PassPro: A secure password protection from the adversaries

Ripon Patgiri
National Institute of Technology Silchar
ripon@cse.nits.ac.in

Laiphrakpam Dolendro Singh
National Institute of Technology Silchar
ldsingh@cse.nits.ac.in

Abstract

In this paper, we present a client-side password hashing method, called PassPro. PassPro uses two secrets and a domain word to shuffle the strings. The shuffled strings are converted into hash values and sent to the identity manager for authentication or identity creation. The shuffling is based on a pseudo-random algorithm. The legitimate user can reproduce the shuffled string again. The hash values are encrypted in the password database with a different key for each user. Therefore, PassPro features- a) client-side password metering, b) client-side password hashing, c) prevention of the domino effect, d) protection of the password database from stealing, e) memory hardness, f) encryption of the hash values using a mutually reproducible secret key, and g) prevention of dictionary and guessing attacks. Also, PassPro guarantees that identity managers, including adversaries, cannot retrieve the original password and user ID of the user. Alternatively, the original user ID and password cannot be retrieved even if the password database is given to the adversary. Furthermore, the user ID and password of a password database are invalid in other domains, even if the same user ID and password are used in multiple domains.

1 Introduction

The emergence of Quantum Computing demands the revision of existing authentication mechanisms because authentication mechanisms protect precious digital assets. Moreover, conventional computers are equipped with more powerful processors and GPUs at low cost, and the attacking becomes easier for adversaries. Interestingly, the authentication mechanism is a widely used method to protect digital assets from being misused. Therefore, a diverse authentication mechanism has already been proposed, but the most popular method is a password-based authentication system. Password-based authentication mechanism has numerous challenges to be addressed, and many of the challenges are already addressed; for instance, the state-of-the-art password-based authentication system uses salt to defeat dictionary attacks [3]. The key challenge of the password-based authentication system is to prevent a guessing attack on the password. To overcome the password guessing attack, many researchers have already proposed diverse techniques; for instance, the transformation of password to context-free grammar [2]. Moreover, oPass defends against diverse attacks such as password stealing and password reuse attacks [7]. Importantly, the state-of-the-art identity manager uses OTP to protect from password stealing, password reuse, and phishing attacks. However, it does not ensure database stealing issues.

Definition 1 *A client sets the user ID as $\mathcal{U}$ and password as $\mathcal{P}$ in multiple domains. Once the $\mathcal{U}$ and $\mathcal{P}$ are compromised, then the adversary can use the identity to gain authentication in multiple domains. Moreover, the identity manager can easily misuse the identity of the user to gain authentication in multiple domains without having consent from the users. This phenomenon is called the domino effect.*

Definition 2 *A function $\mathcal{F}$ transforms an input string, ω, into an output, $\mathcal{O}$, due to the significant influence of another input, $\mathcal{I}$, and the $\mathcal{I}$ is called a context. Alternatively, $\mathcal{F}(\omega) \xrightarrow{\mathcal{I}} \mathcal{O}$ or $\mathcal{O} \xleftarrow{\mathcal{I}} \mathcal{F}(\omega)$.*

1.1 Motivation

Recent developments suggest that users' passwords are published widely, which is fatal for the password-based authentication system. COMB publishes 3279064312 passwords online, which is approximately 3.27 billion pairs of emails and passwords [4]. The report further highlights that 3.27 billion is 40% of the world's total population. Therefore, all passwords of the emails may be leaked because the total number of email users may be the same as approximately 3.27 billion, which shows our identity managers are highly vulnerable to adversaries. Similarly, RockYou2021 publishes a whooping 8.4 billion passwords (100GB text data) [5]. The total number of passwords is almost equivalent to the world's

total population. It shows that it is a sheer failure of state-of-the-art identity management systems. Therefore, it requires a solid security system to prevent such kinds of attacks.

The COMB and RockYou2021 show that the state-of-the-art identity manager stores raw forms of user IDs and passwords in their database, which is highly risky. However, most identity manager uses salt to defeat certain kinds of attacks, but it is not sufficient to prevent diverse attacks. Moreover, salt has several issues, such as short salt and salt reuse. Thus, the use of salt can be avoided. Moreover, the users' passwords are published widely by adversaries. Furthermore, the identity manager can reuse the passwords in different domains to gain authentication because the identities of the users are under the control of the identity manager. There is no solid mathematical proof or method to nullify these possibilities. Therefore, there are a few questions regarding the password-based authentication systems, which are given below-

- Should the identity manager store the users' passwords in raw string format?
- Is there any available method to prevent the domino effect of passwords?
- Should the user ID be public?
- Can we prevent the stealing issue of the password database?
- Can an identity manager keep the users' privacy intact?

The above-raised questions directly doubt the capabilities of state-of-the-art password-based authentication systems or identity managers. As we know that the most advanced identity manager stores raw string using salt or encryption, but it is insufficient and inefficient to protect against diverse attacks, such as password database stealing and the domino effect. To the best of our knowledge, there is no solid mechanism to prevent the domino effect (see Definition 1). Apparently, the most advanced identity managers violate the users' privacy. Therefore, it demands a new mechanism or method to address all the above-mentioned issues.

Figure 1 demonstrates the screenshot of resetting a password in GitHub.com. In the dialogue box, the user password is listed as passwords commonly used on other websites. GitHub.com checks the password strength and checks for quality passwords in published passwords online at [4, 5], which is the best practice to date. However, the users' passwords are published widely and freely online. Therefore, it indicates that almost all users' passwords are compromised. Importantly, the identity manager of any platform can easily utilize the user ID and password in different domains for gaining authentication, which is not desirable. Moreover, the user ID and password are controlled by other administrative staff (identity manager), which means that the password can be sold or published at their own will. Notably, we find no such

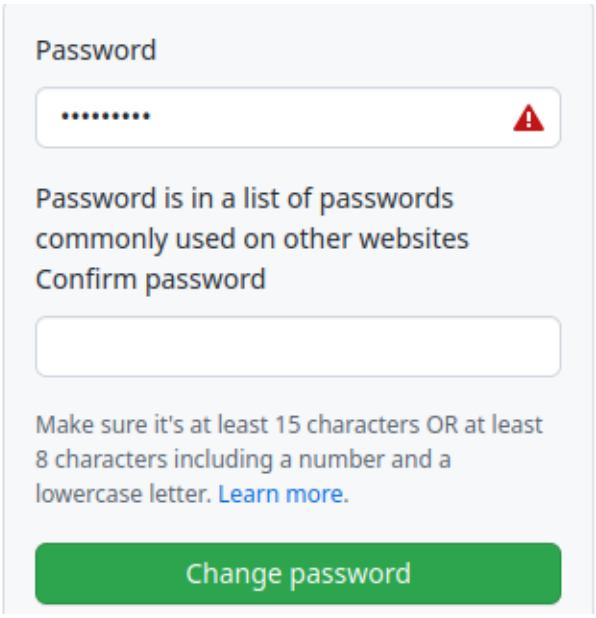

Figure 1: A dialogue box of GitHub.com for resetting the password.

evidence and have never reported such incidents to date, but it is possible and easy. The identity manager deals with the raw form of user identities and stores the identities at their own convenience. Apparently, it is an act of violation of the privacy of the users, which is common practice for password-based authentication systems and is accepted widely to date because there are no other identity managers to address the above-raised issues. For example, the identity manager of GitHub informs the users that the user's password is widely popular and widely listed in different domains. Therefore, it demands the prevention of violation of the users' privacy. Specifically, how does a client feel if the client's password is widely listed as a password in multiple domains or is listed at [4, 5]? The clients' passwords are published by adversaries. Therefore, the identity manager is unable to protect the identities of the users.

In the above-raised issues, the password stretching algorithm does not have any effect. However, it can defeat the guessing attacks. To overcome the password guessing attack, many researchers have already proposed diverse techniques; For instance, the transformation of password to context-free grammar [2]. Moreover, oPass defends against diverse attacks such as password stealing and password reuse attacks [7]. Importantly, the state-of-the-art identity manager uses OTP to protect against phishing attacks which are highly effective till the bulk SMS gateway is not compromised. Therefore, it demands a secure identity management system that guarantees that even if the database is stolen by adversaries, the adversaries are unable to gain authentication.

1.2 Our proposed work

The recent password-hashing competition winner was Argon2, where Argon2i is suitable for password hashing [1]. Argon2 features memory hardness for password hashing. It is designed to hash the password on the server side, which contrasts our proposed work. Our proposed work features client-side password hashing, called PassPro. Our key objectives are outlined below-

- To prevent getting the direct raw form of the user IDs and passwords by the adversary, including the identity manager.
- To ensure that the adversary can never retrieve the original user IDs and passwords, including the identity manager.
- To prevent the domino effect.
- To protect the password database from stealing.

1.3 Key contributions

- PassPro completely prevents the domino effect. It shuffles the user ID, password, and domain word to produce three unique strings. Therefore, the same user ID and password can be used in multiple domains without worrying about the domino effect.
- PassPro is designed based on the client-side password hashing to conceal the original user ID and password. It ensures that the adversary, including the identity manager, never gets the original user ID and password in raw form.
- Identity manager of PassPro never receives the original user ID and password in raw form; therefore, it features client-side password metering. Alternatively, the server (identity manager) can never suggest the strength of a password or user ID. Therefore, it invalidates the use of AJAX in the framework.
- User ID and password of each user are encrypted with a different key in the password database of the identity manager. The identity manager computes a mutually reproducible secret key using the hash value provided by the user and the master secret word from the identity manager.
- PassPro ensures the protection of users' identity from stealing by adversaries. The user ID and password are encrypted with different keys; therefore, the adversary cannot decrypt the user ID and password even if the entire password database of the identity manager is with the adversary.
- PassPro ensures prevention from the dictionary and guessing attacks by using two secrets: user ID and password.

1.4 Our results

PassPro is different than conventional password-based authentication systems that rely on the memory-hard secure hash algorithms, VORSHA [6]. Moreover, PassPro features client-side password hashing rather than hashing a password on the server-side (identity manager). We present our results in the next sub-subsections.

1.4.1 Shuffling with a context

The user enters the user ID ($\mathcal{U}$) and password ($\mathcal{P}$) for authentication. PassPro retrieves a domain word ($\mathcal{D}$) to shuffle the user ID and password. The domain word is public and available to all, including the adversary. We shuffle the strings- a) the $\mathcal{U}$, and $\mathcal{D}$ in the context (see Definition 2) of the $\mathcal{P}$ to produce $\mathscr{U}$ and b) the $\mathcal{P}$ and $\mathcal{D}$ in the context of the $\mathcal{U}$ to produce $\mathscr{P}$. Moreover, we also shuffle $\mathcal{U}$, $\mathcal{P}$ and $\mathcal{D}$ to produce $\mathscr{K}$. Therefore, shuffling produces a shuffled user ID ($\mathscr{U}$), a shuffled password ($\mathscr{P}$), and part of a mutually reproducible secret key ($\mathscr{K}$). The shuffling algorithm produces different shuffled user IDs and passwords-

- For the same user IDs and passwords for different domains.
- For the same user IDs and domains for different passwords.
- For the same passwords and domains for different user IDs.

On the contrary, the original user ID and password can be retrieved from the $\mathscr{U}$ and $\mathscr{P}$ by removing the characters of the domain word. Thus, the identity manager can easily reconstruct the user ID and password of the client, which is not desirable because the identity manager can misuse the user IDs and passwords in different domains to gain authentication. Moreover, adversaries can steal the password database from the identity manager. Furthermore, it does not ensure the domino effect of the password-based authentication system. Therefore, it requires client-side hashing to hide the original user ID and password.

1.4.2 Client-side hashing

A hash function can be invoked at the client's computers to convert shuffled user ID, password, and domain word into hash values. Our proposed system relies on VORSHA [6] which is a variable-sized and randomized secure hash algorithm. Therefore, it prevents the transmission of the original password and user ID to the identity manager. Hence, the

identity manager cannot retrieve the actual user ID and password from the hash values. The hash values of the shuffled strings cannot be the same for the different domains, even if the password and user ID are the same.

1.4.3 Domino-effect

The domino effect is a crucial effect to be prevented due to the adversary's presence in password-based authentication systems. The state-of-the-art password-based authentication system cannot prevent the domino effect. PassPro deals with hash values with a shuffled user ID and password. Therefore, it is computationally infeasible to reconstruct the original string from the hash values $\mathcal{H}_{\mathcal{U}}$ and $\mathcal{H}_{\mathcal{W}}$. This method ensures the prevention of the domino effect because the different domain words create different shuffled hash values for the same user ID and password. Thus, it permits the reuse of user IDs and passwords in multiple domains.

1.4.4 Encryption and decryption

The user hash values are encrypted with a mutually reproducible secret key to store in the password database. Alternatively, each user's hash values are encrypted with a different secret key. The keys are derived by a pseudo-random number generator. The algorithm requires a hash value from the user and a master secret word from the identity manager to produce or reproduce the secret key to encrypt or decrypt the hash values on the server side. The user supplies a hash value of the shuffled string to the identity manager, where shuffling is performed among the user ID, password, and domain word without any context. The hash values are encrypted using a mutually reproducible secret key. Therefore, the identity manager cannot reproduce the mutually reproducible secret key without the hash value from the user. Besides, the user cannot reproduce the mutually reproducible secret key without the master secret word from the identity manager. PassPro encrypts the individual shuffled-then-hashed user ID and password with an individual secret key. Alternatively, the identity manager is unable to decrypt the stored hash values without having consent from the user. Similarly, the user cannot decrypt its own hash values from the password database without permission from the identity manager.

1.4.5 Password database stealing

The adversary always tries to steal the password database to gain authentication in multiple domains. It requires strict measures to prevent such kinds of attacks. Therefore, we introduce a mutually reproducible secret key to encrypt and decrypt. Consequently, it prevents the identity manager from decryption the hash values without having permission from the user. Similarly, the user cannot decrypt the encrypted hash values from the password database without having permission from the identity manager, even if the identity manager provides the user's own encrypted hash values to the user. With the above-mentioned condition, the adversary is unable to decode the encrypted hash values even if the adversary evades the security of the identity manager to steal the database. Therefore, it is computationally infeasible to decrypt all the encrypted hash values from the password database from the identity manager. Therefore, the user IDs and passwords are intact even after the password database is stolen by the adversary.

1.4.6 Guessing and dictionary attacks

The passwords are created eight characters long, using capital letters, small letters, digits, and special symbols, which creates huge password spaces. However, a user always chooses easy-to-remember passwords; thus, the password space becomes small. Thus, the guessing attacks become a reality. Also, the dictionary size of the adversary becomes smaller. For instance, almost all users' passwords are published at [4, 5]. Therefore, it becomes easy for adversaries to perform the attacks. Hence, it demands a new measurement to thwart such kinds of attacks. Our proposed system uses two secret words, namely, user ID and password. Consequently, the guessing and dictionary attacks become harder even if the passwords are published at [4, 5].

2 PassPro: The proposed system

In a password-based authentication system, the user enters a user ID and password to prove the genuineness. However, there are many issues with conventional password-based authentication systems. The prominent issue is the domino effect. People often use the same password and user ID for various identity managers such that they can remember them easily. Therefore, if the password is compromised, then all identity managers can be broken by the adversary. Moreover, conventional identity managers store the password in raw format. Apparently, it violates the privacy of the users. Therefore, an adversary can steal the passwords from the password database from the identity manager, and the evidence is [4, 5]. Therefore, our key objectives are as given below-

- To provide strict security measurement for authentication without causing inconvenience to the users.
- To protect the users' identities from stealing and publishing by adversaries.
- To protect users' privacy from the identity manager.
- To prevent the domino effect such that a user can use the same user ID and password in multiple domains.

Our proposed system relies on existing OTP mechanisms to identify the desired user. Also, it relies on existing captcha to differentiate between robots and humans. Moreover, it relies on existing cryptography techniques, such as public key

Table 1: Important symbols/notations used in the paper and their description.

Symbols	Description
$\mathbb{A}$, $\mathbb{B}$	Example of two users.
$\mathcal{U}$	The user ID in raw foramt.
$\mathcal{P}$	The password in raw format.
$\mathcal{D}$	The domain word; for instance, usenix.org.
$\mathscr{U}$	The shuffled user ID with domain word.
$\mathscr{P}$	The shuffled password with domain word.
$\mathscr{K}$	The shuffled string with the user ID, password, and domain word.
$\mathcal{H}_{\mathscr{U}}$	The hash value of shuffled user ID.
$\mathcal{H}_{\mathscr{P}}$	The hash value of shuffled password.
$\mathcal{H}_{\mathscr{K}}$	The hash value of shuffled string.
$\mathcal{H}()$	Denotes a VORSHA-3D hash function.
$\mathcal{W}$	A single master secret word of the identity manager for the whole database.
$\mathfrak{K}$	The mutually reproducible secret key.
$\mathcal{PR}_{\mathbb{A}}$	The private key of user $\mathbb{A}$.
$\mathcal{PU}_{\mathbb{A}}$	The public key of user $\mathbb{A}$.
$\mathcal{SK}$	The client and server mutually compute a shared secret key.
$\mathcal{H}_{\mathscr{U}}^{received}$	Received hash value from the user for user ID.
$\mathcal{H}_{\mathscr{P}}^{received}$	Received hash value from the user for password.
$\mathcal{H}_{\mathscr{U}}^{decrypted}$	Decrypted hash value from the password database for the user ID.
$\mathcal{H}_{\mathscr{P}}^{decrypted}$	Decrypted hash value from the password database for password.
$\mathcal{E}_{\mathbb{A}}$	Email of user $\mathbb{A}$.
$\cup$	Denotes the shuffling process.
$\cap$	Denotes the reversal of the shuffling process.
$\mathcal{S}$	A seed value of 32-bits.
τ	Number of iterations to be performed while computing a seed value.
ω	Denotes a string.
$\wedge$	Denotes bit-wise AND operation.
$\gg$	Denotes bit-wise right shift operation.
$\ll$	Denotes bit-wise left shift operation.

cryptography for non-repudiation and symmetric key cryptography for communication between the client and server.

2.1 Identity creation

Table 2 shows the identity creation at the identity manager by a user. In our proposed system, email ID and user ID is treated differently. The email ID is public, and the user ID is a secret word. Initially, the user retrieves a domain word and enters a user ID and password. On the client side, the user ID, password, and domain word are shuffled to create a unique identity. The shuffled strings are converted into hash values by the client. The hash value sizes are variable. The correct hash value with a correct length can be reproduced exclusively by the legitimate user [6]. These hash values are encrypted using the private key of the user for non-repudiation. Again, the user encrypts using a shared secret key and sends these hash values to the identity manager. The identity manager decrypts the hash values and produces a mutually reproducible secret key. The hash values are inserted into the password database of the identity manager against the user email ID by encrypting these hash values by the generated mutually reproducible secret key.

2.2 Authentication process

Table 3 shows a similar procedure as the identity creation for the authentication process. The identity manager reproduces a mutually reproducible secret key using a hash value from the user and a master secret word from the identity manager. The mutually reproducible secret key is used to decrypt the stored hash value. The supplied hash values and decrypted hash values are compared for authentication.

2.3 User ID

A user ID can be chosen from the publicly available user ID, for instance, mail@example.com. It is converted into a secret entity by shuffling with a publicly available domain word. However, if a user ID is secret, then it is stronger, and it can protect against diverse attacks. A weak password can reveal the secret of shuffling. Therefore, our recommendation is to keep the secret user ID as like password other than the email ID. The conventional system uses the user ID as a publicly visible identifier. Our proposed scheme treats the user ID as equally important as the password. Therefore, we suggest the following rules for user ID creation to address the above-mentioned objectives-

- It should contain at least one capital letter, one small letter, a digit, and a symbol.
- The minimum length of the user ID is eight.
- The user ID cannot be an email ID.

A weak user ID can be created using the new user ID formation rules, for instance, Abc@1234. There is an inconvenience in remembering a user ID. Therefore, the user always chooses their user ID with their name, for instance, Alan@2023. A user can select the identity in an easy manner. Consequently, it makes it difficult for an adversary to guess with the combination of passwords. Therefore, PassPro uses user ID as a secret word similar to a password.

Table 2: Step-by-step overview of the identity creation of a user.

Client	**Identity Manager**
• User and identity manager computes shared secret key $\mathcal{SK}$. • User retrieves a domain words $\mathcal{D}$. • User enters user ID $\mathcal{U}$ and password $\mathcal{P}$. • The user performs shuffling- $\mathcal{U} \cup \mathcal{D} \xrightarrow{\mathcal{P}} \mathscr{U}$, $\mathcal{P} \cup \mathcal{D} \xrightarrow{\mathcal{U}} \mathscr{P}$, $\mathcal{U} \cup \mathcal{P} \cup \mathcal{D} \rightarrow \mathscr{K}$ • The user performs hashing- $\mathcal{H}_{\mathscr{U}} \leftarrow \mathcal{H}(\mathscr{U})$, $\mathcal{H}_{\mathscr{P}} \leftarrow \mathcal{H}(\mathscr{P})$, $\mathcal{H}_{\mathscr{K}} \leftarrow \mathcal{H}(\mathscr{K})$ • The user encrypts the hash values using its own private key- $Enc(\mathcal{H}_{\mathscr{U}}, \mathcal{PR}_{\mathbb{A}})$, $Enc(\mathcal{H}_{\mathscr{P}}, \mathcal{PR}_{\mathbb{A}})$, $Enc(\mathcal{H}_{\mathscr{K}}, \mathcal{PR}_{\mathbb{A}})$, where $\mathcal{PR}$ is a private key of the user $\mathbb{A}$. • The user encrypts the hash values using shared secret key to send them to the identity manager- $Enc(Enc(\mathcal{H}_{\mathscr{U}}, \mathcal{PR}_{\mathbb{A}}), \mathcal{SK})$, $Enc(Enc(\mathcal{H}_{\mathscr{P}}, \mathcal{PR}_{\mathbb{A}}), \mathcal{SK})$, $Enc(Enc(\mathcal{H}_{\mathscr{K}}, \mathcal{PR}_{\mathbb{A}}), \mathcal{SK})$ • The user sends the encrypted hash values to the identity manager.	
	• Identity manager decrypted the received hash values- $\mathcal{H}_{\mathscr{U}}^{received} \leftarrow Dec(Dec(Enc(Enc(\mathcal{H}_{\mathscr{U}}, \mathcal{PR}_{\mathbb{A}}), \mathcal{SK}), \mathcal{SK}), \mathcal{PU}_{\mathbb{A}})$ $\mathcal{H}_{\mathscr{P}}^{received} \leftarrow Dec(Dec(Enc(Enc(\mathcal{H}_{\mathscr{P}}, \mathcal{PR}_{\mathbb{A}}), \mathcal{SK}), \mathcal{SK}), \mathcal{PU}_{\mathbb{A}})$ $\mathcal{H}_{\mathscr{K}}^{received} \leftarrow Dec(Dec(Enc(Enc(\mathcal{H}_{\mathscr{K}}, \mathcal{PR}_{\mathbb{A}}), \mathcal{SK}), \mathcal{SK}), \mathcal{PU}_{\mathbb{A}})$, where $\mathcal{PU}_{\mathbb{A}}$ is the public key of the user $\mathbb{A}$. • Identity manager computes a mutually reproducible secret key $\mathfrak{K}$ using $\mathcal{H}_{\mathscr{K}}$ and a master secret word from the identity manager $\mathcal{W}$, i.e., $\mathfrak{K} \xleftarrow{\mathcal{W}} \mathcal{G}(\mathcal{H}_{\mathscr{K}}^{received})$. • The identity manager encrypts the hash values- $Enc(\mathcal{H}_{\mathscr{U}}, \mathfrak{K})$, $Enc(\mathcal{H}_{\mathscr{P}}, \mathfrak{K})$ • The identity manager inserts these encrypted hash values into the password database and discards $\mathcal{H}_{\mathscr{K}}^{received}$ and $\mathfrak{K}$.

The user ID is constituted of eight characters from 26 small-case letters, 26 upper-case letters, 10 digits, and 32 symbols (excluding white space). Therefore, by choosing eight characters from 94 available characters, the maximum number of possible user IDs is

$$\binom{94}{8} = 111,315,063,717 \tag{1}$$

We have eight character strings consisting of alphabets, digits, and special symbols. Therefore, we can form 111 billion user IDs. Theoretically, it is quite large enough to thwart diverse attacks, but it cannot withstand guessing attacks and dictionary attacks.

2.4 Password

We suggest the following rules to follow in the creation of the password-

- Password length must be at least ten characters long.
- It should contain at least a capital letter, a small letter, a digit, and a special symbol.
- Email ID cannot be the password.
- User ID and password cannot be the same.

It is similar to conventional passwords with a length of a minimum of ten characters. COMB and RockYou2021 have all the passwords having a length of ten characters or more. Therefore, it is easy to perform guessing attacks.

However, the password is constituted of ten characters from 26 small-case letters, 26 upper-case letters, 10 digits, and 32 symbols (excluding white space). Therefore, by choosing ten characters from 94 available characters, the maximum number of possible passwords is

$$\binom{94}{10} = 9,041,256,841,903 \tag{2}$$

The challenge is to defeat the guessing and dictionary attacks. Theoretically, a password with ten characters can withstand

Table 3: Step-by-step overview of the authentication of a user.

Client	Identity Manager
• The same procedure is applied in authentication the same as Table 2. • The user performs encryption on the hash values to send them to the identity manager for authentication- $Enc(Enc(\mathcal{H}_{\mathcal{U}}, \mathcal{PR}_{\mathbb{A}}), \mathcal{SK})$, $Enc(Enc(\mathcal{H}_{\mathscr{P}}, \mathcal{PR}_{\mathbb{A}}), \mathcal{SK})$, $Enc(Enc(\mathcal{H}_{\mathscr{K}}, \mathcal{PR}_{\mathbb{A}}), \mathcal{SK})$ • The user sends the encrypted hash values to the identity manager.	
	• Identity manager decrypted the received hash values- $\mathcal{H}_{\mathcal{U}}^{received} \leftarrow Dec(Dec(Enc(Enc(\mathcal{H}_{\mathcal{U}}, \mathcal{PR}_{\mathbb{A}}), \mathcal{SK}), \mathcal{SK}), \mathcal{PU}_{\mathbb{A}})$ $\mathcal{H}_{\mathscr{P}}^{received} \leftarrow Dec(Dec(Enc(Enc(\mathcal{H}_{\mathscr{P}}, \mathcal{PR}_{\mathbb{A}}), \mathcal{SK}), \mathcal{SK}), \mathcal{PU}_{\mathbb{A}})$ $\mathcal{H}_{\mathscr{K}}^{received} \leftarrow Dec(Dec(Enc(Enc(\mathcal{H}_{\mathscr{K}}, \mathcal{PR}_{\mathbb{A}}), \mathcal{SK}), \mathcal{SK}), \mathcal{PU}_{\mathbb{A}})$ • Identity manager computes a mutually reproducible secret key $\mathfrak{K}$ using $\mathcal{H}_{\mathscr{K}}$ and a master secret word from the identity manager $\mathcal{W}$, i.e., $\mathfrak{K} \xleftarrow{\mathcal{W}} \mathcal{G}(\mathcal{H}_{\mathscr{K}}^{received})$. • The identity manager decrypts the stored hash values- $\mathcal{H}_{\mathcal{U}}^{decrypted} \leftarrow Dec(Enc(\mathcal{H}_{\mathcal{U}}, \mathfrak{K}), \mathfrak{K})$, $\mathcal{H}_{\mathscr{P}}^{decrypted} \leftarrow Dec(Enc(\mathcal{H}_{\mathscr{P}}, \mathfrak{K}), \mathfrak{K})$ • The identity manager compares the hash values for authentication- $\mathcal{H}_{\mathcal{U}}^{received} = \mathcal{H}_{\mathcal{U}}^{decrypted}$, and $\mathcal{H}_{\mathcal{U}}^{received} = \mathcal{H}_{\mathcal{U}}^{decrypted}$ • The identity manager discards $\mathcal{H}_{\mathscr{K}}^{received}$ and $\mathfrak{K}$.

diverse attacks by above mentioned rules; however, it cannot withstand dictionary and guessing attacks even if the password space is large.

2.5 Guessing attacks

Our proposed system comprises two secrets, namely, user ID and passwords. We assume that all user IDs and passwords are listed in COMB or RockYou2021. Therefore, it is easy to perform a guessing or dictionary attack on either the user ID or the password. Let the user ID and passwords be in the COMB dictionary (the size of COMB is less than the size of RockYou2021), i.e., $\mathcal{U}, \mathcal{P} \in COMB$. But both $\mathcal{U}$ and $\mathcal{P}$ must belong to the same user and be correct. Therefore, the probability of selecting the correct user ID and password from COMB that belongs to a single user is as given below

$$\frac{1}{\binom{3279064312}{2}} = \frac{1}{5376131379476484516} \tag{3}$$

which is fairly small enough to thwart guessing attacks or dictionary attacks that motivate us why we should consider the user ID as a secret word. Apparently, the two secret words can cause inconvenience to the users. However, we need to prevent diverse attacks such as password database stolen issues. Thus, the inconvenience is justified. But it cannot ensure freedom from the domino effect. Moreover, it cannot protect the password database from being stolen. Therefore, we use a domain word to prevent such kinds of attacks.

2.6 Domain Word

A user needs to retrieve the domain word, and the domain word is shuffled with the user ID and password of the user. The domain word is public and available to all, including the adversary. For instance, the domain words are usenix.org, google.com, gmail.com, iacr.org, etc. These words are mixed with the user ID and password to prevent diverse attacks.

2.7 Shuffling the strings

Algorithm 1 Computing seed value for utilization in the hash function.

```
1: procedure GETSEEDVALUE(ω, 𝓛, 𝒮, τ)
2:     for i : 1 to τ do
3:         𝒮 = PRIMARYHASH(ω, 𝓛, 𝒮)
4:     end for
5:     return 𝒮
6: end procedure
```

Algorithm 1 computes the seed value using a primary hash function. A primary hash function is a non-secure hash func-

tion that produces a hash value of β-bit, for instance, the murmur hash function. The function iterates τ times to alter the seed value. Initially, the seed value is public, and it is converted into a private value using a user ID, password, and domain word.

Algorithm 2 Left shifting a string to remove the i^{th} character from ω.

```
1: procedure SHIFTLEFT(ω, i, ℒ)
2:     for j : i to ℒ do
3:         ω[j] = ω[j+1]
4:     end for
5: end procedure
```

The user often uses the same user ID and password over multiple domains; therefore, once the password is compromised, all other domains can be accessed by the adversary. Therefore, we shuffle the user ID and password with a domain word. The domain word is public; however, the legitimate user can correctly reproduce the shuffled word. Algorithm 3 portrays the shuffling between two words with a context (see Definition 2 for context). The word ω_1 and ω_2 are shuffled with the context of the word ω_3.

Table 4 shows the example of shuffling two words using a context, and the inputs are taken as an example for demonstration purposes. PassPro requires three input strings, namely, user ID, password, and domain word. The domain word is shuffled with a user ID and password to prevent the domino effect. It shows that two different password makes different shuffled string for user ID and password in test case 1 and 2, as shown in Table 4. Moreover, the shuffled strings for user ID and password produce a different output in test case 1 and 3 if we alter the domain word. Also, if we change the user ID, then the shuffled string for the user ID and password are also changed, shown in test case 1 and test case 4. Therefore, we can conclude as follows-

- Different passwords with the same user ID and domain word translate into different shuffled user IDs and passwords.
- Different domain names with the same user ID and password translate into different shuffled user IDs and Passwords.
- Different user IDs with the same password and domain word translate into different shuffled user IDs and Passwords.

PassPro creates different user IDs and Passwords for different user IDs, passwords, and domain words. Therefore, PassPro completely eradicates the domino effect issue in password-based authentication systems. Now, the server can easily reconstruct the user ID and password from the shuffled user ID and password by removing the characters from the domain

Algorithm 3 Shuffling two strings, ω_1 and ω_2, with respect to word ω_3. The ω_3 is the context of the shuffling. This shuffling follows a pseudo-random algorithm.

```
1: procedure SHUFFLE(ω1, ω2, ω3, S)
2:     ℒω1, ℒω4 = LENGTH(ω1)
3:     ℒω2, ℒω5 = LENGTH(ω2)
4:     ℒω3, ℒω6 = LENGTH(ω3)
5:     COPY(ω4, ω1)                    ▷ Copy word ω1 to ω4
6:     COPY(ω5, ω2)                    ▷ Copy word ω2 to ω5
7:     COPY(ω6, ω3)                    ▷ Copy word ω3 to ω5
8:     τ = 16, δ = 37, μ = 16
9:     S = GETSEEDVALUE(ω1, ℒω1, S, τ)
10:    τ = S%δ + μ
11:    S = GETSEEDVALUE(ω2, ℒω2, S, τ)
12:    τ = S%δ + μ
13:    S = GETSEEDVALUE(ω2, ℒω3, S, τ)
14:    τ = S%δ + μ
15:    while ℒω1 ≠ 0 and ℒω2 ≠ 0 do
16:        if S ∧ 1 = 0 then
17:            S = GETSEEDVALUE(ω2, ℒω2, S, τ)
18:            τ = S%δ + μ
19:            S = GETSEEDVALUE(ω1, ℒω1, S, τ)
20:            τ = S%δ + μ
21:            S = GETSEEDVALUE(ω3, ℒω3, S, τ)
22:            τ = S%δ + μ
23:            k = S%ℒω4
24:            buff[i++] = ω4[k]
25:            SHIFTLEFT(ω4, k, ℒω4)
26:            ℒω4 = ℒω4 − 1
27:        else
28:            S = GETSEEDVALUE(ω2, ℒω2, S, τ)
29:            τ = S%δ + μ
30:            S = GETSEEDVALUE(ω1, ℒω1, S, τ)
31:            τ = S%δ + μ
32:            S = GETSEEDVALUE(ω3, ℒω3, S, τ)
33:            τ = S%δ + μ
34:            k = S%ℒω5
35:            buff[i++] = ω5[k]
36:            SHIFTLEFT(ω5, k, ℒω5)
37:            ℒω5 = ℒω5 − 1
38:        end if
39:    end while
40:    for k : 0 to ℒω4 do             ▷ Copying the remaining characters
41:        buff[i++] = ω4[k]
42:    end for
43:    for k : 0 to ℒω5 do             ▷ Copying the remaining characters
44:        buff[i++] = ω5[k]
45:    end for
46: end procedure
```

Table 4: Example of shuffling two input strings with respect to a context.

Test case	Context	Shuffling for	String type	Input String	Shuffled string	256-bit hash value using VORSHA [6]
1	Pass@1975	User ID	User ID	Chair@2023	1C7ira5xu9@ oehsni.rg	34f62aaf7f1f352e8c62fa234f9d05a97 9db1a9c2f882ae756c332c275c8faac
			Domain	usenix.org		
	Chair@2023	Password	Password	Pass@1975	.urnieg59oxsP sas@17	c4d05193bab9510b916e65bb81a35e7 6f76a00a4e3760eff9a1c1b1555d83ff0
			Domain	usenix.org		
2	Abc@1975	User ID	User ID	Chair@2023	i9.7urh@gsax oien1rC5	3b6b1c488bc5e8f0e5a2c73c50bd2cec 69f7ba334327696474c22ad55c5ea080
			Domain	usenix.org		
	Chair@2023	Password	Password	Abc@1975	co7rAisxb9e@ 1n5u.g	58ff31bd7c4a0520894bbb2ced233098 39988d17fb0724d05f4a105f5bcaac6a
			Domain	usenix.org		
3	Pass@1975	User ID	User ID	Chair@2023	o79x.1aac@5 rpCilhemem	75e9eac6717e7b7300b526f7516b9da2 5b70bf8adb6252f1db66123f7dc63450
			Domain	example.com		
	Chair@2023	Password	Password	Pass@1975	seaaslm@7oe. 9xp1cmP5	f8b7b9abd60afec5709a58eaac0318bb 8574f1eb4fc0bc8458919afaf7c684ed
			Domain	example.com		
4	Pass@1975	User ID	User ID	Author@2023	ero@.nitmh5sx cuAuo197	d9498984e3353f91cac2a4f6b13e5a4a 89d27ab5cbdbd4394b734bea6ca699c6
			Domain	usenix.org		
	Author@2023	Password	Password	Pass@1975	7mPis5soans@ 91uex.c	09b70ab3aba868241e331d613c981304 d7312c5498a450f7bc558796d1c7df69
			Domain	usenix.org		

words. Also, the adversary can get the user IDs and passwords if the adversary is able to steal the entire database. Therefore, PassPro strongly discourages storing the raw form of user ID and password in the identity manager. Therefore, it demands client-side password hashing, which is described in the next subsection.

2.8 Client-side hashing

PassPro does not transmit the raw password and user ID for storing it in the password database in the identity manager. Instead, PassPro uses client-side password hashing for transmission. Firstly, the client shuffles the user ID and password with a domain word. Secondly, the client of PassPro converts the shuffled user ID and password into two different hash values for transmission to the identity manager. The identity manager of PassPro stores the hash values of the user ID and password for future authentication. The hash value of the same user ID and password is invalid for the different domains. Therefore, these hash values cannot be used in other domains to gain authentication

For illustration, the $\mathscr{U}$, $\mathscr{P}$ and $\mathscr{K}$ are constructed by shuffling $\mathcal{U}$ and $\mathcal{P}$ using a domain word $\mathcal{D}$. Equation (4) shows the shuffling of two strings with a context and the shuffling of three strings without a context.

$$\begin{aligned} \mathcal{U} \cup \mathcal{D} &\xrightarrow{\mathcal{P}} \mathscr{U} \\ \mathcal{P} \cup \mathcal{D} &\xrightarrow{\mathcal{U}} \mathscr{P} \\ \mathcal{U} \cup \mathcal{P} \cup \mathcal{D} &\rightarrow \mathscr{K} \end{aligned} \tag{4}$$

where $\cup$ denotes the shuffling of the strings pseudo-randomly. Equation (4) shows the construction of the shuffled string, and therefore, the following Equation holds

$$\begin{aligned} \mathscr{U} \cap \mathcal{D} &\rightarrow \mathcal{U} \\ \mathscr{P} \cap \mathcal{D} &\rightarrow \mathcal{P} \end{aligned} \tag{5}$$

where $\cap$ denotes the reversal of the shuffling process. Equation (5) shows the retrieval of the original string from the shuffled string. Thus, we hash the shuffled string such that the original string can never be reconstructed from the hash value, which is given as follows-

$$\begin{aligned} \mathcal{H}(\mathscr{U}) &\rightarrow \mathcal{H}_{\mathscr{U}} \\ \mathcal{H}(\mathscr{P}) &\rightarrow \mathcal{H}_{\mathscr{P}} \\ \mathcal{H}(\mathscr{K}) &\rightarrow \mathcal{H}_{\mathscr{K}} \end{aligned} \tag{6}$$

It is guaranteed that the original string is not possible to reconstruct from $\mathcal{H}_{\mathscr{U}}$, $\mathcal{H}_{\mathscr{U}}$, and $\mathcal{H}_{\mathscr{K}}$. Thus, the hash values $\mathcal{H}_{\mathscr{U}}$, $\mathcal{H}_{\mathscr{U}}$, and $\mathcal{H}_{\mathscr{K}}$ are sent to the identity manager for either identity creation or authentication.

We show that two different users, let $\mathbb{A}$ and $\mathbb{B}$, cannot produce the same hash values. There are two cases, a) two different user IDs and the same passwords, and b) two different passwords and the same user IDs. We take the first case by taking two different user IDs as $\mathcal{U}_1 \neq \mathcal{U}_2$, and they are non-empty. The shuffling of the client $\mathbb{A}$'s user ID and password are given in Equation (7).

$$\begin{aligned} \mathcal{U}_1 \cup \mathcal{D} &\xrightarrow{\mathcal{P}} \mathscr{U}_1 \\ \mathcal{P} \cup \mathcal{D} &\xrightarrow{\mathcal{U}_1} \mathscr{P}_1 \\ \mathcal{U}_1 \cup \mathcal{P} \cup \mathcal{D} &\rightarrow \mathscr{K}_1 \end{aligned} \tag{7}$$

The hash values of the client $\mathbb{A}$'s user ID and password are

given in Equation (8).

$$\begin{aligned} \mathcal{H}(\mathscr{U}_1) &\rightarrow \mathcal{H}_{\mathscr{U}_1} \\ \mathcal{H}(\mathscr{P}_1) &\rightarrow \mathcal{H}_{\mathscr{P}_1} \\ \mathcal{H}(\mathscr{K}_1) &\rightarrow \mathcal{H}_{\mathscr{K}_1} \end{aligned} \tag{8}$$

The shuffling of the client $\mathbb{B}$'s user ID and password are shown in Equation (9).

$$\begin{aligned} \mathcal{U}_2 \cup \mathcal{D} &\xrightarrow{\mathcal{P}} \mathscr{U}_2 \\ \mathcal{P} \cup \mathcal{D} &\xrightarrow{\mathcal{U}_2} \mathscr{P}_2 \\ \mathcal{U}_2 \cup \mathcal{P} \cup \mathcal{D} &\rightarrow \mathscr{K}_2 \end{aligned} \tag{9}$$

The hash values of the client $\mathbb{B}$'s user ID and password are given in Equation (10).

$$\begin{aligned} \mathcal{H}(\mathscr{U}_2) &\rightarrow \mathcal{H}_{\mathscr{U}_2} \\ \mathcal{H}(\mathscr{P}_2) &\rightarrow \mathcal{H}_{\mathscr{P}_2} \\ \mathcal{H}(\mathscr{K}_2) &\rightarrow \mathcal{H}_{\mathscr{K}_2} \end{aligned} \tag{10}$$

From Equation (8) and (10), we can draw the Equation (11).

$$\begin{aligned} \mathcal{H}_{\mathscr{U}_1} &\neq \mathcal{H}_{\mathscr{U}_2} \\ \mathcal{H}_{\mathscr{P}_1} &\neq \mathcal{H}_{\mathscr{P}_2} \\ \mathcal{H}_{\mathscr{K}_1} &\neq \mathcal{H}_{\mathscr{K}_2} \end{aligned} \tag{11}$$

Therefore, two users with different user IDs with the same password always create different hash values. Similarly, two different users having the same user ID but different passwords also create different hash values. Let the password be $\mathcal{P}_1 \neq \mathcal{P}_2$, and these are non-empty. The shuffled strings of the client $\mathbb{A}$ are given in Equation (12).

$$\begin{aligned} \mathcal{U} \cup \mathcal{D} &\xrightarrow{\mathcal{P}_1} \mathscr{U}_3 \\ \mathcal{P}_1 \cup \mathcal{D} &\xrightarrow{\mathcal{U}} \mathscr{P}_3 \\ \mathcal{U} \cup \mathcal{P}_1 \cup \mathcal{D} &\rightarrow \mathscr{K}_3 \end{aligned} \tag{12}$$

The hash values of the client $\mathbb{A}$ are created using Equation (13).

$$\begin{aligned} \mathcal{H}(\mathscr{U}_3) &\rightarrow \mathcal{H}_{\mathscr{U}_3} \\ \mathcal{H}(\mathscr{P}_3) &\rightarrow \mathcal{H}_{\mathscr{P}_3} \\ \mathcal{H}(\mathscr{K}_3) &\rightarrow \mathcal{H}_{\mathscr{K}_3} \end{aligned} \tag{13}$$

The shuffled strings of the client $\mathbb{B}$ are given in Equation (14).

$$\begin{aligned} \mathcal{U} \cup \mathcal{D} &\xrightarrow{\mathcal{P}_2} \mathscr{U}_4 \\ \mathcal{P}_2 \cup \mathcal{D} &\xrightarrow{\mathcal{U}} \mathscr{P}_4 \\ \mathcal{U} \cup \mathcal{P}_2 \cup \mathcal{D} &\rightarrow \mathscr{K}_4 \end{aligned} \tag{14}$$

The hash values of the client $\mathbb{B}$ are given in Equation (14).

$$\begin{aligned} \mathcal{H}(\mathscr{U}_4) &\rightarrow \mathcal{H}_{\mathscr{U}_4} \\ \mathcal{H}(\mathscr{P}_4) &\rightarrow \mathcal{H}_{\mathscr{P}_4} \\ \mathcal{H}(\mathscr{K}_4) &\rightarrow \mathcal{H}_{\mathscr{K}_4} \end{aligned} \tag{15}$$

From Equation (13) and (15), we conclude that no two users can produce the same hash values if the users' passwords are different, as shown in Equation (16), even if the user ID and the domain word are the same.

$$\begin{aligned} \mathcal{H}_{\mathscr{U}_3} &\neq \mathcal{H}_{\mathscr{U}_4} \\ \mathcal{H}_{\mathscr{P}_3} &\neq \mathcal{H}_{\mathscr{P}_4} \\ \mathcal{H}_{\mathscr{K}_3} &\neq \mathcal{H}_{\mathscr{K}_4} \end{aligned} \tag{16}$$

Equation (11) and (16) ensure that two different users cannot produce the same hash values.

2.9 Prevention of the domino effect

The domino effect is a crucial effect to be prevented in password-based authentication systems. The state-of-the-art password-based authentication system cannot prevent the domino effect. PassPro deals with hash values with a shuffled user ID, password, and domain word. Therefore, it is computationally infeasible to reconstruct the original string from the hash values $\mathcal{H}_{\mathscr{U}}$, $\mathcal{H}_{\mathscr{U}}$, and $\mathcal{H}_{\mathscr{K}}$ if the adversary does not know the input strings except the domain word. This method ensures the prevention of the domino effect because different domain word creates different shuffled hash values. Thus, it permits the reuse of user IDs and passwords in multiple domains. Let us assume a user $\mathbb{A}$ uses the $\mathcal{U}$ and $\mathcal{P}$ at two domains, namely, the domain words are $\mathcal{D}_1$ and $\mathcal{D}_2$ where $\mathcal{D}_1 \neq \mathcal{D}_2$. The shuffled strings for the domain $\mathcal{D}_1$ for user $\mathbb{A}$ are shown in Equation (17).

$$\begin{aligned} \mathcal{U} \cup \mathcal{D}_1 &\xrightarrow{\mathcal{P}} \mathscr{U}_5 \\ \mathcal{P} \cup \mathcal{D}_1 &\xrightarrow{\mathcal{U}} \mathscr{P}_5 \\ \mathcal{U} \cup \mathcal{P} \cup \mathcal{D}_1 &\rightarrow \mathscr{K}_5 \end{aligned} \tag{17}$$

The shuffled strings for the domain $\mathcal{D}_2$ for user $\mathbb{A}$ are derived in Equation (18).

$$\begin{aligned} \mathcal{U} \cup \mathcal{D}_2 &\xrightarrow{\mathcal{P}} \mathscr{U}_6 \\ \mathcal{P} \cup \mathcal{D}_2 &\xrightarrow{\mathcal{U}} \mathscr{P}_6 \\ \mathcal{U} \cup \mathcal{P} \cup \mathcal{D}_2 &\rightarrow \mathscr{K}_6 \end{aligned} \tag{18}$$

We know that $\mathcal{D}_1 \neq \mathcal{D}_2$ and non-empty; therefore, it produces different shuffled strings. Hence, Equation (19) holds.

$$\begin{aligned} \mathscr{U}_5 &\neq \mathscr{U}_6 \\ \mathscr{P}_5 &\neq \mathscr{P}_6 \\ \mathscr{K}_5 &\neq \mathscr{K}_6 \end{aligned} \tag{19}$$

The hash values for the shuffled string with domain $\mathcal{D}_1$ for user $\mathbb{A}$ are given in Equation (20).

$$\begin{aligned} \mathcal{H}(\mathscr{U}_5) &\rightarrow \mathcal{H}_{\mathscr{U}_5} \\ \mathcal{H}(\mathscr{P}_5) &\rightarrow \mathcal{H}_{\mathscr{P}_5} \\ \mathcal{H}(\mathscr{K}_5) &\rightarrow \mathcal{H}_{\mathscr{K}_5} \end{aligned} \tag{20}$$

Similar to Equation (20), the hash values for the domain $\mathcal{D}_2$ for the same user are demonstrated in Equation (21).

$$\begin{aligned}\mathcal{H}(\mathscr{U}_6) &\rightarrow \mathcal{H}_{\mathscr{U}_6}\\ \mathcal{H}(\mathscr{P}_6) &\rightarrow \mathcal{H}_{\mathscr{P}_6}\\ \mathcal{H}(\mathscr{K}_6) &\rightarrow \mathcal{H}_{\mathscr{K}_6}\end{aligned} \tag{21}$$

By referring to Equation (19), we can conclude that the hash values cannot be the same in Equation (20) and (21). Hence, Equation (22) shows the inequality.

$$\begin{aligned}\mathcal{H}_{\mathscr{U}_5} &\neq \mathcal{H}_{\mathscr{U}_6}\\ \mathcal{H}_{\mathscr{P}_5} &\neq \mathcal{H}_{\mathscr{P}_6}\\ \mathcal{H}_{\mathscr{K}_5} &\neq \mathcal{H}_{\mathscr{K}_6}\end{aligned} \tag{22}$$

Equation (22) shows that the hash values cannot be the same for the same user ID and password for different domains. Thus, there is no domino effect. Moreover, the identity manager and adversary do not know the original input strings of the user IDs and passwords.

2.10 Transmission of the user's hash values

The user transmits the encrypted hash values using a shared secret key $\mathcal{SK}$ as $Enc(Enc(\mathcal{H}_{\mathscr{U}}, \mathcal{PR}_{\mathcal{C}}), \mathcal{SK})$, $Enc(Enc(\mathcal{H}_{\mathscr{P}}, \mathcal{PR}_{\mathcal{C}}), \mathcal{SK})$, and $Enc(Enc(\mathcal{H}_{\mathscr{K}}, \mathcal{PR}_{\mathcal{C}}), \mathcal{SK})$. The identity manager decrypts the encrypted hash values as follows-

$$\begin{aligned}\mathcal{H}_{\mathscr{U}}^{received} &= Dec(Dec(Enc(Enc(\mathcal{H}_{\mathscr{U}}, \mathcal{PR}_{\mathcal{C}}), \mathcal{SK}), \mathcal{SK}), \mathcal{PU}_{\mathcal{C}})\\ \mathcal{H}_{\mathscr{P}}^{received} &= Dec(Dec(Enc(Enc(\mathcal{H}_{\mathscr{P}}, \mathcal{PR}_{\mathcal{C}}), \mathcal{SK}), \mathcal{SK}), \mathcal{PU}_{\mathcal{C}})\\ \mathcal{H}_{\mathscr{K}}^{received} &= Dec(Dec(Enc(Enc(\mathcal{H}_{\mathscr{K}}, \mathcal{PR}_{\mathcal{C}}), \mathcal{SK}), \mathcal{SK}), \mathcal{PU}_{\mathcal{C}})\end{aligned} \tag{23}$$

where $\mathcal{PU}_{\mathcal{C}}$ is the public key of the user $\mathcal{C}$. The retrieved hash value $\mathcal{H}_{\mathscr{U}}^{received}$, and $\mathcal{H}_{\mathscr{P}}^{decrypted}$ are used to compare the existing hash values from the database of the identity manager. However, these hash values are stored in the database of the identity manager for future authentication purposes by encrypting using an individual secret key. The received $\mathcal{H}_{\mathscr{K}}^{received}$ is used to reconstruct an individual key for encryption or decryption.

2.11 Mutually reproducible secret key

Algorithm 4 is used to generate a mutually reproducible secret key, $\mathfrak{K}$, using a shuffled hash value $\mathcal{H}_{\mathscr{K}}$ and identity manager's master secret word $\mathcal{W}$. It is computationally infeasible to reproduce $\mathfrak{K}$ without these two inputs for a large bit-sized key. Alternatively, the client permits the identity manager to reproduce the individual secret key $\mathfrak{K}$ for encryption or decryption of the hash values of the user. Algorithm 4 is similar to VORSHA-3D-S [6], and the key difference is that Algorithm 4 produces bits based on a context, $\mathcal{W}$. The $\mathcal{W}$ influences the seed value generation process. Consequently, Algorithm 4 features memory hardness to defeat parallelism. Thus, it is difficult for the adversary to reproduce the mutually reproducible secret key to decrypt from the password database.

Algorithm 4 Algorithm for the mutually reproducible secret key generation for symmetric encryption or decryption.

procedure GENKEY($\mathcal{H}_{\mathscr{K}}, \mathcal{W}, \mathcal{S}, \eta$)
  $\mathscr{L}_{\mathcal{H}_{\mathscr{K}}} = \text{LENGTH}(\mathcal{H}_{\mathscr{K}})$
  $\mathscr{L}_{\mathcal{W}} = \text{LENGTH}(\mathcal{W})$
  $\tau = 16, \delta = 37, \mu = 16, \theta = 13297$
  $\mathcal{S} = \text{GETSEEDVALUE}(\mathcal{W}, \mathscr{L}_{\mathcal{W}}, \mathcal{S}, \tau)$
  $\mathcal{S} = \text{GETSEEDVALUE}(\mathcal{H}_{\mathscr{K}}, \mathscr{L}_{\mathcal{H}_{\mathscr{K}}}, \mathcal{S}, \tau)$
  $\tau = \mathcal{S}\%\delta + \mu$
  $r = \text{GENDIM}(\mathcal{S}, \theta)$
  $\mathcal{S} = \text{GETSEEDVALUE}(\mathcal{W}, \mathscr{L}_{\mathcal{W}}, \mathcal{S}, \tau)$
  $\mathcal{S} = \text{GETSEEDVALUE}(\mathcal{H}_{\mathscr{K}}, \mathscr{L}_{\mathcal{H}_{\mathscr{K}}}, \mathcal{S}, \tau)$
  $\tau = \mathcal{S}\%\delta + \mu$
  $c = \text{GENDIM}(\mathcal{S}, \theta)$
  $X = prime[r], Y = prime[c];$ ▷ $X \neq Y$
  for $i : 1$ *to* X **do**
    for $j : 1$ *to* Y **do**
      $\mathcal{S} = \text{GETSEEDVALUE}(\mathcal{W}, \mathscr{L}_{\mathcal{W}}, \mathcal{S}, \tau)$
      $h_v = \text{GETSEEDVALUE}(\mathcal{H}_{\mathscr{K}}, \mathscr{L}_{\mathcal{H}_{\mathscr{K}}}, \mathcal{S}, \tau)$
      $\rho = h_v\%31;$
      $bit = (h_v \wedge (1 << \rho)) >> \rho$
      $\mathcal{V}[i][j] = bit$
      $\mathcal{S} = h_v$
    end for
  end for
  for $k : 1$ *to* η **do**
    $\mathcal{S} = \text{GETSEEDVALUE}(\mathcal{W}, \mathscr{L}_{\mathcal{W}}, \mathcal{S}, \tau)$
    $h_v = \text{GETSEEDVALUE}(\mathcal{H}_{\mathscr{K}}, \mathscr{L}_{\mathcal{H}_{\mathscr{K}}}, \mathcal{S}, \tau)$
    $i = (h_v\%X) + 1$
    $\mathcal{S} = h_v$
    $\mathcal{S} = \text{GETSEEDVALUE}(\mathcal{W}, \mathscr{L}_{\mathcal{W}}, \mathcal{S}, \tau)$
    $h_v = \text{GETSEEDVALUE}(\mathcal{H}_{\mathscr{K}}, \mathscr{L}_{\mathcal{H}_{\mathscr{K}}}, \mathcal{S}, \tau)$
    $j = (h_v\%Y) + 1$
    $\mathcal{S} = h_v$
    $\mathfrak{K}[k] = \mathcal{V}[i][j]$
  end for
  return $\mathfrak{K}$
end procedure

Algorithm 4 calculates the dimension of the bit vector, $\mathcal{V}$, using a function, and the function is defined in [6], where the vector dimensions are X and Y. Initially, Algorithm 4 fills the vector $\mathcal{V}$ using pseudo-random bits. In the later phase, it retrieves the pseudo-random bits to form the key $\mathfrak{K}$.

The identity manager computes a mutually reproducible secret key using the following pseudo-random number gener-

ator.

$$\mathcal{G}(\mathcal{H}_{\mathscr{K}}) \xrightarrow{\mathcal{W}} \mathfrak{K} \tag{24}$$

where $\mathcal{W}$ is a master secret word from the identity manager and $\mathfrak{K}$ is a pseudo-random key. Equation (24) represents Algorithm 4 in the context of $\mathcal{W}$. The key $\mathfrak{K}$ can be reproduced using a valid $\mathcal{H}_{\mathscr{K}}$ and $\mathcal{W}$ consistently.

2.12 Encryption and decryption of the hash values in the password database

This process involves encryption and decryption using a symmetric key, termed as a mutually reproducible secret key, $\mathfrak{K}$. Users' hash values are encrypted and inserted into the database of the identity manager in the identity creation. The identity manager decrypts the encrypted hash values from the database of the identity manager for authentication.

2.12.1 Encryption

The identity manager encrypts the hash values of the user ID and password using an individual key. The identity manager requires a hash value to produce the key to encrypt. The identity manager stores $\mathcal{H}_{\mathscr{U}}$, $\mathcal{H}_{\mathscr{P}}$ by encrypting using an individual key, $\mathfrak{K}$. The $\mathfrak{K}$ is produced by Algorithm 4 using $\mathcal{H}_{\mathscr{K}}$ and $\mathcal{W}$. In identity creation, the $\mathcal{H}_{\mathscr{U}}$, and $\mathcal{H}_{\mathscr{P}}$ are encrypted by the identity manager using $\mathfrak{K}$ and inserted these encrypted hash values into the database. The hash values are encrypted as $Enc(\mathcal{H}_{\mathscr{U}}, \mathfrak{K})$, and $Enc(\mathcal{H}_{\mathscr{P}}, \mathfrak{K})$. For authentication, the encrypted hash values are decrypted using the same individual key for comparison.

Let us assume two different users, $\mathbb{A}$ and $\mathbb{B}$, encrypt their user IDs and passwords to store them in the password database. The user $\mathbb{A}$ sends $\mathcal{H}_{\mathscr{U}_1}$, $\mathcal{H}_{\mathscr{P}_1}$, and $\mathcal{H}_{\mathscr{K}_1}$ to the identity manager. Similarly, the user $\mathbb{B}$ also sends $\mathcal{H}_{\mathscr{U}_2}$, $\mathcal{H}_{\mathscr{P}_2}$, and $\mathcal{H}_{\mathscr{K}_2}$ to the identity manager. Therefore, the identity manager produces two mutually reproducible secret keys, $\mathfrak{K}_1$ and $\mathfrak{K}_2$, for the hash values of the user $\mathbb{A}$ and $\mathbb{B}$, respectively. Let us assume that the email ID for the user $\mathbb{A}$ and $\mathbb{B}$ are $\mathcal{E}_{\mathbb{A}}$ and $\mathcal{E}_{\mathbb{B}}$, respectively. Thus, the identity manager encrypts the hash values of $\mathbb{A}$ and inserts the encrypted hash values into the password database against the email ID $\mathcal{E}_{\mathbb{A}}$, as shown in Equation (25).

$$\mathcal{E}_{\mathbb{A}} \begin{cases} Enc(\mathcal{H}_{\mathscr{U}_1}, \mathfrak{K}_1) \\ Enc(\mathcal{H}_{\mathscr{P}_1}, \mathfrak{K}_1) \end{cases} \tag{25}$$

Similarly, the identity manager inserts the encrypted hash values of the user $\mathbb{B}$ and inserts the encrypted hash values into the password database against the email ID $\mathcal{E}_{\mathbb{B}}$, as shown in Equation (26).

$$\mathcal{E}_{\mathbb{B}} \begin{cases} Enc(\mathcal{H}_{\mathscr{U}_2}, \mathfrak{K}_2) \\ Enc(\mathcal{H}_{\mathscr{P}_2}, \mathfrak{K}_2) \end{cases} \tag{26}$$

It is not required to encrypt the user's email IDs because it is public. The email IDs are used to index the users' hash values in the password database.

2.12.2 Decryption

The identity manager receives $\mathcal{H}_{\mathscr{K}}$ to reproduce a key $\mathfrak{K}$ to decrypt the encrypted $\mathcal{H}_{\mathscr{U}}$ and $\mathcal{H}_{\mathscr{U}}$ from its database.

$$\begin{aligned} \mathcal{H}_{\mathscr{U}}^{decrypted} &= Dec(Enc(\mathcal{H}_{\mathscr{U}}, \mathfrak{K}), \mathfrak{K}) \\ \mathcal{H}_{\mathscr{P}}^{decrypted} &= Dec(Enc(\mathcal{H}_{\mathscr{P}}, \mathfrak{K}), \mathfrak{K}) \end{aligned} \tag{27}$$

The decrypted hash values in Equation (27) are used to match the hash value sent from the user for authentication.

2.13 Authentication

The identity manager compares the decrypted hash values from the user and decrypted hash values from the password database. For authentication, the client sends the hash values to the identity manager using public key encryption followed by symmetric key encryption. The identity manager generates the mutually reproducible secret key using Algorithm 4 by its master secret word $\mathcal{W}$ and shuffled hash value $\mathcal{H}_{\mathscr{K}}$. Thus, the identity manager retrieves $\mathcal{H}_{\mathscr{U}}^{received}$, $\mathcal{H}_{\mathscr{P}}^{received}$, $\mathcal{H}_{\mathscr{U}}^{decrypted}$ and $\mathcal{H}_{\mathscr{P}}^{decrypted}$ for comparison as given in Equation (28).

$$\begin{aligned} \mathcal{H}_{\mathscr{U}}^{decrypted} &= \mathcal{H}_{\mathscr{U}}^{received} \\ \mathcal{H}_{\mathscr{P}}^{decrypted} &= \mathcal{H}_{\mathscr{P}}^{received} \end{aligned} \tag{28}$$

If Equation (28) holds, then the user is authenticated; otherwise, authentication fails.

2.14 Variable hashing

Due to the presence of a rainbow table attack, it demands a variable-sized secure hash algorithm, and we incorporate the client-side password hashing using VORSHA-3D-S [6]. It features memory hardness. Also, it produces a variable-sized and randomized secure hash value. A variable-sized hash function can be used to perform client-side password hashing. The variable-sized hash function produces an output in a range $[\mu, \lambda)$ where $\lambda = \mu + \delta$. Alternatively, a correct input string can have $(\lambda - \mu)$ correct hash values. For example, there are a total of 768 correct hash values for a given input string if $\lambda = 1024$ and $\mu = 256$. Therefore, it thwarts rainbow table attacks because no one can produce the correct hash value with a correct length except the legitimate users.

2.15 Client-side password metering

PassPro does not store or deal with the raw form of user IDs and passwords. Therefore, PassPro cannot offer AJAX service to verify the password strength. Also, PassPro cannot

suggests the password vulnerability as shown in Figure 1. Instead, PassPro offers a client-side password-strength metering system such that the password's strength can be suggested to the user.

Algorithm 5 User ID or password strength checking algorithm. For password, $\mathscr{L} \geq 10$ and for user ID, $\mathscr{L} \geq 8$.

procedure STRENGTHMETER($\omega, \mathscr{L}, flag$)
  if $flag = 1$ **then** ▷ Case for password.
    $\ell = 9$
  else ▷ Case for user ID.
    $\ell = 7$
  end if
  if $\mathscr{L} \leq \ell$ **then**
    Invalid length.
    Exit
  end if
  if ISEMAIL(ω) = *true* **then**
    An email ID cannot be the user ID or password.
    Exit
  end if
  for $i : 0$ *to* $\mathscr{L}$ **do**
    if $\omega[i] \leq 65$ and $\omega[i] \geq 90$ **then**
      $CC = CC + 1$ ▷ Capital letter counter.
    else if $\omega[i] \leq 97$ and $\omega[i] \geq 122$ **then**
      $SC = SC + 1$ ▷ Small letter counter.
    else if $\omega[i] \leq 48$ and $\omega[i] \geq 57$ **then**
      $DC = DC + 1$ ▷ Digit counter.
    else if $\omega[i] = 32$ **then**
      Invalid: It cannot contain white space.
      Exit
    else
      $SyC = SyC + 1$ ▷ Symbol counter.
    end if
  end for
  $min =$ MINIMUM(CC, SC, DC, SyC) ▷ Minimum among four variables.
  if $min \leq 1$ **then**
    Invalid
    Exit
  else if $min = 1$ **then**
    Weak
  else if $min = 2$ **then**
    Good
  else if $min = 3$ **then**
    Strong
  else
    Very strong
  end if
end procedure

Algorithm 5 imposes the rules of PassPro as described in subsection 2.3 and 2.4. Algorithm 5 checks the length of the user ID and password. The minimum length of the user ID is eight characters, and the minimum length of the password is ten characters. Also, it imposes restrictions on using an email ID as a user ID or password. Moreover, Algorithm 5 counts the capital letters, small letters, digits, and symbols used in the user ID and password. The *CC*, *SC*, *DC*, and *SyC* are the counters of capital letters, small letters, digits, and symbols present in the input string. The minimum value is used to decide the strength of the password or user ID. To qualify for user ID, the minimum value must be 1. Alternatively, the user ID must contain at least one small letter, one capital letter, one digit, and one symbol. Similarly, the minimum value for a password is also 1 to qualify as a password. Therefore, the password must contain at least a small letter, a capital letter, a digit, and a symbol.

3 Conclusion

In this paper, we presented password security and protection and presented a novel method called PassPro. We have demonstrated that PassPro addresses the issues of the domino effect, which allows users to reuse the same user ID and password in multiple domains. Moreover, PassPro guarantees that no one can retrieve the original user ID and password. Consequently, it prevents the publishing of passwords by adversaries. Furthermore, PassPro also guarantees that the hash values in an identity manager are invalid in other identity managers. We have also presented that PassPro encrypts the users' hash values with separate keys. The keys are produced or reproduced mutually by the user and the identity manager. Hence, we present a mutually reproducible secret key. Thus, it requires a mutually reproducible secret key to decrypt the hash values from the identity manager. Consequently, the adversary cannot decrypt the hash values even if the identity manager provides the password database to the adversary. The mutually reproducible secret key follows the property of memory hardness, and it is designed based on pseudo-random number generation. We have demonstrated the use of shuffling the user ID, password, and domain word, where the order of the shuffled string can be reproduced by legitimate users consistently. However, it is difficult to reproduce the order of the shuffled strings by adversaries. The shuffling process creates unique strings even if the same user ID and password are used in multiple domains. Thus, we have completely solved the issues of password-based authentication systems. However, a user needs to enter an email ID and two secrets, namely, user ID and password, in PassPro. Therefore, it creates inconvenience for the users. However, the password database stealing issue is a major concern for users. Therefore, it justifies the inconvenience caused.

References

[1] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 292–302, 2016.

[2] Weili Han, Ming Xu, Junjie Zhang, Chuanwang Wang, Kai Zhang, and X. Sean Wang. Transpcfg: Transferring the grammars from short passwords to guess long passwords effectively. *IEEE Transactions on Information Forensics and Security*, 16:451–465, 2021.

[3] Wenjian Luo, Yamin Hu, Hao Jiang, and Junteng Wang. Authentication by encrypted negative password. *IEEE Transactions on Information Forensics and Security*, 14(1):114–128, 2019.

[4] Bernard Meyer. COMB: over 3.2 Billion Email/Password Combinations Leaked | Cybernews. [Online], available at https://cybernews.com/news/largest-compilation-of-emails-and-passwords-leaked-free, July 2022.

[5] Edvardas Mikalauskas. RockYou2021: Largest Ever Password Compilation Leaked | Cybernews, July 2022. [Online], Available at https://cybernews.com/security/rockyou2021-alltime-largest-password-compilation-leaked.

[6] Ripon Patgiri, Laiphrakpam Dolendro Singh, and Dalton Meitei Thounaojam. Vorsha: A variable-sized, one-way and randomized secure hash algorithm. Cryptology ePrint Archive, Paper 2023/110, 2023. https://eprint.iacr.org/2023/110.

[7] Hung-Min Sun, Yao-Hsin Chen, and Yue-Hsun Lin. opass: A user authentication protocol resistant to password stealing and password reuse attacks. *IEEE Transactions on Information Forensics and Security*, 7(2):651–663, 2012.