Distributed Single Password Protocol Framework

Devriş İşler, Alptekin Küpçü

Koç University, İstanbul, Turkey
{disler15,akupcu}@ku.edu.tr

Abstract. Passwords are the most widely used factor in various areas such as secret sharing, key establishment, and user authentication. Single password protocols are proposed (starting with Belenkiy et. al [4]) to overcome the challenges of traditional password protocols and provide provable security against offline dictionary, man-in-the-middle, phishing, and honeypot attacks. While they ensure provable security, they allow a user securely to use a single *low-entropy human memorable* password for all her accounts. They achieve this with the help of a cloud or mobile storage device. However, an attacker corrupting both the login server and storage can mount an offline dictionary attack on user's single password. In this work, we introduce a framework for distributed single password protocols (DiSPP) that analyzes existing protocols, improves upon them regarding novel constructions and distributed schemes, and allows exploiting alternative cryptographic primitives to obtain secure distributed single password protocols with various trade-offs. Previous single password solutions can be instantiated as part of our framework. We further introduce a secure DiSPP instantiation derived from our framework enforcing the adversary to corrupt several cloud and mobile storage devices in addition to the login server in order to perform a successful offline dictionary attack. We also provide a comparative analysis of different solutions derived from our framework.

Keywords: Password, authentication, offline dictionary attack.

1 Introduction

Passwords are used in many different contexts such as authentication, key exchange, and sensitive information storage. In password-based authentication, a user registers with a server using her low-entropy password. The server stores the user's information (e.g. $< username, hash(password) >$). Later on, the user authenticates herself to the server whenever she wants to get a service. In password authenticated key exchange, following registration, a user and a server wish to establish a session key for a secure and authenticated channel [5,12,13,35,36]. In password protected sensitive information storage, a user stores her sensitive

information (e.g. credentials) among a server or multiple servers using her password [11, 15, 16, 28, 30]. Whenever she wants to reconstruct the sensitive information, she has to convince the server(s) that she is the legitimate user holding the correct password.

However, passwords are vulnerable to many prevalent online and offline attacks such as *phishing*, *man-in-the-middle*, *honeypot*, and *offline dictionary* attacks. The damage of a successful attack increases with password reuse, which is common in practice [22]. This brings many issues, where an attacker compromising user's password can use it to gain access to the services on behalf of the user. Such an attacker may be a malicious server (as in phishing, honeypot, or man-in-the-middle attacks), or hackers obtaining the server database and mounting offline dictionary attacks. Indeed, such attacks are very prevalent, and recent studies even propose improved offline dictionary attacks [40].

Single password protocols (starting with Acar et al. [2] with their patent application dating 2010 [4], Jarecki et al. [31], and İşler and Küpçü [26]) are proposed to provide *provable security* against the aforementioned attacks that traditional password protocols are vulnerable to. These protocols enable users to use a single password for all their accounts securely, where users do not store any information locally. They achieve this with the help of a cloud or mobile storage device. Among previous works, only İşler and Küpçü [26] employ a distributed protocol for cloud storage.

The general idea of a distributed single password protocol (DiSPP) is to create a high entropy secret *independent* of the user's password and registering a verification information based on this secret with the login server(s). Later, the user stores the secret among storage providers (e.g. personal devices, online storage providers) using her password. Whenever the user wants to authenticate herself (implicitly or explicitly) to the login servers, she reconstructs the secret and authenticates with the server accordingly.

In this paper, we present a framework for *distributed* single password protocols that can employ possibly *more than one* storage provider (any combination of cloud and mobile devices) and *more than one* login server. Our framework consists of four phases: registration and authentication (between a user and login servers) and secret storage and retrieval (between a user and storage providers). We discuss possible cryptographic building blocks that can be employed in these phases so that the combination of these building blocks constitute a secure DiSPP solution.

In our framework, we employ a total of n_{stor} storage providers and n_{ls} login servers with a threshold $1 \leq t_{stor} \leq n_{stor}$ for the storage providers and a threshold $1 \leq t_{ls} \leq n_{ls}$ for the login servers. This setting serves two purposes. Firstly, for an adversary to be able to successfully mount an offline dictionary attack, he must corrupt t_{ls}-many login servers in addition to t_{stor}-many storage providers. Secondly, to login, the user must access t_{stor} storage providers out of n_{stor}; thus availability can be balanced against security easily by setting these parameters. While the underlying techniques are different, in terms of security, all previous

solutions correspond to setting $t_{ls} = n_{ls} = 1$ and $t_{stor} = n_{stor} = 1$ (except for İşler and Küpçü [26]).

Our **contributions** can be summarized as follows:

1. We present a framework for distributed single password protocols (DiSPP), where we discuss the features of a DiSPP to resist against **offline dictionary, phishing, man-in-the-middle, and honeypot** attacks.
2. We identify cryptographic building blocks to build a secure DiSPP. We provide possible solutions with or without server-side changes and various network channel requirements (e.g. secure-but-unauthenticated channel).
3. We present a distributed single password protocol instantiation derived from our framework where a user can define an access control mechanism for storage providers (e.g. a personal mobile device must participate to reconstruct the secret). Our instantiation is secure against offline dictionary attacks where an adversary corrupts t_{ls}-1 *login servers* and all storage providers except an authorized set of storage providers.
4. Our DiSPP instantiation is also secure against phishing and man-in-the-middle attacks during authentication, after a secure registration, and honeypot attacks during registration and authentication.
5. We present **performance** evaluations numerically, showing the efficiency of different existing building blocks employed in our framework.

Related Work: In the literature, Acar et al. [2] (with their patent application dating 2010 [4]), Jarecki et al. [31], and İşler and Küpçü [26] are the only known provably secure DiSPP solutions (against offline dictionary attacks). Similarly, Bicakci et al. [8] present a solution idea, but without a formal security proof. These solutions include a single storage provider (which is either a cloud storage or a mobile device) for the secret storage, except İşler and Küpçü [26] employ a threshold of cloud storage providers. These DiSPP instantiations are realized by our framework. Therefore, we discuss how the existing works can be instantiated within our framework (see Section 6.1) and what kind of features they provide (e.g. no server-side change, security against eavesdropping, man-in-the-middle). Moreover, our proposed DiSPP construction is inspired from and improves upon them.

Overview: A DiSPP achieves full threshold security as follows: A *k-bit* entropy secret (e.g., a k-bit random string *rnd*) *independent* of the password *pwd* is created and a verification information based on this secret is computed (e.g., $Hash(rnd||ls)$ where *ls* is the domain name of the login server). Then, the verification information is shared with the login servers whereas the secret is securely shared with the storage providers (e.g., mobile devices, online storage providers). But, these shares are not directly sent to the storage providers, since these shares should be retrieved only by the legitimate user holding the correct password. Therefore, the shares are stored using a password protected secret sharing solution, where only the user holding the correct password can reconstruct the secret and up to threshold-many storage providers cannot perform an offline dictionary attack on the user password. Using this k-bit entropy secret, the user may authenticate or establish keys with the login servers.

For authentication, the user interacts with threshold-many storage providers, and reconstructs the original secret by running the password protected secret sharing reconstruction protocol using her password. Then, the user computes the authentication protocol with threshold-many login servers using the secret. Only when at least threshold-many (t_{ls}) login servers and an authorized set of (or threshold-many) storage providers collude, they can reconstruct the secret via an offline dictionary attack by trying different passwords and running the authentication protocol internally. Otherwise, offline dictionary attacks are impossible. Consider, for example, an adversary corrupting t_{ls} login servers and $t_{stor} - 1$ storage providers. He needs to interact online with at least one honest storage provider to be able to reconstruct the secret, which is an online attack whose rate or the number of attempts can be limited. Moreover, consider that t_{stor}-many storage providers are colluding while the login servers are honest. In this case, even though they can reconstruct a secret, since all passwords yield to a valid secret (in terms of format), they can only try to authenticate online with the login server to verify whether or not the secret they constructed is the correct one.[1] Again, this online attack can easily be rate limited.

2 Preliminaries

Let $\lambda \in N$ be the security parameter. A probabilistic polynomial time (PPT) algorithm A is a probabilistic algorithm taking 1^λ as an input and has running time bounded by a polynomial in λ. We say that a function $negl(\lambda)$ is negligible if for every positive polynomial $poly(\lambda)$ there exists a constant $\lambda' \in N$ such that $\forall \lambda > \lambda'$ $negl(\lambda) < 1/poly(\lambda)$. $||$ denotes concatenation.

Hash Function: A hash function H (chosen from a family of such functions) is a deterministic function from an arbitrary size input to a fixed size output, denoted $H : \{0,1\}^* \rightarrow \{0,1\}^l$. The hash function is assumed to be *collision resistant* if it is hard to find two different inputs $x \neq y$ that hash to the same output $H(x) = H(y)$.

Oblivious Pseudorandom Function (OPRF): A pseudorandom function (PRF) F is a deterministic function that takes two inputs: a secret key k and an input x to compute on, and outputs $F_k(x)$. A function chosen randomly from a PRF family (a PRF with random key k) is secure if it is distinguishable from a random function with the same domain and range with only negligible probability for all PPT distinguishes given oracle access. An Oblivious PRF (OPRF) [24] is a protocol between two parties (*sender* and *receiver*) that securely computes $F_k(x)$ where k and x are the inputs of *sender* and *receiver*, respectively, such that the sender learns *nothing* from the interaction and the receiver learns *only* $F_k(x)$. Threshold Oblivious Pseudorandom Function (TOPRF), introduced by [30], is a PRF between multiple senders and a receiver. A Unique Blind

[1] Some early password protected secret sharing methods perform authentication of the user by the storage providers (e.g. [16]), which would mean security up to a collusion of $t_{stor} - 1$ storage providers.

Signature [10] (where the signer acts as the sender) can be used as an alternative to OPRF.

Symmetric Encryption Scheme: It consists of three PPT algorithms: $KeyGen(1^\lambda)$ generates a secret key sk, $Enc_{sk}(msg)$ encrypts the message using the secret key and outputs the ciphertext c, and the decryption algorithm $Dec_{sk}(c)$ uses the secret key sk to decrypt the ciphertext c, and outputs the original message msg. The encryption scheme we use needs to be semantically secure (encryptions of different messages are indistinguishable).

Message Authentication Code (MAC): A MAC scheme is a symmetric scheme consisting of three PPT algorithms: $MACKeyGen(1^\lambda)$ generates a key K, $MAC_K(msg)$ generates a MAC tag sig on the message msg using the MAC key K, and $MACVerify_K(sig, msg)$ outputs **accept** if the sig is valid for the given message msg using the MAC key K, and outputs **reject** otherwise. The MAC we employ needs to be secure against adaptive existential forgery attacks [33], meaning that even though the adversary adaptively obtains many msg, sig pairs on his choice of messages, he cannot forge a valid sig on a new message.

Digital Signature: A digital signature scheme is an asymmetric scheme consisting of three PPT algorithms, where $SignKeyGen(1^\lambda)$ generates a secret signing key ssk and a public verification key svk, $Sign_{ssk}(msg)$ generates a signature σ on the message msg using secret signing key ssk, and $SignVerify_{svk}(\sigma, msg)$ outputs *accept* if the given signature σ is a valid signature on msg given the public verification key svk, and outputs *reject* otherwise. The digital signature scheme we employ needs to be secure against adaptive existential forgery attacks (no PPT adversary holding svk can come up with a valid signature on a new message that the oracle has not created a signature on).

2.1 Secret Sharing (SS)

A secret sharing (SS) scheme is a method such that a dealer holding a secret (user in our context) distributes the secret among participants in a way that only the authorized set of participants can reconstruct the secret. The authorized subset varies depending on the secret sharing scheme (e.g. any subset with threshold many participants). The security is that any unauthorized subset of participants cannot reveal any partial information about the secret. For simplicity, let the access structure Γ be a collection of sets of authorized participants who can reconstruct the original secret, and γ denote an authorized set such that $\gamma \in \Gamma$. $\{s_i^\gamma\}$ refers all elements (shares) in an authorized set γ. An SS protocol consists of two PPT algorithms: $\{s_i{}^\gamma\}_{\gamma\in\Gamma} \leftarrow SS(S, \Gamma)$ to create the shares of the secret S, and we use $S \leftarrow SSRecon(\{s_i^\gamma\}, \Gamma)$ to reconstruct the original secret.

Threshold Secret Sharing (TSS): In a TSS protocol [9, 39], any subset containing threshold-many participants is an authorized set (i.e. for each γ, we have $|\gamma| = t$). The security is that fewer than threshold-many (t) shares provide no information regarding the original secret.

Access Controlled Secret Sharing (ACSS): An ACSS [6, 7, 14, 19, 27] is the general form of secret sharing, where Γ is defined explicitly, as long as it is monotonic (i.e. if some $\gamma \in \Gamma$ then for any χ such that $\gamma \subset \chi$ we have $\chi \in \Gamma$).

Non-Interactive Verifiable Secret Sharing (NIVSS): NIVSS was introduced by [37], where the shareholder can verify whether or not the share received is consistent with other shares without learning the secret itself. The secret sharing and reconstruction algorithms employ some verification information regarding the shares: $\{s_i^\gamma, v_i^\gamma\}_{\gamma\in\Gamma} \leftarrow NIVSS(S, \Gamma)$ to create the shares s_i and corresponding proofs v_i and $S \leftarrow NIVSSRecon(\{s_i^\gamma, v_i^\gamma\}, \Gamma)$ to reconstruct and verify the original secret. In addition to secret sharing and reconstruction algorithms, an NIVSS has a third PPT algorithm $0/1 \leftarrow NIVSSVerify(s_i, v_i)$ that is used to verify that the share s_i is a valid share using the proof v_i. Any party holding s_i, v_i can run $NIVSSVerify(s_i, v_i)$ offline.

Password Protected Secret Sharing (PPSS): A PPSS [1,3,16,28–30] is an SS with the password being involved both in secret sharing and reconstruction steps: $\{share_i^\gamma\}_{\gamma\in\Gamma} \leftarrow PPSS(pwd, S, \Gamma)$ to create the shares of the secret S protected by the password pwd, and $S \leftarrow PPSSRecon(pwd, \{share_i^\gamma\}, \Gamma)$ to reconstruct the original secret. The security is that fewer than threshold-many shares provide no information regarding the original secret and only the legitimate user who knows the password can reconstruct the secret.

2.2 Server-Side Password Protocols

A *server-side password protocol* is a protocol between a user and n_{ls}*-many* login servers. Basically, a user holding a low entropy human-memorable password registers with the servers. During registration, the servers store a verification information based on user password and/or a key. Later on, the user and *threshold-many* (t_{ls}) servers jointly compute a password protocol where the user and each server receive outputs as $output_U$ and $output_i$, respectively. A server-side password protocol has two purposes in the literature (to the best of our knowledge):

1. **Key Establishment:** A user and (login) server(s) jointly compute a *password authenticated key exchange (PAKE) protocol* to establish a joint session key K. Later on, they use K for establishing secure and authenticated communication channel between the server(s) and the user. This is also called **implicit authentication**, since establishment of a joint key ensures that the user is the one who indeed previously registered with the server. The key K can be generated from symmetric (PAKE) or asymmetric (APAKE) information. [5, 12, 13, 31, 32, 35, 36] are some examples.
   Observe that standard (A)PAKE solutions depend on the complexity of the password for security, and are vulnerable to offline dictionary attacks, whereas in DiSPP, security will depend on the k-bit entropy secret generated independently of the password and offline dictionary attacks will not be possible.
2. **User Authentication:** A user and login server(s) compute a password protocol where server(s) authenticate the user. The authentication can be either symmetric (e.g. based on stored hashes) or asymmetric (e.g. based on signature verification). Message authentication code (MAC) [41], digital signatures [25], and identification schemes [20, 21, 38] are some of the known examples of such **explicit authentication** schemes.

3 Distributed Single Password Protocol (DiSPP) Framework

In a DiSPP, there are three types of players. There are ***users*** who register with one or more ***login servers*** using (possibly) the same password, and later on compute a server-side password protocol with these login servers. For this purpose, the users securely store some secret information (that is needed for the password protocol with the login servers) at one or more ***storage providers***, which consist of personal user devices (e.g. mobile phone, tablet) and online storage providers (e.g. Dropbox, Google Drive). The main objective of a DiSPP solution is to protect the user's password against offline dictionary attacks by the storage providers, the login servers, and many other adversaries (including honeypots and phishing sites).

In a DiSPP, the user creates a *k-bit* entropy secret (e.g., k-bit random string rnd) *independent* of the password pwd (which only has l-bit entropy such that $l << k$). The password is assumed to resist only *online* dictionary attacks, but is not secure enough to resist offline dictionary attacks. Then, associated verification information based on the secret is computed (e.g., $H(rnd||ls)$ where ls is the domain name of the login server) and shared with the login servers depending on the server-side protocol, whereas the k-bit entropy secret is securely shared with the storage providers. This independent generation of the k-bit entropy secret for each registration enables a DiSPP to employ a single low-entropy password securely. But to authenticate or establish keys with the login server, the user needs to retrieve this secret from the storage providers (because it is of high entropy, it is not human-memorable). During the secure storage and retrieval of the secret at the storage providers, a DiSPP should ensure that only the legitimate user holding the correct password can retrieve these shares and reconstruct the secret. Since verifying whether or not the user is legitimate requires another authentication step (causing a chicken-egg problem), a DiSPP solves this paradox by enforcing the user (and hence any attacker) to employ an online protocol with the storage providers (e.g. OPRF).

A DiSPP consists of four main phases (see Figure 1): **Registration** where the user registers with login server(s) creating a high entropy secret and its associated verification information, **Secret Storage** where the user *securely* stores the secret using her password at the storage providers, **Secret Retrieval** where the user reconstructs the secret using her password employing (an authorized subset of) storage providers, and **Authentication** where the user implicitly or explicitly authenticates herself to the login server(s) using the secret reconstructed during the *secret retrieval* phase.[2]

The **registration** phase is for the user to register with the login server(s) with domain name ls. The user registers using a low-entropy password pwd (only secure against *online* attacks). Each login server obtains the user's verification information $vInfo_i$ such that the login server can authenticate the legitimate

[2] Protocols may include some username information, but we choose not to complicate our presentation with things not directly associated with security.

user (implicitly via a key establishment protocol or explicitly). The user obtains a k-bit entropy secret information S that is associated with the verification information to facilitate later protocols. More formally we have the following multi-party protocol:

Registration:

- **The user's input** is a password *pwd* and the server identifier *ls* (e.g. domain/url).
- **Each login server's input** is an identifier *ls*.
- **The user receives as output** a secret S (which will be used in secret storage) associated with the verification information.
- **Each login server receives as output** a verification information $vInfo_i$ based on the secret S, and stores this information in his database. The verification information $vInfo_i$ is used by the login servers to authenticate the user implicitly or explicitly during the authentication protocol.

The **secret storage** phase is for the user to store the $k-bit$ entropy secret S (generated during registration) among n_{stor}-many storage providers using a low entropy password. Each storage provider obtains a secret share $share_i$, while the user obtains nothing.

Secret Storage:

- **The user's inputs** are a password *pwd*, and the secret S.
- **Each storage provider receives as output** a share $share_i$ and stores the data received in its database.

The **secret retrieval** phase is for the user to retrieve and reconstruct the secret needed for authentication by interacting with an authorized subset of (e.g. *threshold* t_{stor}-many) storage providers.

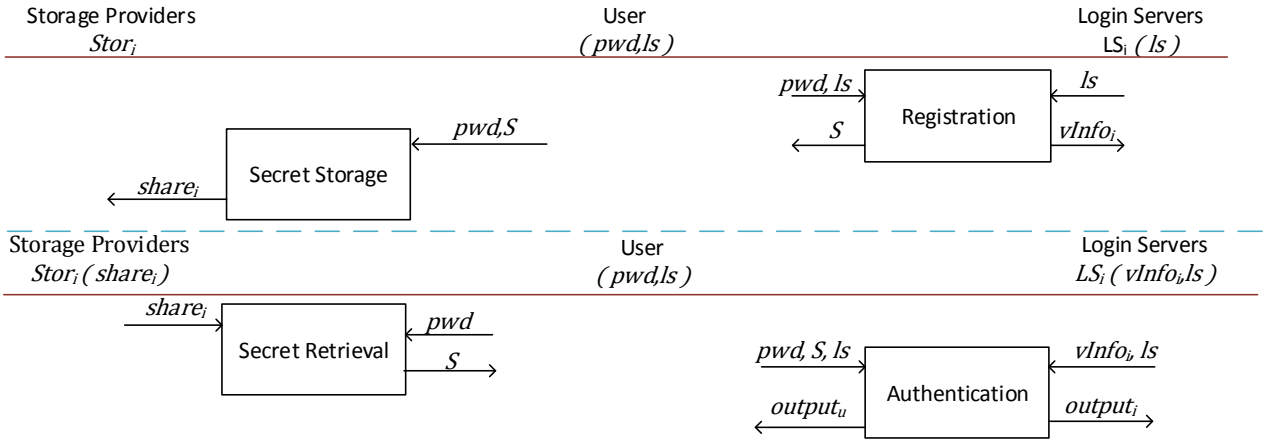
Secret Retrieval:

- **The user's input** is the password *pwd*.
- **Each storage provider's input** is the share $share_i$ that they hold for that user.
- **The user receives as output** the reconstructed secret S.

4 Dispp Construction

We now present how to achieve a secure DiSPP in our framework by employing secure cryptographic building blocks. We firstly discuss possible cryptographic solutions for registration and authentication between *a user* and *one* login server (and later extend to n_{ls}-many login servers). Secondly, we present possible cryptographic solutions for the secret storage and retrieval phases that takes place between a *user* and *storage provider(s)*.

Fig. 1. DiSPP Overview



4.1 Registration and Authentication Phases

Since registration and authentication phases are both server-side protocols, they need to be picked to work together. In a DiSPP, the essential step in the registration phase is to create a k-bit entropy secret *independent* of the user password, together with the associated verification information. Depending on the password protocol employed, the structures of the secret S and verification information $vInfo$ generated vary. Below we present and discuss various schemes that can be employed for registration and authentication purposes securely. We start by implicit authentication (key establishment) and then continue with explicit authentication methods.

Key Establishment (Implicit Authentication): A user and login server(s) run a setup phase where later on they can jointly compute a password authenticated key exchange protocol to establish a session key for a secure and authenticated channel. Standard PAKE protocols keep the password and APAKE protocols keep a deterministic function of the password (e.g. $H(pwd)$) at the server database, they are insecure against offline dictionary attacks. In a DiSPP, to provide provable security, key establishment works over the k-bit entropy secret S rather than the passwork. The **registration** and **authentication** of key exchange protocol is as follows;

Registration: The **user**:

- generates a k-bit entropy random string rnd as $rnd \leftarrow \{0,1\}^k$ or by running a key generation scheme of OPRF as $K \leftarrow OPRFKeyGen(1^\lambda)$.
- computes the verification information $vInfo$ as $H(rnd)$ or $F_K(pwd)$ using the random string or OPRF key respectively.
- sends $vInfo$ and assigns random strings rnd or K as the secret S.

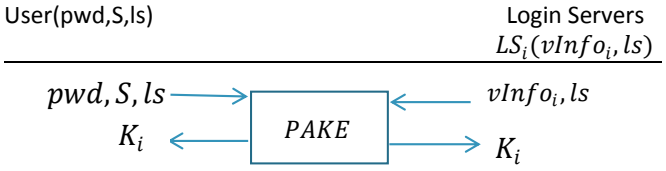
The **login servers** receive $vInfo$ and store it.

Authentication: The user and login server(s) jointly compute a password authenticated key exchange protocol to establish a session key K. Later on, they can use K for establishing a secure and authenticated communication channel, where the server(s) implicitly authenticate the user. A PAKE protocol is computed as follows;

- The **user** holding the secret S, password pwd and domain name of login server ls computes a PAKE protocol with the login server(s) [5,12,13,31,32, 35, 36], where each login server holds a verification information $vInfo_i$ and domain name ls (see Figure 2).
- The **user** and the **login server(s)** receive a session key K_i, where the user receives a session key per login server.

The security at the login server side can be increased using a threshold-PAKE (TPAKE) solution [28,34,36]. TPAKE settings involve n_{ls} login servers such that a threshold t_{ls} of them must participate in the user authentication. An attacker corrupting any fewer than t_{ls} servers *cannot* perform an offline dictionary attack on the user's password.

Fig. 2. Password authenticated key exchange protocol for implicit authentication.



Explicit Authentication: We categorized explicit authentication protocols into two: interactive (e.g. challenge-response) and non-interactive (e.g. $H(S||ls)$). In explicit authentication, we assume a single login server (i.e. $t_{ls} = n_{ls} = 1$).[3] Firstly, we describe the registration and authentication phases of interactive authentication protocols and later we discuss the non-interactive authentication protocols.

For all interactive authentication protocols below, the login server sends a challenge *chal* to the user.Then, the **user** runs the authentication scheme (based on the registration phase) to generate a response *resp* based on the challenge *chal* using her secret S and the domain name ls of the login server.

[3] [23] proposed two-servers setting, where there is a main login server the user wants to authenticate herself with, and a helper login server such that the adversary needs to corrupt both.

MAC-based Registration: The **user**:

- generates a MAC key K by running key generation algorithm as $K \leftarrow MACKeyGen(1^\lambda)$ (see Figure 3(a)).,
- sends the MAC key K as the verification information key $vInfo$ to the login server,
- assigns the MAC key as the secret ($S = K$).

The **login server** receives and stores $vInfo = K$.

MAC-based Authentication: The user runs $resp \leftarrow MAC_K(chal||ls)$ and sends the response $resp$ to the login server, who runs $MACVerify_K(resp, chal||ls)$ (see Figure 3(b)).

Digital Signature-based Registration: The **user**

- generates signing and verification keys by running digital signature key generation algorithm as $ssk, svk \leftarrow SignKeyGen(1^\lambda)$ (see Figure 3(c)),
- sends the signature verification key svk as the verification information key $vInfo$ to the login server,
- assigns the signing key as the secret ($S = ssk$).

The **login server** receives and stores $vInfo = svk$.

Digital Signature-based Authentication:
The user runs $resp \leftarrow Sign_{ssk}(chal||ls)$ and sends the response $resp$ to the login server, who runs $SignVerify_{svk}(resp, chal||ls)$ (see Figure 3(d)).

OPRF-based Registration: The **user**

- generates a key K by running OPRF key generation algorithm as $K \leftarrow OPRF(1^\lambda)$ (see Figure 3(e)),
- sends verification information as $vInfo = F_K(pwd)$,
- assigns the OPRF key as the secret ($S = K$).

The **login server** receives and stores $vInfo = F_K(pwd)$. **OPRF-based Authentication:** The user computes $vInfo \leftarrow F_K(pwd)$ where K is the secret S (see Figure 3(f)) and then sends the response as $resp = H(chal||ls||vInfo)$ to the login server, who checks whether locally computed $H(chal||ls||vInfo)$ is equal to the response of the client or not.

Hash-based Registration: The **user**:

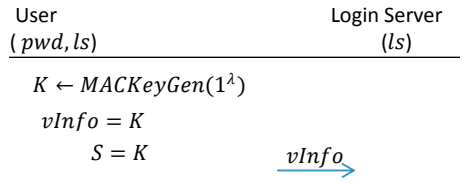
- generates a *k-bit* entropy random string rnd as $rnd \leftarrow \{0,1\}^k$ (see Figure 3(g)),
- sends the verification information as $vInfo = H(rnd||ls)$ to the login server,
- assigns the generated random as the secret ($S = rnd$).

The **login server** receives and stores $vInfo = H(rnd||ls)$.

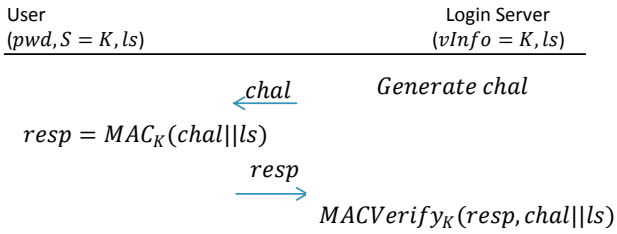
Hash-based Authentication: The user computes the response as $resp \leftarrow H(chal||ls||H(rnd||ls))$ (see Figure 3(h)) and sends $resp$ to the login server, who checks whether locally computed $H(chal||ls||vInfo)$ is equal to the response of the client or not.

Fig. 3. Registration and authentication phases of interactive authentication schemes.

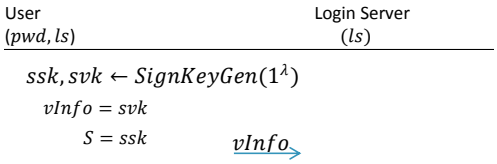
(a) MAC-based registration.



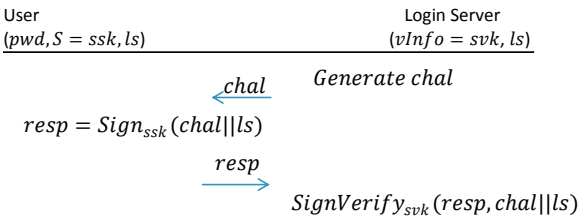
(b) MAC-based authentication.



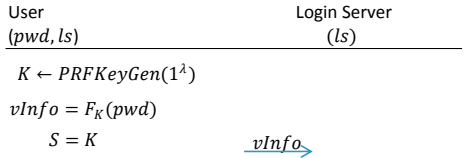
(c) Digital signature-based registration



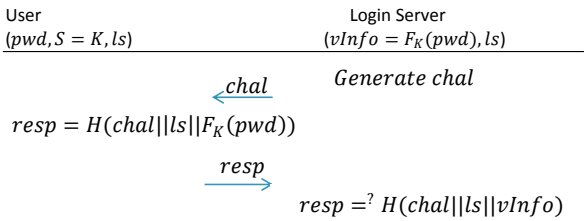
(d) Digital signature-based authentication.



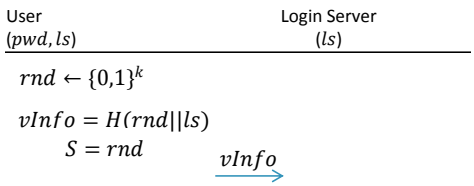
(e) OPRF-based registration



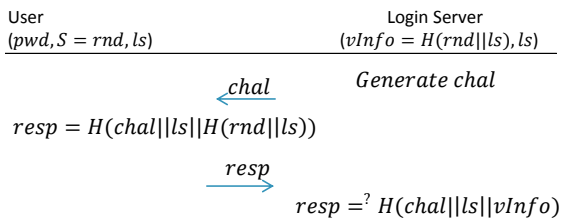
(f) OPRF-based authentication



(g) Hash-based registration



(h) Hash-based authentication



Non-interactive authentication: In non-interactive authentication, the user sends only one message to the login server as a login request. Such a round-optimal solution can be based on challenge-response (non-interactive transformation of interactive authentication) or one-way functions such as a hash or a pseudo-random function. Non-interactive authentication **requires a secure and server-authenticated communication channel**, since it does not provide security against eavesdropping and replay attacks (except maybe some synchronous time-based measures). The reason is that an attacker can use the messages captured in earlier authentications to impersonate the user to the login server.

To convert the interactive protocols above to their non-interactive versions, the registration phases of the protocols above do not change. During authentication, the user picks a challenge *chal* randomly and sends $chal, resp$ to the server, who performs the same verification as above.

Alternatively, assuming $vInfo = H(rnd||ls)$ can be seen as a site-specific password, the hash-based authentication method can be simplified to the user computing $resp \leftarrow H(rnd||ls)$ and sending $resp$ to the login server, who checks it against the locally stored $vInfo$ (or $H(vInfo)$ or $H(vInfo, salt)$) **without the need to modify the existing login servers**.

4.2 Secret Storage and Secret Retrieval Phases

We discuss possible building blocks for the secret storage and retrieval phases. The user stores the secret S (generated during the registration phase) among n_{stor} storage providers using her single password. Password protected secret sharing (PPSS) enables only the user who has the correct password to reconstruct the secret during secret retrieval. We firstly describe overview of password protected secret sharing employed in secret storage and retrieval phases. Later, we show how existing building blocks can be employed to have a secure PPSS.

PPSS-based Secret Storage:
The user runs $\{share_i{}^{\gamma}\}_{\gamma \in \Gamma} \leftarrow PPSS(pwd, S, \Gamma)$ and sends $share_i$ to storage provider i, who stores it in its database.

PPSS-based Secret Retrieval:
The user communicates with *threshold-many* storage providers to reconstruct the secret S via running the $S \leftarrow PPSSRecon(pwd, \{share_i{}^{\gamma}\}, \Gamma)$ protocol with t_{stor}-many storage providers where $share_i$ and pwd are the inputs of each authorized storage provider and the user, respectively.

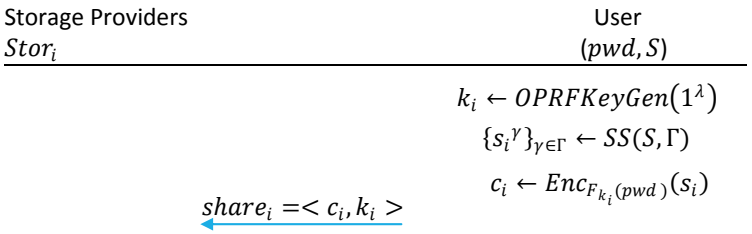
A PPSS can be constructed using any secret sharing scheme together with an OPRF as follows: The user (dealer) runs the secret sharing method on a secret S to create its shares. To protect the shares, the user encrypts each share using OPRF evaluation result on the password. In details, the secret sharing and OPRF based solution (storage and retrieval) works as follows;

Secret Sharing and OPRF-based Secret Storage:

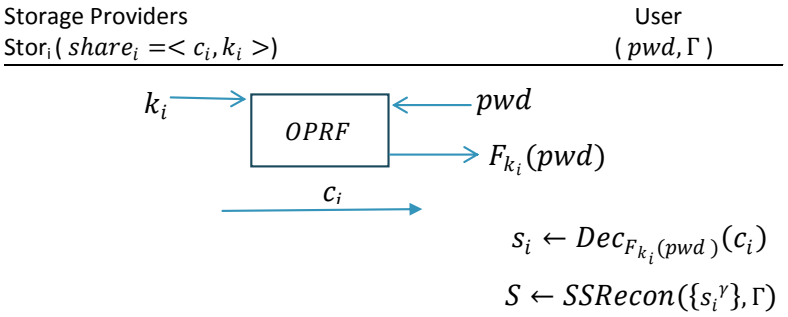
- **The user**
  1. generates an *OPRF* key $k_i \leftarrow OPRFKeyGen(1^{\lambda})$ per storage provider,

Fig. 4. PPSS secret storage and reconstruction based on SS-and-OPRF.

(a) SS-and-OPRF-based secret storage.



(b) SS-and-OPRF-based secret reconstruction.



2. runs a secret sharing scheme on the secret S to create the shares as $\{s_i{}^\gamma\}_{\gamma\in\Gamma} \leftarrow SS(S,\Gamma)$,
3. encrypts each share using *oblivious pseudorandom function* of the password pwd using generated $OPRF$ key of each storage provider obtaining $c_i \leftarrow Enc_{F_{k_i}(pwd)}(s_i)$,
4. sends $share_i = (c_i, k_i)$ to each storage provider.

– **Each storage provider** receives a share $share_i$ and stores it in his database.

Secret Sharing and OPRF-based Secret Retrieval:

1. **The user and each storage provider jointly** execute the oblivious pseudorandom function (OPRF) protocol. Each storage provider acts as the sender and the user acts as the receiver in these protocol executions. The user obtains the OPRF value (with key k_i) of the password $F_{k_i}(pwd) \leftarrow OPRF(pwd, k_i)$ as the output.
Remark: The $OPRF$ result is received *only* by the user and the storage

providers are oblivious to the password. It is enough that this protocol is run among some authorized subset of storage providers and the user, rather than all storage providers.
2. **Each storage provider** sends c_i to the user.
3. **The user** decrypts each ciphertext c_i using the corresponding OPRF output already received to obtain the secret shares $s_i \leftarrow Dec_{F_{k_i}(pwd)}(c_i)$ and computes threshold secret sharing reconstruction algorithm to reconstruct the secret as $S \leftarrow SSRecon(\{s_i^\gamma\}, \Gamma)$.

The secret sharing employed takes an important role for security of DiSPP. In the following, we investigate the secure secret sharing and OPRF instances. If a **threshold secret sharing** scheme is employed [2,26,30,31], the user runs the secret retrieval phase with threshold-many (t_{stor}) storage providers. Alternatively, one can employ an **access control secret sharing** solution. Our framework allows usage of such a solution securely, and this brings more flexibility to the protocol as discussed below.

Consider, for example, a user employing n_{stor} storage providers in total, which consist of n_{mob} user personal devices (e.g. mobile phone, tablet) and n_{os} online storages (e.g. Dropbox). The user can define the authorized set of participants by choosing the thresholds as t_{mob} and t_{os} for specified personal devices and online storage providers, respectively. This means, to reconstruct the secret, (at least) t_{mob} personal devices and t_{os} online storage providers need to be involved in the secret retrieval phase.

For instance, consider a user who stores the secret by employing Dropbox (D) and GoogleDrive (G) as online storage providers and her mobile phone (M) and tablet (T) as personal devices. The scenario described above with mobile and online storage thresholds being one correspond to setting $\Gamma = \{\{D, M\}, \{D, T\}, \{G, M\}, \{G, T\}\}$ (and sets containing these subsets). Thus, the user retrieves the secret by accessing one of the online storage providers and one of her personal devices. Any attacker, who corrupts a set χ such that for any $\gamma \in \Gamma$ we have $\gamma \nsubseteq \chi$, cannot perform an offline dictionary attack to find user password. In particular, in the scenario above, consider an attacker who corrupts both online storage providers. As long as the attacker does not additionally have access to a personal device of the user, the protocol remains secure against offline dictionary attacks.

5 Inappropriate uses of primitives

5.1 Offline dictionary attack on NIVSS-and-OPRF-based solution

In DiSPP, the security property of secret sharing plays an important role. If NIVSS and OPRF based PPSS is employed in a distributed password protocol as secret storage, then an attacker can successfully perform an offline dictionary attack to find user password. The attack scenario is as follows;

A user runs secret storage protocol with n_{stor} storage providers described as in Figure 4(a), where NIVSS is employed as SS along with OPRF.

Consider an attacker $\mathcal{A}$, holding a dictionary $\mathcal{D}$ containing the user password, corrupts one of the storage providers called $Stor_c$, where the storage provider is holding a share as $< k_c, c_c >$. The attacker $\mathcal{A}$ runs the attack code as described in Algorithm 1 to find the user password. Since NIVSS is non-interactive, share verification algorithm can be computed by $\mathcal{A}$ without communicating with other stakeholders. If share verification is successful, then it would mean that the correct password was chosen while decrypting the c_c. Therefore, a DiSPP solution must *not* employ a NIVSS-and-OPRF-based PPSS solution.

Input: $\mathcal{D}, c_c, k_c$
Output: *password*
foreach $pwd' \in \mathcal{D}$ **do**
  $F_{k_c}(pwd') \leftarrow OPRF(pwd', k_c)$;
  ${s_c}'||{v_c}' \leftarrow Dec_{F_{k_c}(pwd')}(c_c)$;
  $b = NIVSSVerify({s_c}', {v_c}')$;
  if $b == 1$ **then**
    return pwd';
  end
end

Algorithm 1: Attack code on PPSS based on NIVSS-and-OPRF

5.2 How to encrypt secret shares

The particular attack above works because inside the ciphertext c_c, some extra information is stored that enables the adversary to verify the decryption password offline. To generalize the attack above, as long as the decryption of the ciphertext c_c may lead to some verifiable information, the solution would be insecure. For instance, in Shamir [39] secret sharing, a share consists of a share number i (essential to reconstruct the secret) and the secret share s_i. If one encrypts this information as $c_i \leftarrow Enc_{F_{k_i}(pwd)}(s_i||i)$, the above attack still works simply by decrypting with the passwords in the dictionary and checking whether or not a proper integer i is obtained.

Therefore, for SS-and-OPRF-based secret storage and retrieval to remain secure, it is important to only encrypt the random values, as observed by Acar et al. [2] and later supported by Demay et al. [18]. Since, in Shamir [39] secret sharing, the share number i is a public value that does not allow reconstruction of the secret by itself, it can be sent in clear, and only the random share value s_i is encrypted: $c_i \leftarrow Enc_{F_{k_i}(pwd)}(s_i)$ and each storage provider receives $share_i = (c_i, k_i, i)$ during secret storage, and sends c_i, i during secret retrieval.

5.3 Malware and Phishing Protection

We consider a strong phishing attack with man-in-the-middle between the user and the login server(s) during authentication (not registration). This means, the

user registered with a legitimate server with ls (e.g. ls = paypal.com), but now is trying to authenticate with an attacker with ls' (e.g. ls'=paypa1.com).

During the authentication protocol between the user and login server(s), a DiSPP should employ ls information as an input to the cryptographic function evaluation (e.g. MAC). For instance, while computing the response $resp$ during authentication, the ls is concatenated with $chal$ (see Figure 3) and used as part of server-side verification. Therefore, if a phishing man-in-the-middle attacker with ls' exists, the user's response will be on $chal||ls'$, whereas the real server would verify using $chal||ls$, making the attack impossible. Without the use of ls during authentication, such attacks would have been possible.

(A)PAKE settings already ensure security against man-in-the-middle attacks intrinsically.

6 Dispp instances

6.1 Existing DiSPP Instantiations

We discuss the DiSPP systems in the literature that provide provable security, employ a storage provider (e.g. mobile device or online storage provider) and ensure security with a *single password*. We investigate how these proposed systems fit in our DiSPP framework. Acar et al. [2] (with their patent application dating 2010 [4]), Bicakci et al. [8], Jarecki et al. [31], and İşler and Küpçü [26] are the known examples of DiSPP. These DiSPP systems are secure against offline dictionary attacks, even by the server, hackers obtaining the server database, or up to threshold-many storage providers that are corrupted by the same adversary. We compare the aforementioned DiSPP systems in Table 1. All the existing DiSPP solutions ([2,8,31]) except İşler and Küpçü [26] are built on an environment where there is one storage provider and one login server. İşler and Küpçü [26] proposed a DiSPP where there are n_{stor} online storage providers and one login server. We discuss an instantiation that generalizes these solutions in the next subsection.

Table 1. DiSPP solutions. TO: Threshold for online storage providers (t_{os}), TM: Threshold for personal devices (t_{mob}), TL: Threshold for login servers (t_{ls})

	TO	TM	TL
Acar et. al [2] Cloud Version	1	0	1
Acar et. al [2] Mobile Version	0	1	1
Bicakci et. al [8]	1	0	1
Jarecki et. al [31]	0	1	1
İşler and Küpçü [26]	t_{os}	0	1
Our instantiation	t_{os}	t_{mob}	t_{ls}

Acar et. al [2] proposed four DiSPPs in different settings with different concerns such as storage provider optimality, login server optimality, user anonymity,

and mobile device as a storage provider. For our framework, user's anonymity is not a major concern since revealing the identity of the user does not break the security captured in our model, and we assume the user identifiers (e.g. usernames) already leak this information. Below, we only discuss their server optimal DiSPP instance which employs digital signatures for registration and authentication, and unique blind signatures instead of OPRF.
Registration:

– The user and login server run the digital signature registration as in Figure 3(c) where the secret S is the signing key ssk.

Secret Storage:

– The user and the storage provider run the secret storage protocol as in Figure 4(a). Since there is only one storage provider ($t_{stor} = n_{stor} = 1$), the user does not run a secret sharing on S and sends only the encryption of the S to the storage provider.

Secret Retrieval:

– The user and the storage provider run the secret reconstruction as in Figure 4(b) where the user receives the secret S as an output.
Remark: As in the secret storage phase, the user does not run a secret sharing reconstruction. She computes one unique blind signature (as an OPRF) with the storage provider and performs one decryption locally to obtain the secret without further reconstruction.

Authentication:

– The user and login server run the digital signature authentication as shown in Figure 3(d).

Jarecki et. al [31] is an instance of DiSPP that employs the mobile device as storage provider. In Jarecki et. al [31], PAKE is embedded as server-side password protocol.
Registration:

– The user and the login server compute the registration of a PAKE protocol. The login server stores an OPRF evaluation of password $F_K(pwd)$ as in Figure 3(e).

Secret Storage:

– The user and mobile device compute the secret storage protocol as described in Figure 4(a).
Remark: In their setting, since there is only one storage provider, secret sharing is not employed, and the user sends the secret (which is the OPRF key K) as it is without an encryption ($share = K$).

Secret Retrieval: The user holding her password and the storage provider (mobile device) compute the secret retrieval solution as in Figure 4(b).
Remark: They only run the OPRF part, and no decryption or secret sharing is employed.
Authentication: The user holding the $F_K(pwd)$ retrieved from the secret retrieval runs PAKE with the login server as described in Figure 2.

Bicakci et. al [8] proposed a DiSPP (without a formal security proof), where they employed one login server and one storage provider. Their model is captured by our framework as follows;
Registration:

- The user generates a pair of keys (ssk, svk) for a unique blind signature based on (deterministic) RSA, where the high entropy secret is generated as $sig \leftarrow Sign_{ssk}(H(pwd||username))$ and the verification information is computed as $vInfo = H(sig||ls)$.

Secret Storage:

- The user and the storage provider compute the secret storage protocol as in Figure 4(a).
  Remark: Since there is only one storage, secret sharing is not computed as well as no encryption on the secret ($share = ssk$).

Secret Retrieval:

- The user runs secret retrieval as in Figure 4(b) where unique blind signature is employed instead as OPRF (recall both OPRF and unique blind signature provides the same functionality and security). The output to the user is $sig \leftarrow Sign_{ssk}(H(pwd||username))$.

Authentication:

- The user runs (non-interactive) hash-based authentication protocol with the login server as described in Section 4.1.

İşler and Küpçü [26] introduced the first DiSPP with a threshold on online storage providers, where in their system there are t_{stor} online storage providers and one login server. The solution by İşler and Küpçü [26] fits in our model as follows;
Registration:

- The user and login server run the hash-based registration as in Figure 3(g) where the secret S is k-bit entropy random string rnd.

Secret Storage:

- The user and the storage providers compute the secret storage protocol as in Figure 4(a).

Secret Retrieval:

– The user runs the secret retrieval as in Figure 4(b) with t_{stor} storage providers. At the end, the user obtains the secret, which is the k-bit entropy random string rnd.

Authentication:

– The user and login server compute hash-based non-interactive authentication as described in Section 4.1.

6.2 Another DiSPP Instantiation

Our framework enables us to derive other DiSPP instances not in the literature.
Registration:

– uses the hash-based registration as described in Figure 3(g) that outputs a high entropy random-string rnd as the secret S and verification information as $vInfo = H(rnd||ls)$.
Remark: To further strengthen login servers against offline dictionary attacks, rather than employing a single server, threshold password authenticated key exchange (TPAKE) can be employed so that an attacker is required to compromise *threshold-many* login servers in addition to *threshold-many* storage providers to mount a successful offline dictionary attack.

Secret Storage:

– employs ACSS-and-OPRF-based secret storage as described in Section 4.2.

Secret Retrieval:

– runs ACSS-and-OPRF-based secret retrieval protocol with an authorized set of storage providers to retrieve the secret $S = rnd$.

Authentication:

– runs hash-based authentication as shown in 3(h) using the reconstructed secret S during the secret retrieval.
Remark: Alternatively, one may employ non-interactive version or TPAKE as discussed above.

Note that while we do not provide a formal security proof of this instantiation, it can easily be seen that the definitions and proof theorems in [26] apply here with only minor modifications. They employ a TSS-and-OPRF-based PPSS solution, and prove security of their full protocol assuming that the adversary can corrupt at most threshold-many storage providers or fewer than threshold-many storage providers together with the login server. In our case, this can be easily converted to a proof where the adversary can corrupt any set of storage providers that are not authorized together with the login server, or some minimal set of authorized storage providers. Moreover, if TPAKE is employed during the authentication phase, the adversary needs to corrupt threshold-many

login servers rather than one. Lastly, if a challenge-response based interactive authentication is employed (or a TPAKE is employed), then the authentication protocol does not need a secure and server-authenticated channel as opposed to [26]. Intuitively, this instantiation is a secure DiSPP assuming that OPRF is secure, $\{KeyGen, Enc, Dec\}$ is a semantically secure encryption scheme, ACSS is a secure monotone access controlled secret sharing solution, and the hash function is picked from a collision resistant family.

6.3 Performance Evaluation

In this section, we first evaluate the performance of some building blocks, and then look at the performance of the DiSPP instance proposed above. We omit the evaluations of registration and authentication for the login servers, since schemes employed are run in constant time and with non-interactive hash-based registration and authentication, the login servers remain unmodified.

Performance measurements are processed on a standard laptop machine with Intel Core(TM) i7-5600U CPU 2.60 GHz, 8.00 GB RAM, 64-bit OS, and implemented in Java. For our implementation, we chose AES [17] for encryption, implemented OPRF in [31], TSS in [39], ACSS in [6], TOPRF in [30], and PPSS in [16] with various thresholds. Table 2 shows the local computations of the secret storage and retrieval phases at the user side. For the secret storage and registration, the storage providers and the login server *do not* compute anything; they only receive and store some values.

Table 2. Performance evaluation of DiSPP phases (in milliseconds).

User-Secret Storage			
	ACSS-OPRF [6, 31]	TOPRF [30]	PPSS [16]
2-5 Threshold	9.30	3.6	66.35
3-6 Threshold	11.88	4.6	66.60
5-10 Threshold	17.8	7.8	68.40
User-Secret Retrieval			
	ACSS-OPRF [6, 31]	TOPRF [30]	PPSS [16]
2-5 Threshold	5.22	4.0	952.18
3-6 Threshold	7.8	5.5	1273.12
5-10 Threshold	12.4	8.4	1603.21
Storage Provider-Secret Retrieval			
	ACSS-OPRF [6, 31]	TOPRF [30]	PPSS [16]
2-5 Threshold	2.1	3.3	807.15
3-6 Threshold	3.15	3.9	901.34
5-10 Threshold	5.2	6.5	1523.02

Finally, we show the performance evaluation of our proposed instantiation for the user and storage providers in Table 3. From the communication round perspective, the user can communicate with the storage providers in parallel, which decreases the network round trip to 1.5 rounds for secret retrieval and authentication in total.

Table 3. Performance evaluation of our DiSPP instantiation (in milliseconds).

	User (Reg.)	User (Auth.)	Storage Provider (Retrieval)	Login Total
2-5 Threshold	9.90	5.96	2.1	8.06
3-6 Threshold	12.48	8.56	3.15	11.71
5-10 Threshold	18.40	13.13	5.2	18.33

7 Conclusion

For the last decade, several single password protocols have been proposed [2, 8, 26, 31], where only [26] among them provides threshold security where an adversary is required to corrupt *threshold-many* storage providers to mount an offline dictionary attack on the user's password. For the first time in the literature, we provide a framework for *distributed* single password protocols that are secure against offline dictionary, man-in-the-middle, phishing, and honeypot attacks. We discussed correct and incorrect ways of using proper cryptographic building blocks, and concluded that use of non-interactive verifiable secret sharing schemes would result in an offline dictionary attack by a single storage provider. Our framework not only encompasses existing protocols, but also enables derivation of new DiSPP constructions. Finally, we implemented several building blocks and measured performance of our proposed DiSPP instance.

8 Acknowledgment

We acknowledge the support of TÜBİTAK (the Scientific and Technological Research Council of Turkey) under the project number 115E766, and the Royal Society of UK Newton Advanced Fellowship NA140464.

References

1. M. Abdalla, M. Cornejo, A. Nitulescu, and D. Pointcheval. Robust password-protected secret sharing. In *European Symposium on Research in Computer Security*. Springer, 2016.
2. T. Acar, M. Belenkiy, and A. Küpçü. Single password authentication. *Computer Networks*, 2013.

3. A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. Password-protected secret sharing. In *Proceedings of the 18th ACM conference on Computer and Communications Security*. ACM, 2011.
4. M. Belenkiy, T. Acar, H. Morales, and A. Küpçü. Securing passwords against dictionary attacks. 2011. US Patent 9,015,489.
5. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on*. IEEE, 1992.
6. J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *Proceedings on Advances in cryptology*. Springer., 1990.
7. M. Bertilsson and I. Ingemarsson. A construction of practical secret sharing schemes using linear block codes. In *International Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 1992.
8. K. Bicakci, N. B. Atalay, M. Yuceel, and P. C. van Oorschot. Exploration and field study of a browser-based password manager using icon-based passwords. In *Workshop on Real-Life Cryptographic Protocols and Standardization*, 2011.
9. G. R. Blakley et al. Safeguarding cryptographic keys. In *Proceedings of the national computer conference*, 1979.
10. A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *International Workshop on Public Key Cryptography*. Springer, 2003.
11. X. Boyen. Hidden credential retrieval from a reusable password. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*. ACM, 2009.
12. X. Boyen. HPAKE: Password authentication secure against cross-site user impersonation. In *Cryptology and Network Security*. Springer, 2009.
13. V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *EUROCRYPT 2000*. Springer, 2000.
14. E. F. Brickell. Some ideal secret sharing schemes. In *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1989.
15. J. Camenisch, R. R. Enderlein, and G. Neven. Two-server password-authenticated secret sharing uc-secure against transient corruptions. PKC 2015, 2015.
16. J. Camenisch, A. Lehmann, A. Lysyanskaya, and G. Neven. Memento: How to reconstruct your secrets from a single password in a hostile environment. In *CRYPTO 2014*. Springer, 2014.
17. J. Daemen and V. Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.
18. G. Demay, P. Gazi, U. Maurer, and B. Tackmann. Per-session security: Password-based cryptography revisited. In *ESORICS*, 2017.
19. O. Farràs, J. Martí-Farré, and C. Padró. Ideal multipartite secret sharing schemes. *Journal of cryptology*, 2012.
20. U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of cryptology*, 1988.
21. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO'86*, 1986.
22. D. Florencio and C. Herley. A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007.
23. W. Ford and B. S. Kaliski. Server-assisted generation of a strong secret from a password. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000.(WET ICE 2000)*. IEEE, 2000.

24. M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference*. Springer, 2005.
25. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 1988.
26. D. İşler and A. Küpçü. Threshold single password authentication. In *ESORICS DPM'17*. Springer, 2017.
27. M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 1989.
28. S. Jarecki, A. Kiayias, and H. Krawczyk. Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model. ASIACRYPT'14, 2014.
29. S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online).
30. S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. Toppss: Cost-minimal password-protected secret sharing based on threshold oprf. In *International Conference on Applied Cryptography and Network Security*. Springer, 2017.
31. S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena. Device-enhanced password protocols with optimal online-offline protection. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM, 2016.
32. S. Jarecki, H. Krawczyk, and J. Xu. Opaque: An asymmetric pake protocol secure against pre-computation attacks. In *EUROCRYPT 2018*, 2018.
33. J. Katz and Y. Lindell. *Introduction to modern cryptography*. CRC press, 2014.
34. J. Katz, P. MacKenzie, G. Taban, and V. Gligor. Two-server password-only authenticated key exchange. In *International Conference on Applied Cryptography and Network Security*, 2005.
35. J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT 2001*. Springer, 2001.
36. P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. In *CRYPTO 2002*. Springer, 2002.
37. T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*. Springer, 1991.
38. C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 1991.
39. A. Shamir. How to share a secret. *Communications of the ACM*, 1979.
40. E. I. Tatli. Cracking more password hashes with patterns. *Information Forensics and Security, IEEE Transactions on*, 2015.
41. J. M. Turner. The keyed-hash message authentication code (hmac). *Federal Information Processing Standards Publication*, 2008.