Wake Forest J. Bus. Intell. Prop.*, vol. 14, p. 26, Dec. 2013.
[29] S. Vaithyasubramanian, A. Christy, and D. Saravanan, "An analysis of Markov password against brute force attack for effective Web applications," *Appl. Math. Sci.*, vol. 8, pp. 5823–5830, 2014.
[30] Q. Yan, J. Han, Y. Li, and R. H. Deng, "On limitations of designing leakage-resilient password systems: Attacks, principles and usability," in *Proc. 19th Netw. Distrib. Syst. Secur. Symp. (NDSS)*. San Diego, CA, USA: Research Collection School of Information System, 2012.
[31] J. Owens and J. Matthews, "A study of passwords and methods used in brute-force SSH attacks," in *Proc. USENIX Workshop Large-Scale Exploits Emergent Threats (LEET)*. 2008, pp. 1–5.
[32] K. T. Dave, "Brute-force attack seeking but distressing," *Int. J. Innov. Eng. Technol. Brute-force*, vol. 2, no. 3, pp. 75–78, 2013.
[33] C. Adams, G.-V. Jourdan, J.-P. Levac, and F. Prevost, "Lightweight protection against brute force login attacks on Web applications," in *Proc. 8th Int. Conf. Privacy, Secur. Trust*, Aug. 2010, pp. 181–188.
[34] O. Masanori, T. Ryo, and S. Tadamasa, "An evaluation of string search algorithms at user standing," in *Proc. 3rd WSES Int. Conf. Math. Comput. Mech. Eng. (MCME)*, 2001, pp. 4231–4236.
[35] E. Erdem and M. T. SandÄkkaya, "OTPaaS—One time password as a service," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 3, pp. 743–756, Aug. 2018.
[36] W. Luo, Y. Hu, H. Jiang, and J. Wang, "Authentication by encrypted negative password," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 1, pp. 114–128, Jan. 2019.
[37] H. Channabasava and S. Kanthimathi, "Dynamic Password Protocol for User Authentication," in *Proc. Intell. Comput.-Proc. Comput. Conf.* Cham, Switzerland: Springer, 2019, pp. 597–611.
[38] M. Al-Zubaidie, Z. Zhang, and J. Zhang, "RAMHU: A new robust lightweight scheme for mutual users authentication in healthcare applications," *Secur. Commun. Netw.*, vol. 2019, pp. 1–26, Mar. 2019.
[39] S. Subangan and V. Senthooran, "Secure authentication mechanism for resistance to password attacks," in *Proc. 19th Int. Conf. Adv. ICT Emerg. Regions (ICTer)*, vol. 250, 2019, pp. 1–7.

MUSHTAQ ALI received the M.Sc. degree in computer science from Gomal University, Pakistan, in 2002, the M.S. degree in computer science from COMSATS University, Pakistan, in 2007, and the Ph.D. degree in computer science and technology from the University of Electronic Science and Technology of China in 2015. He is currently an Assistant Professor with the Department of Information Technology, Hazara University Mansehra. He is a Ph.D. Supervisor recognized and approved by the Higher Education Commission, Pakistan. His research areas include digital image processing, content-based image retrieval systems, computer vision, and steganography.

AMANULLAH BALOCH received the M.Sc. degree in computer science from Gomal University, Pakistan, in 1991, and the M.S. degree in computer science from Hazara University, Pakistan, in 2018, where he is currently pursuing the Ph.D. degree. He is currently an Assistant Professor with the Department of Information Technology, Hazara University Mansehra. His research areas include digital image processing, content-based image retrieval systems, and information security.

ABDUL WAHEED received the master's degree in computer sciences from the Department of Information Technology, Hazara University Mansehra, in 2014, where he is currently pursuing the Ph.D. degree in computer sciences. He is currently a member of the Crypto-Net Research Group, Hazara University. He was a Ph.D. Research with the NetLab-INMC, School of Electrical and Computer Engineering, Seoul National University, South Korea, in 2019, under the HEC Research Program. He is also serving as a Lecturer with the Department of Computer Sciences, IQRA National University, Peshawar. He has numerous publications in international conferences and journals. His research interests are information security, secure and smart cryptography, heterogeneous communications within the IoT, mobile ad-hoc networks, wireless sensor networks security, and fuzzy logic-based decision-making theory.

MAHDI ZAREEI (Senior Member, IEEE) received the M.Sc. degree in computer network from the University of Science, Malaysia, in 2011, and the Ph.D. degree from the Communication Systems and Networks Research Group, Malaysia-Japan International Institute of Technology, University of Technology, Malaysia, in 2016. In 2017, he joined the School of Engineering and Sciences, Tecnologico de Monterrey, as a Postdoctoral Fellow, where he has been as a Research Professor since 2019. His research mainly focuses on wireless sensor and ad-hoc networks, energy harvesting sensors, information security, and machine learning. He is a member of the Mexican National Researchers System (level I). He is also serving as an Associate Editor for IEEE Access and the *Ad Hoc & Sensor Wireless Networks* journals.

RIMSHA MANZOOR received the master's degree in computer science from Hazara University Mansehra, in 2018, where she is currently pursuing the M.S. degree in computer science. She is currently a Lecturer with the Zubaida Aman Government Girls Postgraduate College, Ghazi. Her research areas include image steganography and digital image processing.

HASSAM SAJID received the master's degree in computer science from Hazara University Mansehra, in 2018, where he is currently pursuing the M.S. degree in computer science. His fields of interest are digital image processing and image steganography.

FAISAL ALANAZI received the B.Sc. degree in electrical engineering (electronics and communication) from KSU and the M.Sc. and Ph.D. degrees in electrical and computer engineering from The Ohio State University in 2013 and 2018, respectively. He is currently an Assistant Professor with PSAU. His research interests span cryptography, vehicular ad-hoc networks, and delay tolerant networks. He is a member of the IEEE Communication Society.