

## Article

# Bingo: A Semi-Centralized Password Storage System

Abdullah F. Al-Aboosi <sup>1</sup>, Matan Broner <sup>2</sup> and Fadhil Y. Al-Aboosi <sup>3,\*</sup><sup>1</sup> Department of Multidisciplinary Engineering, Texas A&M University, College Station, TX 77843, USA; abdullah.alaboosi@tamu.edu<sup>2</sup> Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843, USA; mbronner@tamu.edu<sup>3</sup> RAPID-AICHe, New York, NY 10005, USA

\* Correspondence: fadha@aiche.org

**Abstract:** A lack of security best practices in modern password storage has led to a dramatic rise in the number of online data breaches, resulting in financial damages and lowered trust in online service providers. This work aims to explore the question of how leveraging decentralized storage paired with a centralized point of authentication may combat such attacks. A solution, “Bingo”, is presented, which implements browser side clients which store password shares for a centralized proxy server. Bingo is a fully formed system which allows for modern browsers to store and retrieve a dynamic number of anonymized password shares, which are used when authenticating users. Thus, Bingo is the first solution to prove that distributed password storage functions in the context of the modern web. Furthermore, Bingo is evaluated in both simulation and cloud in order to show that it achieves high rates of system liveness despite its dependence on its users being active at given intervals. In addition, a novel simulator is presented which allows future researchers to mock scheduled behavior of online users. This work concludes that with the rise in online activity, decentralization may play a role in increasing data security.

**Keywords:** distributed authentication; passwords; peer to peer; Bingo



**Citation:** Al-Aboosi, A.F.; Broner, M.; Al-Aboosi, F.Y. Bingo: A Semi-Centralized Password Storage System. *J. Cybersecur. Priv.* **2022**, *2*, 444–465. <https://doi.org/10.3390/jcp2030023>

Academic Editor: Carlo Blundo

Received: 16 May 2022

Accepted: 20 June 2022

Published: 21 June 2022

**Publisher’s Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Users of online services define secrets such as passwords and place their trust in these services to keep their secrets secure from attackers. Passwords, along with some user identifiers such as an email address, are used to authenticate a user’s identity when logging into a service [1]. Recent studies show that the number of accounts that a single user may use on a weekly basis has grown substantially over the past decade [2]. A given user may be a member of close to a dozen social media platforms, have a number of password-protected bank accounts, maintain profiles for work related tasks, and so forth. As such, a daunting level of trust is placed on online services [3]. With that said, countless password breaches in the past two decades have caused millions of stolen identities and billions of dollars in damages to both individual users and online companies [4]. Malicious groups have become more adept at exploiting common mistakes in password storage databases and web protocols, requiring companies to quickly apply security patches which often only expose further vulnerabilities in their services [5–7].

Passwords may be stored in plain text, hashed with or without a salt, or reversibly encrypted [8]. Generally, a password is stored alongside other user identifying information in a database such as PostgreSQL or MongoDB [9]. The state-of-the-art password hashing algorithms are Bcrypt, Scrypt, and Argon2. Modern best security practices state that while hashing a password, a number of salt rounds should be included to add increased entropy [10]. Salting a password prevents common attacks such as pre-computed “rainbow tables” for offline attacks [11,12]. In an ideal scenario, an attacker should require a practically infinite amount of time (i.e., decades) to crack a hashed password [13]. Unfortunately,

a staggering number of online services still use outdated algorithms for hashing passwords or even go as far as storing passwords in plain text. If an attacker is able to recover a password either by cracking its hash or simply reading it in plain text, a user's account may become compromised, and the attacker is able to cause damage such as identity theft and attain private banking information.

Due to a rise in compromised passwords, Google unveiled the earliest commercial version of 2FA in 2011 [14]. Since then, nearly all secure online services utilize some form of "out of band" verification of a user's identity. An out of band verification requires that the user prove their identity through another means outside of the current session they are a part of. This includes but is not limited to a one-time code emailed to the user or a ping notification on a smartphone application. In the common case of a ping notification, both the user's device and the 2FA provider's server use a shared secret to generate a one-time password (i.e., OTP) [15]. If both parties generate the same password, the authentication succeeds. By using 2FA, an online service makes stronger guarantees as to the safety of its users' information, since even a compromised password requires an attacker to compromise another authentication mechanism belonging to the user, which is oftentimes quite difficult [16].

Rather than tackling compromised passwords through another form of authentication, Password aggregators such as LastPass aim to combat passwords that are easy to guess, as well as the reuse of passwords [17]. An aggregator requires that a user choose a "master password", which is then used to access a collection of all of the user's passwords for all online services. An aggregator will oftentimes provide the ability to choose randomly generated passwords and later pre-fill them for a user upon login [18]. This provides the benefit that the user only has a single password to remember, as well as removing the requirement from the user to choose many unique hard to remember passwords. Today, most leading web browsers provide this service as a built in feature. Google Chrome, Firefox, and Safari all recognize when a user is registering for a new online service by parsing a web page's HTML contents and finding a password form field. The user is then prompted if they wish to have the browser choose a secure randomly generated password, which is then stored in the browser and pre-filled on demand. For these browser based aggregators, the master password is often the user's system password.

With the rise of fingerprint readers and face recognition sensors in smartphones and computers, the question has often been raised as to why passwords are used at all. Given that biometric identifiers such as a user's face are incredibly difficult to forge, many web services utilize modern API's such as the "Web Authentication API", which allow web developers to leverage biometric readers on devices to authenticate a user [19]. For example, an iPhone user visiting a website which has enabled the Web Authentication API will be prompted to use their thumbprint reader or face recognition sensor to authenticate rather than enter a password. All leading browsers have support for this functionality, although the web development community has been slow to adopt it in development. In an attempt to prevent passwords from ever being stored long enough to allow for a meaningful attack on a database, authors [20] utilize the password reset functionality on most websites in their solution, "Øpass". Their solution targets timing attacks in 2FA push notifications by resetting a user's password to a random value each time the user logs in. The authors of [21] present a solution where a collection of authenticating entities are used to prove a user's identity. These may include a number of devices belonging to the user as well as a number of trusted third parties, such as friends or family.

This work aims to tackle what can be seen as the core of all web based password breaches: a central insecure entity of storage. With companies wary to expose their back-end codebases to scrutinization by the public, and codebases often becoming unmanageable just a few years after being written, it comes as no surprise that outdated databases and lack of best security practices leave countless passwords open for the taking by even an amateur attacker. Two-factor authentication (i.e., 2FA) has become the de-facto standard of password security in the past decade. A trusted company handles authenticating a

given user by requiring an “out of band” communication step to prove identity, such as responding to a ping notification on a smartphone. 2FA aims to prevent stolen passwords by introducing a secure indirection layer of authentication which is difficult for an attacker to compromise. We define this process of authenticating a user on behalf of a third party service as middle manning. This paper aims to show that while 2FA and other adopted solutions are an improvement on a standard password authentication scheme, they still maintain a number of security flaws in addition to not directly addressing the problems with password storage.

The rest of this paper is structured as follows: The imitations of existing solutions are discussed in Section 2. Section 3 contains the threat model. Section 4 contains the key challenges. Section 5 contains the system architecture. Section 6 contains demonstrating the Bingo workflow. Section 7 contains result and discussion. Section 8 contains the conclusion.

## 2. Limitations of Existing Solutions

### 2.1. Two-Factor Authentication

Despite its wide adoption, 2FA presents challenges in both usability and security. Firstly, requiring a user to perform an out of band authentication increases login time and serves as an impractical solution to the overall problem of authentication. Most companies wish for their users to have the easiest interaction with their service as possible. Shifting the load of security onto users by requiring an intricate password on top of diverting them away from their session to prove identity avoids the problem at its core: online services are not storing passwords responsibly. If online services were to safeguard users’ passwords from the start, there would be no reason for a user to use 2FA. Furthermore, 2FA providers must still store a long-term secret for each user to be used during each authentication cycle in order to generate a one-time password. As such, a single source of potential insecurity is merely shifted to the 2FA provider. An adept attacker may compromise the long-term secret for a user by attacking a 2FA provider’s database system, leading back to the original problem of the need for secure storage of a user’s secrets.

### 2.2. Password Aggregators

Despite their ability to generate passwords with high entropy, password aggregators shift the single point of failure from the service provider’s end to the user. If a user forgets their master secret, a lengthy account recovery process is required. This makes aggregators such as LastPass impractical and, as with 2FA, puts a high assumption on the security of the service managing the stored password in a central manner.

### 2.3. Biometric Identifiers

While using a unique bodily feature rather than a password presents a series of benefits, biometric identification maintains a series of challenges. Firstly, most computers still do not have biometric readers. Given that most users still log into online services primarily through a computer, the benefits of biometric identification fall through and the user must resort to either using a password or utilizing their smartphone with a biometric reader to authenticate, posing the same impracticalities as 2FA. Furthermore, during the past decade, a number of academic efforts have shown vulnerabilities in commonly used biometric readers such as Apple’s FaceID. These efforts pose the same questions as those regarding insecure password storage but re-framed in a new set of technologies. Lastly, adoption of biometric authentication has been slow on the web, with most online services still using passwords for authentication. Given that passwords are likely to be used for at least another decade, the question of how to securely store them is as relevant as ever.

## 3. Threat Model

The following aims to explain the most commonly used techniques by attackers in order to steal and crack users’ passwords [22]. The goal of this section is to enumerate the aspects of Bingo which makes it largely resistant to such attacks. We base our attacker

model on the assumption that an attacker  $\$A\$$  aims to target a single or multiple users  $\{U_i \dots U_n\}$  of an online service  $\$S\$$ . Furthermore, service  $\$S\$$  uses a traditional password storage process where each user's password is stored alongside their other profile data, such as an email address and name. Passwords may be stored in plain text, hashed, or hashed and salted.

### 3.1. Brute Force

Arguably the simplest method of attaining a password, brute-forcing involves repeatedly testing possible passwords for a user until the correct one is found [23]. Generally, brute force involves an online service that does not attempt to limit the rate at which an attacker can guess. In such a scenario, any user with a short, low-entropy password risk having their credentials guessed in a relatively short amount of time

### 3.2. Rainbow Tables

Assuming an attacker has successfully attained a set of passwords from an online service's database, the difficulty in using these passwords may vary [24]. Assuming the passwords are stored in plain text, no further work is required. If a password is hashed but not salted, a common attack technique is a "Rainbow Table", a series of precomputed hash values that allow the hacker to crack a given hash in a reasonable amount of time. As mentioned in the previous section, salting does indeed thwart most Rainbow Table attacks, but contrary to most users' beliefs about their security on the web, many online services still resort to plaintext password storage or unsalted hashing.

### 3.3. Man in the Middle

In order to obtain a user's password in plain text, an attacker may act as a "man in the middle" by intercepting the connection between the user's browser and the end server. While best safety practices indicate that all traffic between clients and servers should be secured by encryption such as HTTPS, a large number of online sites still use unencrypted traffic over HTTP [25,26]. If an attacker is able to intercept requests by the user in plain text, there is no need to brute force the password.

Furthermore, even if HTTPS is used, an attacker can utilize a "phishing attack" where the user is deceived into believing that communication with the server is secure. This may be achieved by forging a certificate for a specific domain and proxying traffic or simply by setting up a decoy website which looks exactly like the legitimate website. Both methods and others cause the user to input credentials in an unsafe communication session, allowing the attacker to intercept all inputted credentials. Bingo, in its current form, targets the first concern by ensuring all traffic between users and its proxy server is encrypted with {TLS}. Future work in this area aims to tackle phishing attacks as well.

## 4. Key Challenges

We return to the enabling factor in most attacks aimed at retrieving users' passwords: a weak centralized storage entity. Countless inexperienced developers are tasked daily with implementing scalable web services for the most profitable online services around the globe. Without an understanding of security best practices, these developers are likely to leave threat loopholes such as logging plain text passwords to development logs and forgetting to use salt rounds when hashing a password. There is simply too much variability in designing a secure user profile system, which is the reason why most companies that value security turn to a 2FA service to add an indirection layer to their authentication scheme.

- As such, Bingo aims to leverage the best qualities of all of the aforementioned solutions for insecure password storage. We define a series of key design challenges for Bingo to ensure that it provides seamless usability while also employing best security practices:
- Bingo should never store a user's password in a centralized manner. This means that the original plain text password should never be held in persistent storage, nor should

Bingo independently store enough shares of a user's password such that an attacker can recover all shares without attacking peers in the network.

- Bingo should never expose the user's password to the online service for which it is middle manning. The online service is tasked with maintaining a profile for the user, as is the norm today. With that said, Bingo takes full ownership of a user's password and acts as the single source of truth for confirming a user's identity, thus offloading this responsibility from the online service.
- A retrieved password from a collection of peers in the network should be verifiable by Bingo, meaning that an attacker cannot inject fake information into the system.
- The user profiles stored in Bingo's database should not provide a meaningful advantage to an attacker seeking to crack a user's password.
- Bingo should provide strong security guarantees but not be an overly opinionated system. Online services which register with Bingo should be granted the ability to prioritize availability or security.
- Bingo should maintain the highest availability possible given a set of configurations, meaning that there should be as few instances as possible where a user cannot register or log in to an online service.

We summarize these overarching challenges as a series of basic principles: decentralization, anonymity, verifiability, configurability, and availability. The following section describes the system architecture for Bingo which allows it to uphold these principles.

## 5. System Architecture

Bingo's system design stems from a simple core idea: a password should be distributed safely in a series of shares across peers in a network, with a central entity dictating all communication. As such, we define Bingo as being "semi-centralized" in that it leverages the benefits of secure distributed storage while still benefiting from a single source of truth for all transactions in the network. Figure 1 shows an overview of Bingo Proxy.

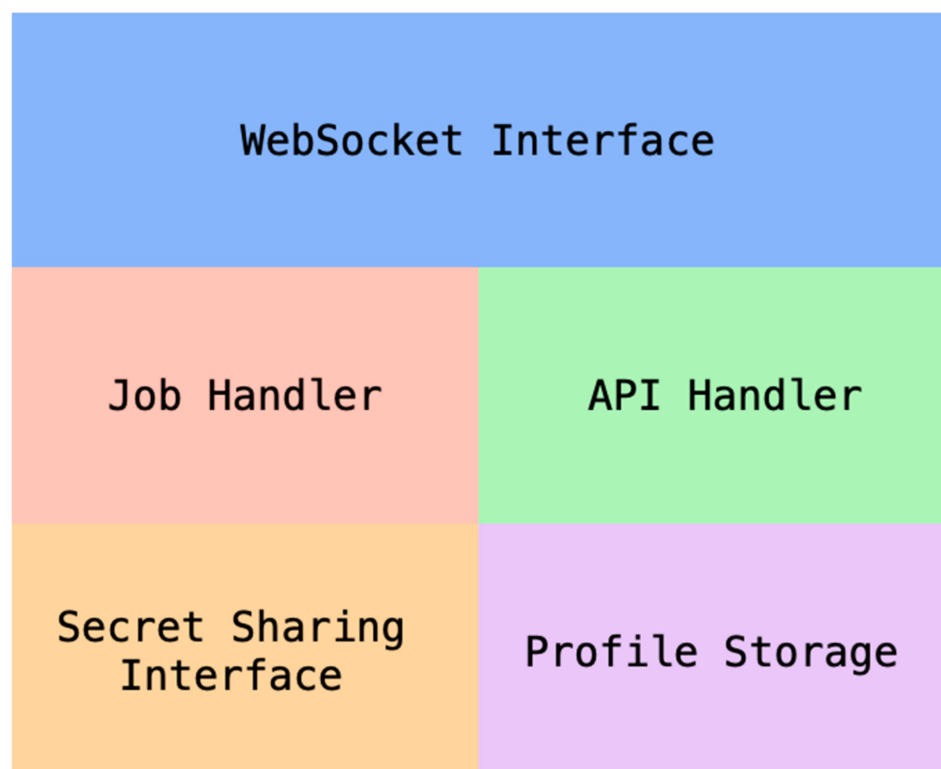


Figure 1. Bingo Proxy.

### 5.1. Cryptography

Bingo utilizes a number of state-of-the-art cryptographic functions to ensure that passwords are safely stored and that users cannot be tracked by examining individual password shares.

- (1) **Murmurhash:** Murmurhash (and its variant Murmurhash3) is a non-cryptographic hash function which can be seeded with a unique value to prevent offline rainbow table attacks so long as the seed remains a secret. We use a seeded Murmurhash3 implementation in Bingo to securely hash identifiers for data objects such that an attacker is unable to easily crack the hash. Furthermore, using Murmurhash3 provides the benefit of a consistent output for a given input (i.e., the basic property of hash functions), which aids in the repeated lookups of a plain text identifier. We define the Murmurhash3 function used in Bingo as M.
- (2) **Bcrypt:** Bcrypt is a secure password-hashing function with no proven attacks to date. The algorithm uses a defined number of salt rounds to increase entropy and prevent rainbow table attacks. A large number of websites which follow best security practices for hashing users' passwords prior to storage utilize Bcrypt as their hash function. We define the Bcrypt function used in Bingo as B.

### 5.2. Data Objects

- (1) **Domains:** When an online service wishes to use Bingo for authentication of its users, the service is required to register itself with Bingo through a series of configuration details. We refer to a service registered with Bingo as a "Domain". Domains are assigned a unique ID which is the Domain's URL domain hashed with M. A unique hash ID prevents an attacker from being able to discern the site origin of a secret share on a stolen device. Figure 2a displays the attributes of a Domain object.
- (2) **Users:** Bingo associates users with the Domains for which they register and login. When a new user wishes to register with a given Domain, Bingo generates a "User" object containing a unique key, which is a tuple of the Domain's unique ID and a hashed identifier for the user (e.g., an email) within the Domain. Figure 2b displays the attributes of a user object.
- (3) **Secret Shares:** A Secret Share is a fractional piece of a larger piece of secret information, such as a hashed password. Bingo distributes these shares among peers in the network alongside a tuple identifier consisting of a Domain ID and a user ID. Peers are tasked with storing these shares in a manner that allows them to later retrieve them given the same tuple identifier. Figure 2c displays the attributes of the Secret Share object.

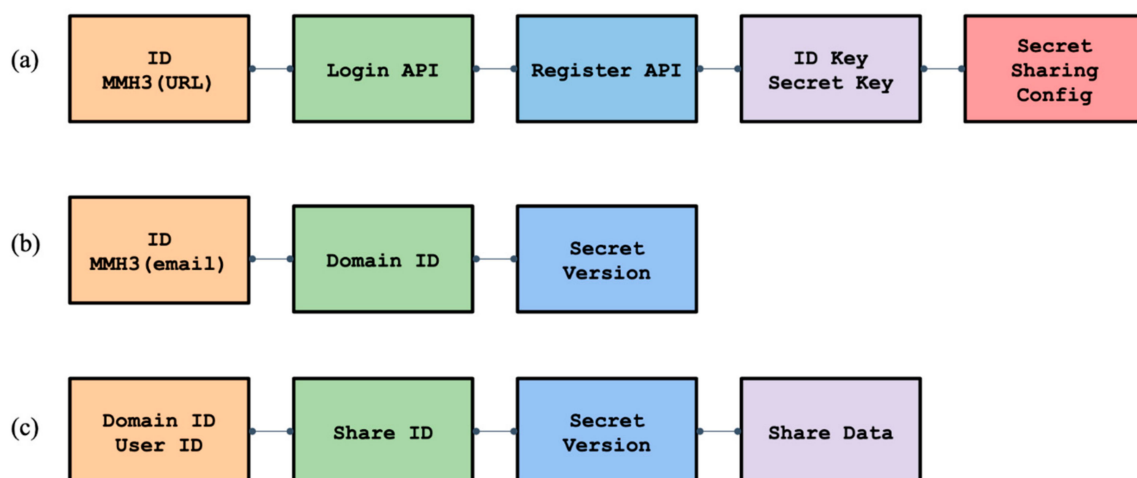


Figure 2. Bingo's data model utilizes a set of three basic objects.



### 5.3. Peers

A user owned device performs the role of a Bingo “Peer” by communicating with a central Bingo proxy server. We liken the interactions of Bingo’s peers to those in a classic peer-to-peer network in the fact that they contribute towards a collaborative task. With that said, Bingo’s peers do not maintain direct contact with one another, rather they rely on a centralized proxy server to broker all communication, resulting in a largely different system than a classic peer-to-peer network.

### 5.4. Bingo Proxy

The core logic for Bingo resides in its proxy server, named accordingly as “Bingo Proxy”. Bingo Proxy acts as the single source of truth for validation of a user’s identity. Unlike a 2FA providers’ authentication servers, Bingo Proxy requires zero out of band interaction by the user, thus simplifying the authentication workflow back down to its bare bones. This allows Bingo to provide a seamless user experience while maintaining strong security guarantees. The following is an explanation of the various components that make up Bingo Proxy.

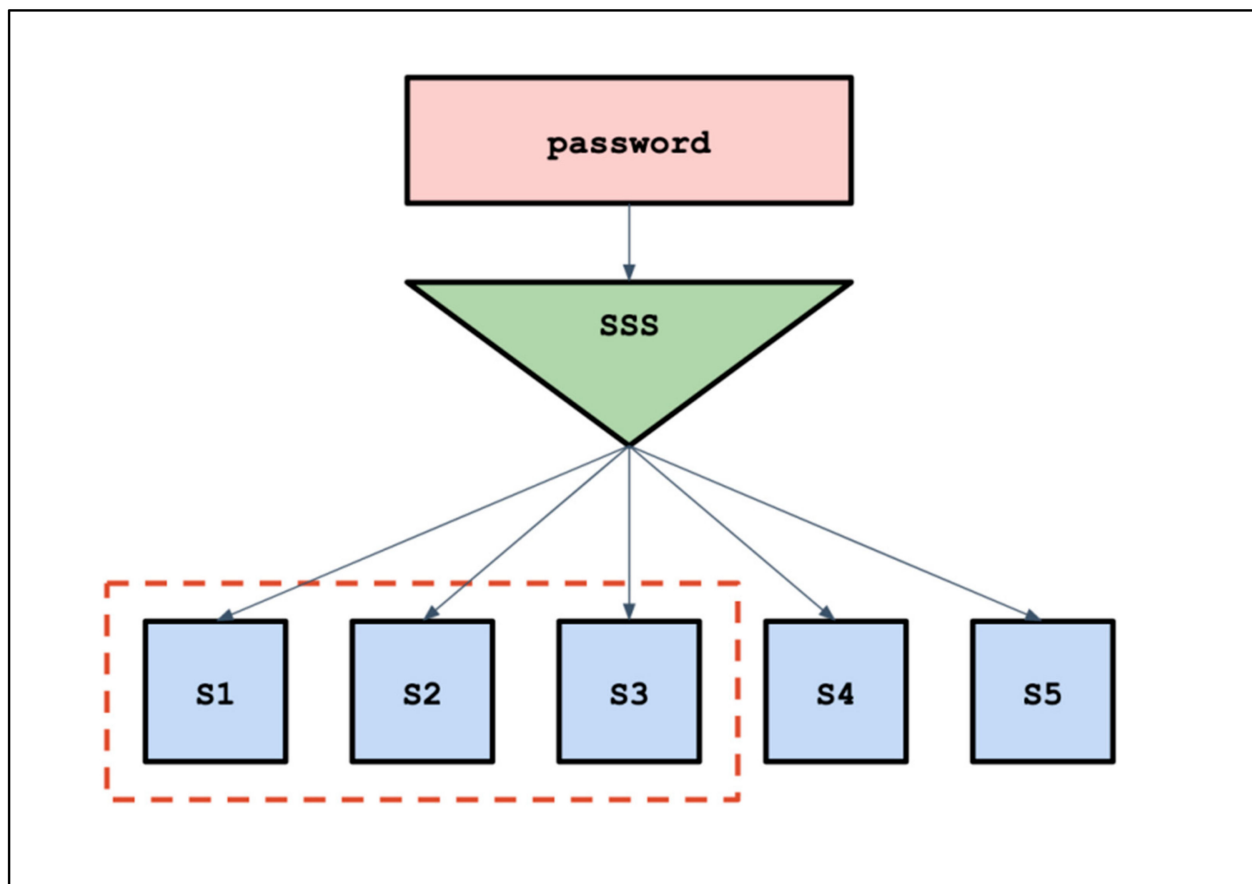
- (1) **Communication:** Bingo Proxy maintains all communications with peers through a dedicated Secure WebSocket Server (i.e., SWSS). Secure WebSockets communicate over TLS much like an HTTPS connection between clients and a given web browser. Secure communication over TLS guarantees that a man-in-the-middle attempting to eavesdrop on the authentication workflow is unable to read more than encrypted traffic. When a peer becomes live in the network, Bingo adds said peer to a collection of “active” peers which may be tasked to store and retrieve password shares. Likewise, when a peer disconnects from the network, its state is set to inactive and is promptly removed from the collection of peers. Peers are never associated with a specific user at the server-side. As such, if an attacker is able to attain a list of active peers in the network, this list does not allow any backtracking to a specific device belonging to a given user, ensuring the desired principle of anonymity. Bingo Proxy requires that all peers listen to two main events, “distribute” and “retrieve”. Peers should acknowledge these events and include retrieved data as required. Furthermore, peers can inform Bingo Proxy of new registration or login workflows with the “action” event, to which Bingo Proxy responds with an “action-update” event once the workflow has been completed (i.e., a callback). Further details regarding action workflows and responses will be provided in upcoming sections. A full listing of the API events sent by Bingo Proxy can be seen in Table 1.

**Table 1.** BINGO API EVENTS.

Bingo Proxy Event	Description	Bingo Peer Response
ID	Assign a unique identifier to a connected Peer	None
retrieve	Retrieve any password shares stored for a tuple of a Domain ID and User ID	retrieved
distribute	Store a share associated with a tuple of a Domain ID and User ID	distributed
action-update	A callback function indicating an action (login or register) has succeeded or failed	None

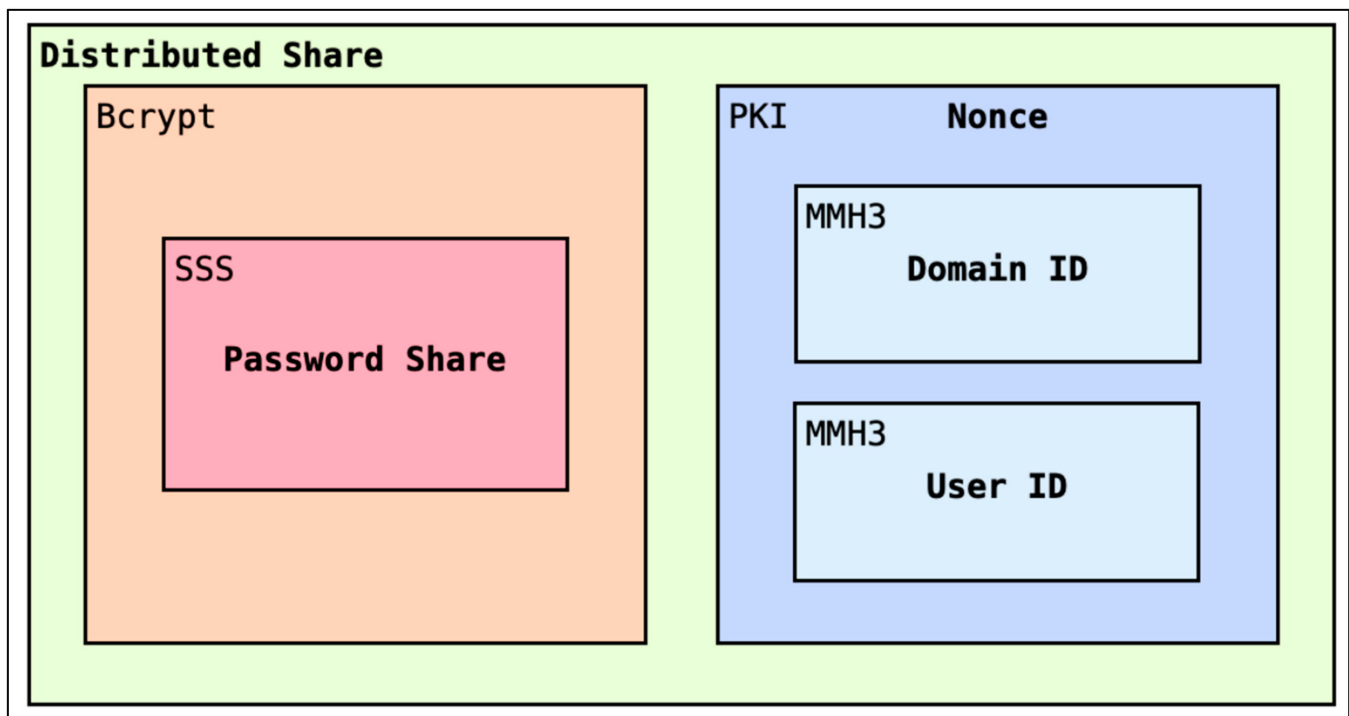
- (2) **API Handler:** Registered Domains are required to expose specific API methods to Bingo in order to facilitate registrations and logins. Domain objects should include a base API URL, a login route, and a registration route. For example, a Domain might provide a base API URL of <https://my-service.com/api> (accessed on 23 February 2022), a login route of `user/login` and a registration route of `user/register`. Furthermore, Domains must list which attribute in a user profile identifies the user, an ID Key, as well as which attribute serves as the password, a Secret Key. When registering a new user, Bingo makes an API call to a Domain's registration route alongside the original request body from the user with the value of the Secret Key redacted. If the API call returns a success status, Bingo stores a new profile for the user with the Domain's ID alongside the user's ID Key hashed with M. While the tuple identifier allows Bingo to verify that a user is registered with a domain, an attacker that manages to steal this information is not aided in verifying that the user is registered with the Domain, nor in the cracking of the user's password. Note that by only storing a profile for the user within Bingo if the API call is successful, the online service associated with the Domain is still given full authority to validate all remaining attributes of the request body, such as the email address or name. A similar workflow takes place during a login, except that naturally a different API route is used. Furthermore, the order of which secret sharing and retrieval takes place in regard to making API requests is dependent on the workflow. During registration, a secret is shared with the network after a successful API call to make sure that the secret should be shared at all. During login, the opposite ordering takes place, with the secret first being retrieved from the network in order to validate the user's identity before sending a login API request. Note that Bingo does not take any responsibility for managing third-party API tokens for a user. It is the responsibility of an online service to assign credentials at login time such that any subsequent requests which are not handled by Bingo can be traced back to the user.
- (3) **Secret Sharing:** Bingo Proxy's main functionality set is to distribute and retrieve shares of passwords. Bingo uses the tried and tested Shamir Secret Sharing algorithm, dubbed SSS, which divides a given password into a series of N shares such that the original password can be reconstructed given a set threshold T of these shares. For example, if  $N = 5$  and  $T = 3$ , then a share is distributed one per set of five peers, while only three peers must return their shares to be able to reconstruct the password. A threshold allows one to control the desire for availability versus security. If a threshold is set too low, an attacker must only retrieve a small number of shares to crack the password, lowering security. If the threshold is too high, we require a larger amount of active peers to be present in the network for registration and login procedures, lowering availability. Figure 3 displays Shamir Secret Sharing for the given values of N and T.





**Figure 3.** Shamir Secret Sharing for  $N = 5$  and  $T = 3$ .

To harden the security of Bingo's secret sharing, we aim to prevent the original password from ever being observable. As such, rather than apply SSS on the original plain text password, Bingo Proxy first generates a hashed version of the password with B. With the password hash, Bingo then applies SSS and collects  $N$  shares. Note that each registered Domain may set custom values for  $N$  and  $T$ , allowing a preservice trade off between availability and security, and upholding our desired principle of configurability. While using SSS aids in securely distributing a user's password across the network, we still require a mechanism to fulfill our desired design principle of verifiability, meaning that shares retrieved from peers should be able to be validated for their legitimacy by Bingo proxy. As such, Bingo Proxy employs Verifiable Secret Sharing through the usage of Public Key Infrastructure (i.e., PKI). Public-private key pairs are commonly used in client-server architectures to validate the identity of each party. Since the Bingo Proxy only needs to verify the shares it generates, a nonce is generated for each share, consisting of the Domain ID and User ID tuple. The nonce is then signed with Bingo Proxy's public key and appended to the share. When a share is retrieved and returned to Bingo proxy during a login workflow, an attempt is made to decrypt the nonce with Bingo Proxy's private Key. If the nonce has been tampered with, Bingo Proxy rejects the share, thus providing integrity protection for all exchanged shares and satisfying our requirement for verifiability. A visualization of the cryptographic methods employed for each share can be found in Figure 4.



**Figure 4.** Cryptography used for distributed secret shares.

In order to distribute shares during a registration workflow, Bingo Proxy chooses a set random number of active peers  $P$ , which is calculated by multiplying  $N$  by a configured replication factor  $R$ . Each domain configures  $R$  in order to further balance availability and security. Each share is distributed  $R$  times among the network, thus increasing the likelihood of the share being available for retrieval during a login workflow. As with setting the SSS threshold  $T$ , if  $R$  is set too high, then an attacker can more easily accumulate a user's password shares by attacking fewer devices. Note that Bingo Proxy does not distribute shares solely among other users associated with the same domain. Rather, any active peer in the entire Bingo network may store a share, regardless of which Domain the user behind said peer is associated with. This is done for two main reasons. First, a lack of selectivity in choosing peers for share distribution and retrieval generates higher availability, allowing all online services to essentially collaborate with one another through their shared usage of Bingo. Second, anonymizing peers in regard to which domain they are used to authenticate with simplifies the Bingo Proxy architecture as well as the user's workflow. Much like 2FA providers, Bingo users only require one tool to authenticate with all online services which use it for authentication. With that said, a private deployment of Bingo may allow selectivity in choosing peers for distribution and retrieval of shares, but this is beyond the scope of this work.

In Bingo Proxy's basic form, peers are chosen randomly for distribution. We leave the task of improving this distribution method for future work. We propose a distribution scheme which selects peers based on the amount of time they have been active. By choosing more active peers, Bingo Proxy makes a gamble that the user being managed by this peer is likely to be active for longer periods of time throughout the day, and thus, the user is more likely to be able to retrieve shares on demand. In any case, a peer is sent a WebSocket distribute event along with the share assigned to it. When the peer has successfully stored the share, it acknowledges the distribution with a distributed event back to Bingo Proxy. When  $N \times R$  peers have acknowledged the distribution, the registering user receives a success status. If not enough acknowledgements are received, the user fails to register. During a login workflow, Bingo proxy queries all active peers in the network for a share assigned to the Domain ID and User ID by broadcasting a retrieve event. Any active

peer storing a share which matches both identifiers respond with a retrieved event along with the corresponding share. When  $T$  unique shares are retrieved by the network, the aforementioned login API route for the Domain is called, and the user is successfully logged in along with the contents of the API call's response, which should include a third-party API key. The following elaborates on Bingo Proxy's job system, which aims to provide an elegant programming interface for registrations and logins which are asynchronous in their nature, and may require numerous attempts to succeed.

- (4) **Job Service:** In order to limit the negative effects on system availability caused by a fluctuating number of active peers at any given time, Bingo Proxy implements user workflows as "jobs" which run asynchronously and deliver results upon completion. The current implementation of Bingo Proxy provides two job interfaces: Distribution Job for registrations and Retrieval Job for logins. During a workflow, a corresponding job is initialized with all necessary information to be provided to peers, along with a callback function which is used once the job either succeeds or fails. A job is instantiated with three retry attempts, with exponential backoff timing used to space out retries. Each attempt is assigned a timer, with the timer being stopped if the job succeeds. Each job also maintains the state of its required data, and uses retries to attempt to fill gaps in its success requirements. Both distribute and retrieve events contain a unique Job ID which peers include in their responses in order to map acknowledgements to their corresponding job.

A Distribution Job stores all acknowledgements received from peers as a unique set. When a peer responds to Bingo Proxy with a distributed event, the Job is sourced by its ID and the peer's ID is stored if it was included in the distribution set. If an attacker attempts to acknowledge a distribute event it did not receive, the Distribution Job will ignore the attacker's response. If the Job does not contain  $N \times R$  acknowledgements when the timer goes off, an exponentially growing amount of time is waited before the next retry begins in order to allow for more peers to join the network. During the next retry, assuming that  $D$  peers have acknowledged the job,  $(N \times R) - D$  new peers are chosen based on whether they have already been sent a distribute message for the Job. As such, each attempt requires less peers than the last to acknowledge the event. Once all  $N \times R$  peers have acknowledged the event, the Job's callback function is called with a success message, indicating the workflow may continue.

A Retrieval Job behaves in a similar fashion to a Distribution Job in regard to using exponential backoff between retries. Unlike a Distribution Job, a Retrieval Job broadcasts a retrieve event to all peers in the network during each attempt in an effort to receive as many retrieved shares as possible. A Retrieval Job maintains a unique set of all retrieved shares. Once  $T$  unique shares have been saved to the Job, the assigned callback function returns a success message, indicating that the workflow may continue. Note that shares sent by peers which are already contained in the set will not count towards the completion of the Job, since SSS demands that  $T$  unique shares be collected in order to reconstruct the original secret.

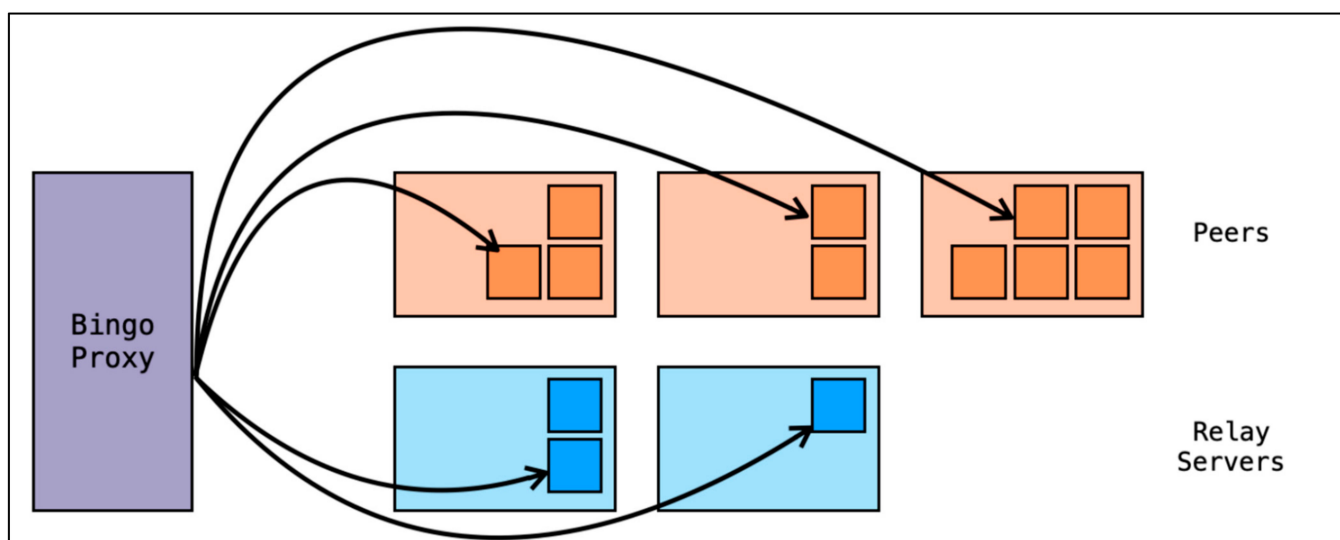
In the case of both Job types, if all attempts expire and a given Job fails to collect enough data to succeed, the Job's assigned callback function will return an error message, indicating that the workflow may not continue and that the user should make another attempt to either register or login.

- (5) **Password Versioning:** Bingo cannot be considered a practical medium of distributed secret storage if the shares stored for a given user cannot be replaced, such as during a password change. In order to achieve this goal, Bingo Proxy assigns a numerical version to each password share it distributes among peers. The most recent version count, denoted  $V$ , is stored alongside the hashed identifiers for a stored User in Bingo Proxy's database. As such, during the retrieval of a user's shares for reconstruction,  $V$  is included within the request body. Since all peers receive the distribution request in a broadcast, each peer may use the value of  $V$  to prune any stale data it may store

for the user, thus preventing shares of any version  $\{1 \dots V - 1\}$  from continuing to be stored or broadcasted. Furthermore, any peer which mistakenly or maliciously retrieves and returns a share older than  $V$  will be ignored, and the given share will not be counted towards the completion of a Retrieval Job. Note that the use of a primitive numerical counting system for share versioning is not without its flaws. Without mandatory integrity support, a malicious actor may tamper with the version of a share they store, as well as tamper with the version of an in-flight share either during distribution or retrieval. We leave the improvement of Bingo's share versioning system to future work, and propose the use of unique hashes or encrypted version numbers to improve integrity.

### 5.5. Bingo Relays

In order to facilitate higher availability for users registering or logging in, Bingo allows an online service to utilize Bingo Relays, a series of static peers which remain active at all times. These peers behave exactly the same as all other peers in the network, excluding the fact that they are deployed by Bingo Proxy and have nearly zero downtime. Each Bingo Relay may store at most one share per user, like all other peers. Furthermore, Bingo Proxy permits an online service to configure how many of the  $N$  shares associated with a user's password be sent to Bingo Relays, with the maximum allowed being  $N - 1$  in order to never allow a scenario where an attacker can compromise a user's password by solely attacking Bingo servers. Configuring this metric is once again a balancing act of availability and security. While storing as many of a user's password shares as possible on Bingo Relays generates high availability, it also shrinks the necessary attack surface in order to compromise a user's password. By default, a single share from each user's password is assigned to a Bingo Relay, with the chosen relay being the one with the least stored shares at the time of registration. Figure 5 provides an example of how Bingo Proxy may distribute shares to both peers and Bingo Relay server.



**Figure 5.** Bingo Proxy distributing shares to both peers and Bingo Relay servers.

### 5.6. Bingo Extension

Bingo Extension is a proof-of-concept browser extension framework that serves as a peer for communication with Bingo Proxy. Given its advanced functionality and well documented API, this work demonstrates a Bingo Extension on Google Chrome. We use the term "Bingo Chrome" to refer to the Bingo Extension created as a proof of concept for this work, while we reserve the term Bingo Extension to refer to the general framework used to create a peer for Bingo on a given device.

A Bingo Extension must have access to a persistent storage medium for the storage and retrieval of password shares delivered by Bingo Proxy. Shares must persist across sessions and as such may not be stored in a volatile storage such as an in memory database. In the case of Bingo Chrome, Indexed DB is used due to the simplistic queries that can be written to store and retrieve data for a created table, as well as the ubiquitous support for Indexed DB across modern browsers. Browsers do not support direct writes to a user's hard disk, requiring the usage of in-browser storage. While this mechanism is not ideal, due to a user's ability to wipe all stored shares by clearing their browser's cookies and site data, we leave the improvement of this storage scheme to future work.

A Bingo Extension must also be able to accept messages from the browser window which indicates the beginning of a registration or login workflow, as well as be able to forward these requests to Bingo Proxy. This inter-process messaging system is generally device and browser specific from a software development point of view. In the case of Bingo Chrome, a set of two scripts are used: background.js and content.js. These are both standard scripts included in any Google Chrome extension. Bingo Chrome's background script is run persistently at all times, keeping an active peer available at hand. While background scripts default in behavior to terminating when no longer in use, we found that the latency introduced by using a non-persistent background script and peer created a slower user experience. As such, we leave the task of initiating and terminating peers more efficiently to future work. The content script is activated once per web page, allowing the background script to send tab-specific API messages as needed. When a content script receives a Bingo action request from a web page, it forwards the request to the background script for processing, which in turn uses an active peer to relay the message to Bingo Proxy. Once the workflow has either completed successfully or failed, the background script returns the status of the workflow to the content script, which forwards the message back to the web page. The intra-extension communication scheme within Bingo Chrome can be observed in Figure 6, while the higher level communication between Bingo Chrome and Bingo Proxy through the previously discussed WebSocket API is shown in Figure 7. Note that while the Google Chrome runtime library is used in Bingo Chrome to facilitate tab-specific messaging, all modern browsers expose a variant of the standardized browser runtime object. This ensures that Bingo Chrome serves as a model for Bingo Extensions on most modern browsers such as Firefox and Edge.

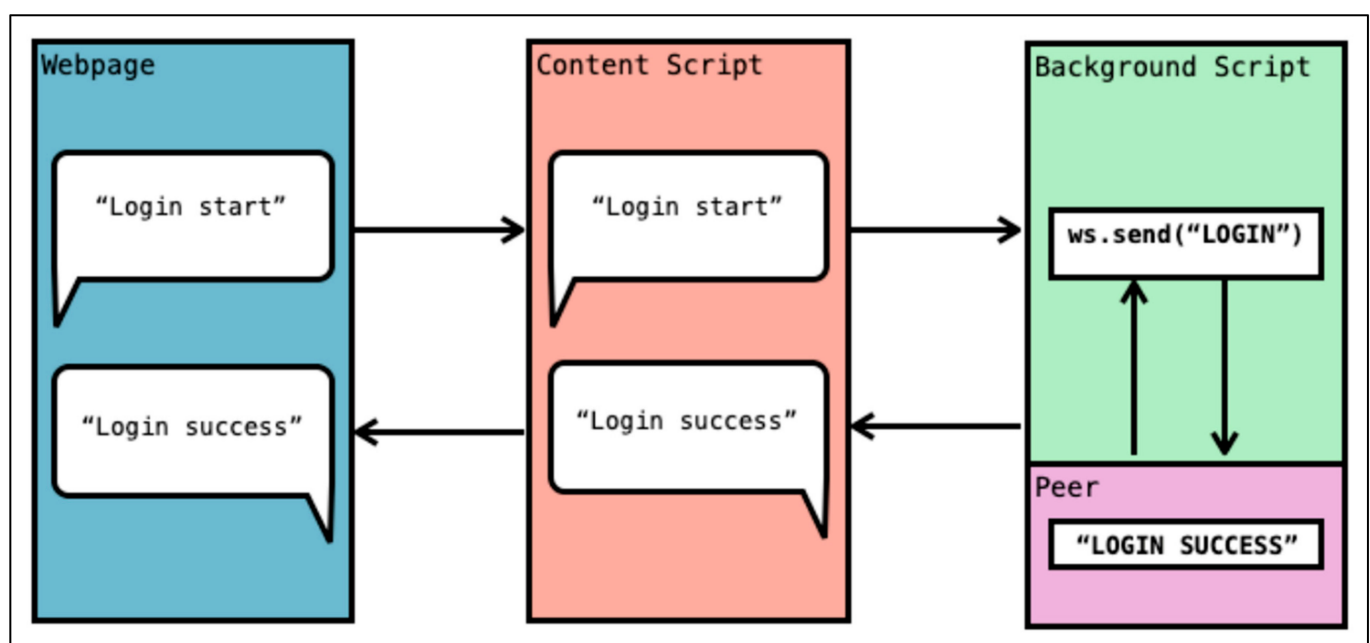
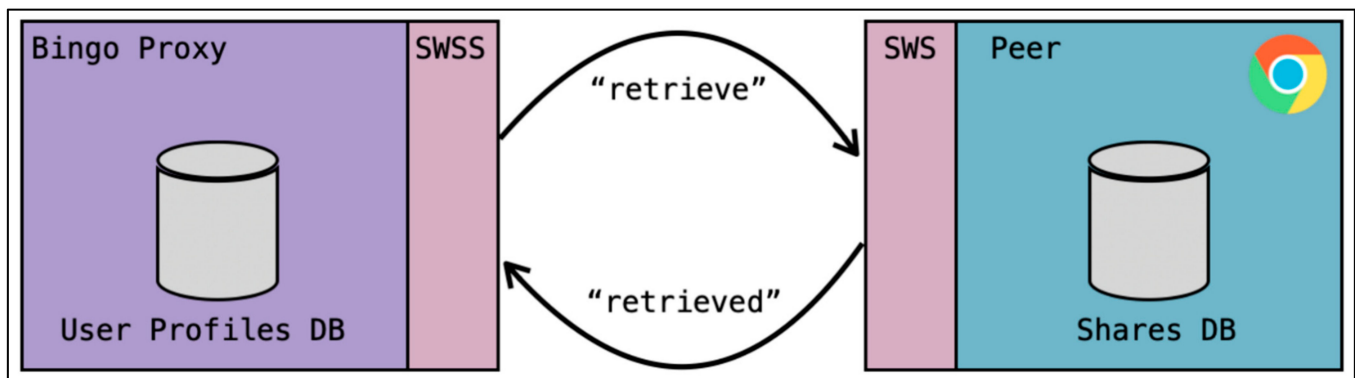


Figure 6. Intra-extension communication within Bingo Chrome.



**Figure 7.** Bingo Chrome communicating with Bingo Proxy.

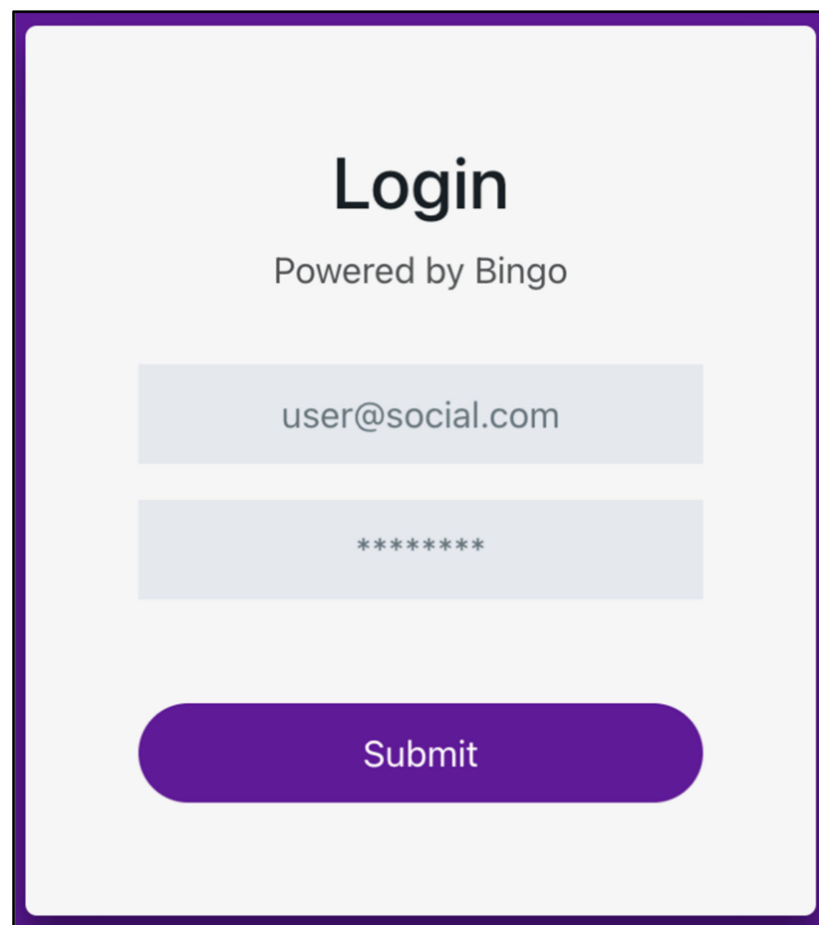
Lastly, while browser extensions were chosen as the testbed for this work due to desktop browsers being one of the most ubiquitous tools used for authentication, any WebSocket client which correctly accepts and responds to Bingo Proxy’s API events may serve as a Bingo Extension. Developers may reference Bingo’s documentation if they wish to develop a custom Bingo Extension for any device of their choosing that is able to make a WebSocket connection, such as smartphones and tablets.

### 5.7. Bingo UI

While Bingo Proxy handles the back-end logic of registering and logging in users, a front-end interface (i.e., UI) is required for triggering these workflows. Bingo UI is a framework for designing components that can be included in a web page to facilitate communication with a Bingo Extension. A Bingo UI component should serve the purpose of accepting credentials from a user and communicate them safely for processing. A web developer should be able to include a Bingo UI component on their web page and configure the component with attributes such as the address of a Bingo Proxy instance. Most 2FA providers use an iframe that displays a third party web page within the active window and redirects the user according to the success or failure of the operation taking place, such as logging in. Given that iframe tags are considered to be largely unsafe due to the possibility of cross-site scripting attacks, we demonstrate a proof of concept Bingo UI component built with ReactJS, a modern JavaScript component library. We dub our component as “Bingo React”, following the same naming convention as Bingo Chrome. Figure 8 displays Bingo React with its “login” flag set.

A web developer places Bingo React within their web page along with a flag indicating whether the component is in login or registration mode. Bingo React first verifies that a form of Bingo Extension is installed. In our case, Bingo Chrome injects a hidden HTML tag which Bingo React attempts to locate. If the tag is present, Bingo React knows that an extension is installed and running correctly. When a user attempts to register or log in, Bingo React sends an appropriate window message to be intercepted by the active extension. A Bingo UI component must be sure to indicate the loading status of a workflow until the extension returns a status for the sake of the user’s experience. Bingo React sets a visual loading icon while a response has not yet been received. Throughout the various system components of Bingo described in this work, Bingo UI is purposely least elaborated upon. Designing the ideal user interface for authentication is secondary to exposing a robust and simple API which web developers may use in their own programs. Countless companies have exposed robust public API’s for which the open source community has included in custom web components and posted to forums such as NodeJS’s NPM. We believe that as Bingo generates popularity and develops a more robust feature set, UI components will be developed naturally by those who need them.





**Login**

Powered by Bingo

user@social.com

\*\*\*\*\*

Submit

**Figure 8.** Bingo React in “login” mode.

## 6. Demonstrating the Bingo Workflow

With an understanding of the system components making up Bingo, we now aim to provide a demonstrative view of the interactions between an online service, its users, and Bingo. We demonstrate our user workflows in the context of a simple website which is called “Social Home”. This website incorporates Bingo React and runs within Google Chrome with Bingo Chrome installed. While the Evaluation portion of this work dives into the intricacies of the number of active peers in the Bingo network, we assume that 100 active persistent peers exist within the network at the time of registration and login by a user accessing Social Home. A simple web server for Social Home is run as well and exposes two API routes: /login and /register. Furthermore, our running instance of Bingo Proxy is set to split passwords into five shares with a recovery threshold of three, a replication factor of two, and the number of shares being stored within Bingo Relay servers being one. Note that we assume a “perfect” system where all workflows complete either successfully or with an error. Any errors during the subsequently described workflows are propagated back to Bingo React where they are displayed on the web page.

### 6.1. Peer Registration

A user U accesses Social Home for the first time, having never registered with the service before. Given that Bingo Chrome is installed on the browser used by U, Bingo React shows no error message and displays a tabbed window allowing for registration and login. U accesses the registration tab and inputs a set of credentials. Once the registration is finalized, Bingo React emits a window message REGISTER, which is intercepted by Bingo Chrome. The active content script within Bingo Chrome assigned to Social Home accepts the registration credentials passed through the window message and forwards

them to the background script. With its active peer, the background script initiates an action of type “register” with the peer, and is provided a unique action ID. Bingo React transitions to the loading state, indicating that the registration workflow is in progress. Bingo Proxy receives the registration action request which includes the credentials from U as well as the action ID provided by the peer. Bingo Proxy extracts the email and password fields from the credentials, which Social Home has configured as its ID Key and Secret Key, respectively. After sending an API registration request to Social Home’s back-end server with U’s password withheld, Bingo Proxy awaits a successful status code, namely a 201 in the case of registration. Following the successful registration at the Social Home’s server, U’s email is hashed with M, and the withheld password is hashed with B and split into 5 shares, each of which is replicated once for a total of 10 shares. Note that each share is assigned a version attribute set to 1 in order to allow U to change their password in the future. Bingo Proxy creates a Distribution Job, instantiating it with the 10 generated shares for U. As described in a previous section, this Job makes at most three attempts to distribute the shares across the network. Since this demonstration uses a set of 100 constantly online peers, the Job succeeds with 10 received acknowledgements and the callback function assigned to the Job is called with a success message. Bingo proxy then randomly chooses 1 of the 10 shares it generated and delivers it to one of its Bingo Relay servers with the least stored shares. Finally, a user profile containing U’s hashed email and Social Home’s Domain ID is written into Bingo Proxy’s database for reference during subsequent logins. With the workflow completed, Bingo Proxy sends the initiating peer within Bingo Chrome an action-update event with a success message. Bingo Chrome forwards the message from the background script through to the content script, which then sends a window message REGISTER\_SUCCESS, triggering Bingo React to terminate the loading state and inform the user of the successful registration.

### 6.2. Peer Login

After U has successfully registered with Social Home, they proceed to log in. After shifting to the login tab of Bingo React, U inputs the credentials chosen at registration time and finalizes the login request. The communication between Bingo React and Bingo Chrome is largely the same as the previous workflow, other than the obvious messaging changes from registration indicators to login indicators. Bingo Proxy accepts the login request from Bingo Chrome’s peer and hashes the provided email field with M. A database query for a profile containing U’s hashed ID and Social Home’s Domain ID is initiated in order to verify that U is indeed registered with the website. The database returns a single entry, and Bingo proxy instantiates a Retrieval Job with the threshold T of 3 configured by Social Home. The Job makes three attempts to retrieve three unique shares for U by broadcasting to the network. Once enough shares have been returned, the job calls its assigned callback function with a success message and the retrieved shares. Bingo Proxy uses an inverse SSS function to reconstruct the original password hashed with B from the returned shares, and B is used to compare U’s inputted password and the reconstructed password. If they match, Social Home’s login API is called, which should return a credential such as an API key for U to use in further requests. The response with the credential is forwarded to the originating peer through an action-update message, which is propagated from Bingo Chrome to Bingo React. The API key assigned by Social Home is stored as a browser cookie to be used in subsequent requests, and U is redirected to the site’s main content as an authorized user. The following section describes our evaluation of Bingo using the aforementioned workflows as base cases for its expected behavior.

## 7. Result and Discussion

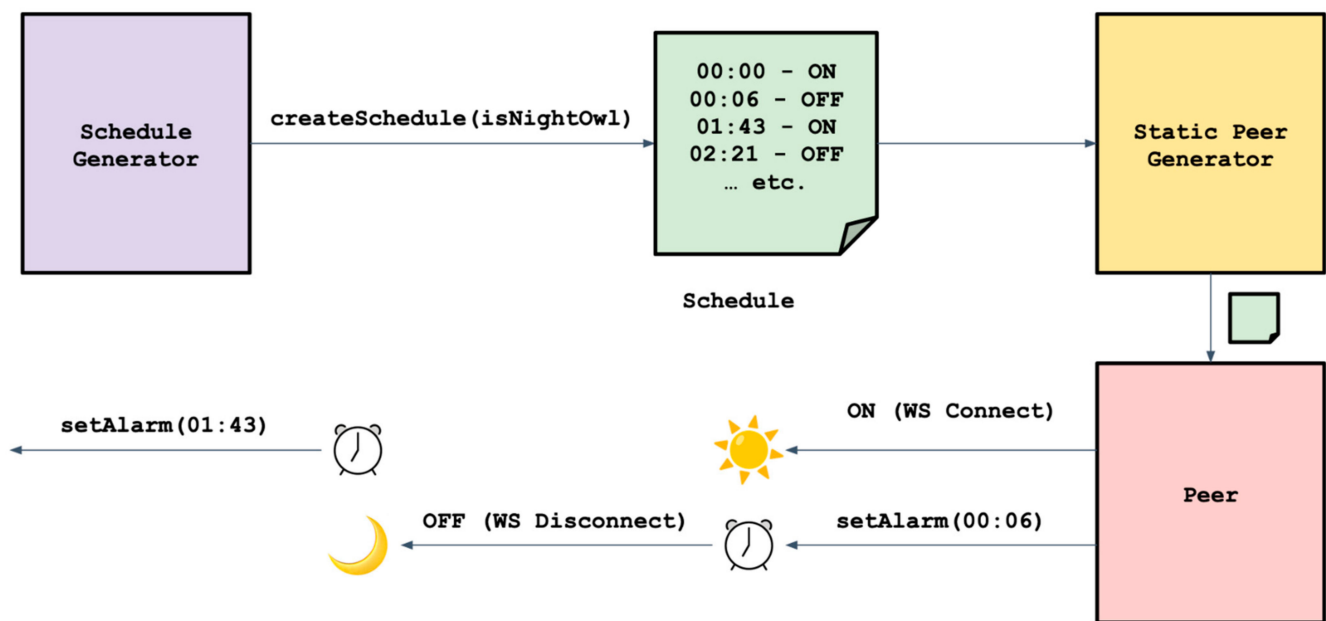
This section aims to explain how Bingo was evaluated within our test-bed. First, we provide a simple base evaluation proving that the previous section's workflows do in fact work correctly. Next, we describe our simulation of browser usage by users following daily schedules in order to better profile how well Bingo provides availability when peers are not always active.

### 7.1. Base Evaluation

With the Bingo system explained, our initial testing is a basic sanity test, which aims to show that in a perfectly live system, a user of Bingo can expect to successfully register and log in with a web service. Using Social Home as our web service, we have successfully tested the workflows mentioned in the previous section. With Bingo Chrome installed and an instance of Bingo Proxy deployed, our testing showed that a user U's credentials were successfully distributed among a group of test peers and said peers successfully retrieved U's secret shares on demand. Note that this sanity test is not a complete evaluation of Bingo's ability to perform under real-world conditions; rather, it serves to show the baseline we set as "perfect" behavior of Bingo which we strive to reach under the realistic conditions of fluctuating peer availability within the network. The following section details our simulation of real world conditions with the goal to prove Bingo's capabilities as a usable authentication system.

### 7.2. Peer Uptime Simulator

We now transition to our evaluation strategy of Bingo, specifically targeting the desired liveness expected from any authentication system. We define "liveness" as the metric that quantifies a user's ability to make use of a service without failure due to a lack of available resources. Since we are not able to deploy Bingo Extensions among enough real-world devices for the sake of an accurate evaluation of Bingo's liveness abilities, we resorted to designing a "Peer Uptime Simulator". The simulator is tasked with generating unique schedules for peers to activate and deactivate which mimic the way real people use their browsers. Most people tend to use their browsers primarily towards the middle of the day, when at work or school. Likewise, a minority of people tend to be active primarily throughout the night. We dub these people as "night owls". Since the average amount of time spent on browsers daily by most people is between 4 and 7 h, our schedule generator chooses a random number of total hours of active time within this range. Furthermore, each user has a 25% chance of being designated a night owl. Schedules are designed to concentrate most of their user's activity (i.e., being live) towards the middle of the day or night, depending on if the user is a night owl. For example, assuming the user is not a night owl, their activity will tend to be concentrated between 12 PM and 4 PM, much like a person working a day job. Our concentration of live activity tends to follow a bell curve with random outliers in activity to prevent all users from being active at the exact same time intervals. Lastly, in order to save time in evaluating Bingo, our simulator is able to translate each second of real-world time to T minutes in simulated time. For the purpose of our evaluations, we used  $T = 1$ , meaning each real-world second corresponded to one simulated minute. Figure 9 shows the workflow of generating schedules and assigning them to peers which are generated.



**Figure 9.** Peer Uptime Simulator generating schedules for peer.

Using our simulator, we generated 300 peers which are each assigned a schedule. Bingo Proxy uses the same metrics of  $N = 5$  and  $T = 3$  for secret sharing, and no relay servers a replication factor of one are used to prevent a heightened illusion of availability. Furthermore, we use a pre-registered user of Social Home in order to which attempts to log in every second (i.e., one simulated minute) for 1440 s (i.e., 24 simulated hours). The peers which store our user's password shares are randomly chosen for each iteration of our evaluation, and so we can conclude that about 30% of peers chosen to store these shares are night owls. Our user is using Google Chrome with Bingo Chrome installed on the same web page for Social Home as in our base evaluation, which used Bingo React. The Selenium browser control software is used to automate the process of logging in. Each time our user attempts to log in, the success or failure of this login is stored by a "Stats Profiler" which generates data reports in a manner fit for graphing. Our entire test-bed is displayed in Figure 10. Peers generated by the Static Peers Generator are colored blue or yellow if there are inactive or active, respectively.

We ran our complete simulation for a total of 20 iterations, culminating in a total of 28,800 login attempts. We display our results in Figure 11, which shows the percentage of liveness (i.e., successful logins over total logins in a time window) for each simulated hour, averaged over our 20 iterations. As indicated by Figure 11, Bingo's liveness averages between 80% and 95%. While this indicates that at certain points in a 24 h window that users may experience downtime, we still show that despite a low number of shares being distributed and no relay servers being used. As such, we conclude that our preliminary implementation of Bingo shows strong liveness and displays the necessary characteristics to be used as a real-world authentication system.

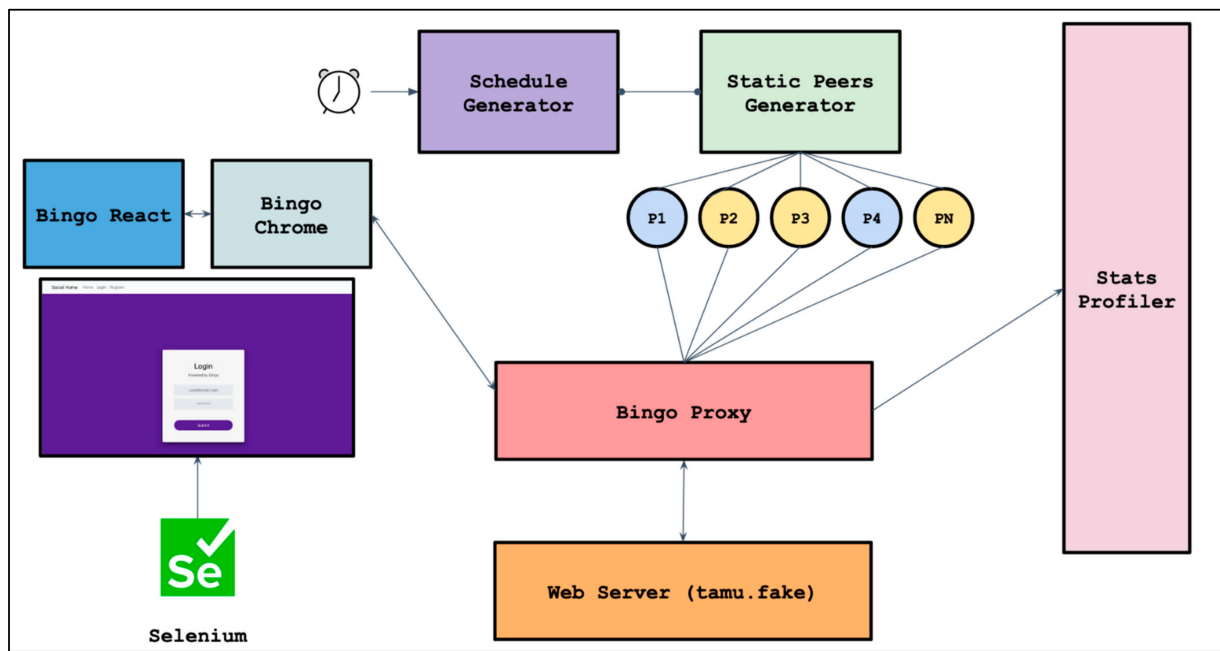


Figure 10. Test-bed used for running 24 simulated hours of logins.

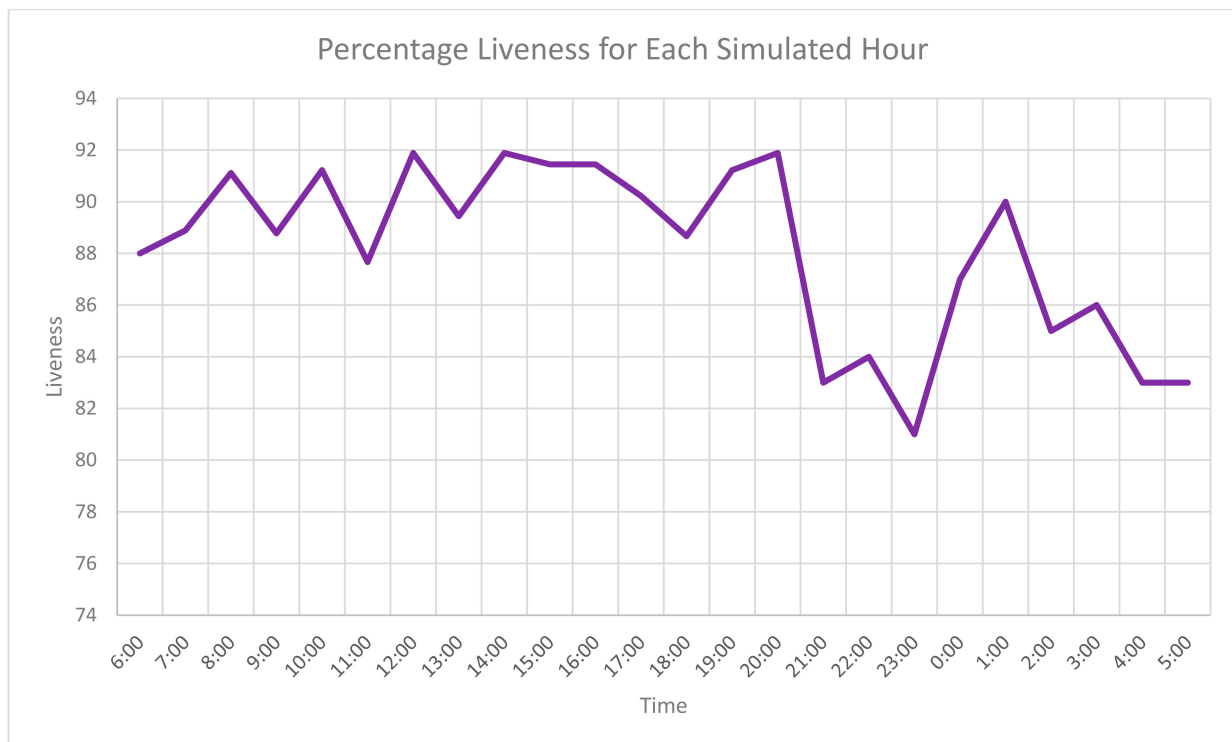


Figure 11. Percentage of liveness for each simulated hour, averaged over 20 iterations.

Bingo's current password share distribution scheme of randomly choosing active peers causes unnecessary overhead in the retrieval of said shares. While the goal is to never centrally store where shares are held, we aim to improve our distribution scheme to use the average activity levels of peers over time. As noted in previous sections, peers which display longer consecutive uptime are likely to continue to do so. Furthermore, it is possible that using machine-learning may aid Bingo in determining which peers are likely the best candidate to store shares.

### 7.3. Peer Uptime Simulator

To show that the results of the Peer Uptime Simulator are representative of real world distributed environments, the final evaluation phase places peers in a global setting, showing that even when peers are placed in far apart regions of the world, logging in is not affected by a great deal of increased latency.

The base case of the evaluation places 10, 50, and 100 peers in a single environment, as with our Peer Uptime Simulator. Latency for logging in is measured over 100 attempts of a user attempting to log in. Table 2 shows these results along with the remaining results of this evaluation.

**Table 2.** Evaluation result of places 10, 50, and 100 peers in a single environment.

Peers	Local Only (s)	Remote (s)
10	2.43	3.21
50	1.96	2.86
100	1.78	2.31

Next, a script is used to randomly select five regions where an AWS EC2 instance can be deployed, and a fifth of the number of peers (i.e., 10, 50, or 100) is placed in each region. The user is then re-registered to distribute their password shares among the peers, and the simulation is run again. This attempts to show that even with password shares being retrieved simultaneously from regions such as the USA, Asia, and Europe, logging in is not affected in terms of latency. The results of these simulations over 20 averaged runs show that in the very worst case where only 10 peers are used, meaning that the least amount of password shares are stored in the pool of peers, the increased latency of a global deployment is nearly 1 s, which is complexity negligible in the context of modern web requests.

Given that the Peer Uptime Simulator along with a global deployment both show that Bingo is able to scale in the modern web, the conclusion of this evaluation can be made that given correct configurations, Bingo is able to perform quite well as a login system overall, despite suboptimal conditions (i.e., highly distributed peers). Future iterations of this work will include tests showing optimized share distribution algorithms which will largely prevent the need for a massively global share distribution, as was shown here for demonstrative purposes.

## 8. Conclusions

This paper presented Bingo, a semi-centralized password storage system which aims to tackle the widespread insecure storage of passwords. The solution uses secret sharing and a series of cryptographic functions to ensure that a user's password is never attainable by an attacker from a single source. The various components which make up Bingo were discussed, and a real-world implementation of these components was provided. It appears that Bingo is able to provide strong security guarantees while also maintaining excellent liveness properties. This research aims to improve the distribution and retrieval mechanisms used by Bingo for password shares, as well as provide an improved evaluation in our future work.

**Author Contributions:** Conceptualization, A.F.A.-A. and M.B.; methodology, A.F.A.-A. and M.B.; software, A.F.A.-A. and M.B.; validation, A.F.A.-A. and M.B.; formal analysis, A.F.A.-A. and M.B.; investigation A.F.A.-A. and M.B.; writing—original draft preparation, A.F.A.-A. and M.B.; writing—review and editing, A.F.A.-A., M.B. and F.Y.A.-A.; visualization, A.F.A.-A. and M.B. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.



**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

- Gaw, S.; Felten, E.W. Password Management Strategies for Online Accounts. In Proceedings of the Second Symposium on Usable Privacy and Security, Pittsburgh, PA, USA, 12–14 July 2006; pp. 44–55.
- Sivertsen, B.; Knapstad, M.; Petrie, K.; O'Connor, R.; Lønning, K.J.; Hysing, M. Changes in mental health problems and suicidal behaviour in students and their associations with COVID-19-related restrictions in Norway: A national repeated cross-sectional analysis. *BMJ Open* **2022**, *12*, e057492. [[CrossRef](#)] [[PubMed](#)]
- Sushama, C.; Kumar, M.S.; Neelima, P. Privacy and security issues in the future: A social media. *Mater. Today Proc.* **2021**, *11*, 105. [[CrossRef](#)]
- Dubey, R.; Martin, M.V. Fool Me Once: A Study of Password Selection Evolution over the Past Decade. In Proceedings of the 18th International Conference on Privacy, Security and Trust (PST), Auckland, New Zealand, 13–15 December 2021; pp. 1–7.
- Venkatachalam, K.; Prabu, P.; Almutairi, A.; Abouhawwash, M. Secure biometric authentication with de-duplication on distributed cloud storage. *PeerJ Comput. Sci.* **2021**, *7*, e569.
- Luo, W.; Hu, Y.; Jiang, H.; Wang, J. Authentication by encrypted negative password. *IEEE Trans. Inf. Forensics Secur.* **2018**, *14*, 114–128. [[CrossRef](#)]
- Oesch, S.; Ruoti, S. That was then, this is now: A security evaluation of password generation, storage, and autofill in browser-based password managers. In Proceedings of the 29th USENIX Conference on Security Symposium, Boston, MA, USA, 12–14 August 2020; pp. 2165–2182.
- Morris, R.; Thompson, K. Password security: A case history. *Commun. ACM* **1979**, *22*, 594–597. [[CrossRef](#)]
- Gasti, P.; Rasmussen, K.B. On the security of password manager database formats. In Proceedings of the European Symposium on Research in Computer Security, Pisa, Italy, 10–12 September 2012; pp. 770–787.
- Provos, N.; Mazieres, D. A Future-Adaptable Password Scheme. In Proceedings of the USENIX Annual Technical Conference, FREENIX Track, Monterey, CA, USA, 6–11 June 1999; pp. 81–91.
- Gauravaram, P. Security Analysis of Salt || Password Hashes. In Proceedings of the International Conference on Advanced Computer Science Applications and Technologies (ACSAT), Kuala Lumpur, Malaysia, 26–28 November 2012; pp. 25–30.
- Han, A.L.-F.; Wong, D.F.; Chao, L.S. Password cracking and countermeasures in computer security: A survey. *arXiv* **2014**, arXiv:1411.7803.
- Belenko, A.; Sklyarov, D. “Secure Password Managers” and “Military-Grade Encryption” on Smartphones: Oh, Really? In Proceedings of the Blackhat Europe, Amsterdam, The Netherlands, 14–16 March 2012. 56p.
- Petsas, T.; Tsirantonakis, G.; Athanasopoulos, E.; Ioannidis, S. Two-factor authentication: Is the world ready? In Quantifying 2FA adoption. In Proceedings of the Eighth European Workshop on System Security, Bordeaux, France, 21 April 2015; pp. 1–7.
- Apriansyah, Y. Implementation of One Time Password (OTP) for Login Security on Web-Based Systems. Ph.D. Dissertation, University of Technology Yogyakarta, Daerah Istimewa Yogyakarta, Indonesia, 2022.
- Da Silva Torres, R.J. Identity Management: Analysis of Secure Authentication Propositions. Master’s Thesis, Universidade Do Porto, Porto, Portugal, 2020.
- Merdenyan, B.; Petrie, H. Perceptions of risk, benefits and likelihood of undertaking password management behaviours: Four components. In Proceedings of the IFIP Conference on Human-Computer Interaction, Paphos, Cyprus, 2–6 September 2019; pp. 549–563.
- Aziz, I.T.; Abdulqadder, I.H.; Alturfi, S.M.; Imran, R.M.; Flaih, F.M. A Secured and Authenticated State Estimation Approach to Protect Measurements in Smart Grids. In Proceedings of the International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT), Sakheer, Bahrain, 20–21 December 2020; pp. 1–5.
- Tomaszewska-Michalak, M. Biometric Technology 20 Years After 9/11—Opportunities and Threats. *Studia Politol.* **2022**, *63*, 123–134. [[CrossRef](#)]
- Tzagarakis, G.; Papadopoulos, P.; Chariton, A.A.; Athanasopoulos, E.; Markatos, E.P. Øpass: Zero-storage password management based on password reminders. In Proceedings of the 11th European Workshop on Systems Security, Porto, Portugal, 23–26 April 2018; pp. 1–6.
- Youssou, N.; Barais, O.; Blouin, A.; Bouabdallah, A.; Aillery, N. Requirements for preventing logic flaws in the authentication procedure of web applications. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, Limassol Cyprus, 8–12 April 2019; pp. 1620–1628.
- Rodríguez-Barroso, N.; López, D.J.; Luzón, M.; Herrera, F.; Martínez-Cámara, E. Survey on Federated Learning Threats: Concepts, taxonomy on attacks and defences, experimental study and challenges. *arXiv* **2022**, arXiv:2201.08135.
- Aron, G. Improving attacks on round-reduced speck32/64 using deep learning. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 2019; Springer: Cham, Switzerland; pp. 150–179.

24. Singh, A.; Tiwari, V.; Naidu, A.S.; Tentu, A.N.; Raju, K.S.; Saxena, A. Analysis of Password Protected Documents Using Statistical Approaches on High Performance Computing. In *Advances in Micro-Electronics, Embedded Systems and IoT*; Springer: Singapore, 2022; pp. 533–545.
25. Kampourakis, V.; Kambourakis, G.; Chatzoglou, E.; Zaroliagis, C. Revisiting man-in-the-middle attacks against HTTPS. *Netw. Secur.* **2022**, 2022. [[CrossRef](#)]
26. Goulart, A.; Chennamaneni, A.; Torre, D.; Hur, B.; Al-Aboosi, F.Y. On Wide-Area IoT Networks, Lightweight Security and Their Applications—A Practical Review. *Electronics* **2022**, *11*, 1762. [[CrossRef](#)]