Threshold Single Password Authentication

Devriş İşler and Alptekin Küpçü

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

Abstract. Passwords are the most widely used form of online user authentication. In a traditional setup, the user, who has a *human-memorable low entropy password*, wants to authenticate with a login server. Unfortunately, existing solutions in this setting are either non-portable or insecure against many attacks, including phishing, man-in-the-middle, honeypot, and offline dictionary attacks. Three previous studies (Acar et al. 2013, Bicakci et al. 2011, and Jarecki et al. 2016) provide solutions secure against offline dictionary attacks by additionally employing a *storage provider* (either a cloud storage or a mobile device for portability). These works provide solutions where offline dictionary attacks are impossible as long as the adversary does not corrupt *both* the login server and the storage provider.
For the first time, improving these previous works, we provide a more secure generalized solution employing *multiple* storage providers, where our solution is proven secure against offline dictionary attacks as long as the adversary does not corrupt the login server and *threshold-many* storage providers. We define ideal and real world indistinguishability for threshold single password authentication (Threshold SPA) schemes, and formally prove security of our solution via ideal-real simulation. Our solution provides security against all the above-mentioned attacks, including *phishing, man-in-the-middle, honeypot, and offline dictionary attacks*, and requires *no* change on the server side. Thus, our solution can immediately be deployed via a browser extension (or a mobile application) and support from some storage providers. We further argue that our protocol is efficient and scalable, and provide performance numbers where the user and storage load are only a few milliseconds.

Keywords: Password based authentication, threshold secret sharing, dictionary attack, phishing

1 Introduction

Passwords are the most widely used form of online user authentication. In a traditional password based authentication, there are two parties: the *user* who has a *human-memorable low entropy password* and the *login server* that creates an account for the user and keeps the user's account information (e.g. $<$*username*,

hash(password)>) to authenticate the user later when she wants to login. Another related field includes password authenticated key exchange (PAKE) protocols, where a user and a server desire to establish a *secure and authenticated channel* via a shared secret password [3, 6, 7, 19, 20]. Passwords are also employed in password-protected secret sharing (PPSS) techniques, where the user stores her credential(s) on a server or among multiple servers [5, 8, 9, 16]. The server(s) verify that the user is legitimate before giving an access to the stored secret, and this authentication is done via passwords.

The existence of servers' adversarial behaviors such as *phishing*, *man-in-the-middle*, and *honeypot* attacks, where the adversarial server tries to trick the user to willingly reveal her password (unaware of the attack), as well as *offline dictionary attacks* that can be mounted by the server or hackers obtaining the server database are commonly known and powerful attacks on users' passwords. Unfortunately, all the above-cited constructions, where the server stores the user's password as plain text or the hash of the password, lose security against such attacks. Indeed, such attacks are very prevalent, and recent studies even propose improved offline dictionary attacks [24]. The damage of the successful attack is increased dramatically if the user reuses the same password to register with more than one login server, which is a common practice [13].

Our focus in this paper is on password based authentication secure against those attacks. Consider the following approaches:

- ***Traditional insecure approach*:** Store the output of a deterministic (hash) function of the password at the login server. The user's password is easily compromised in the current traditional approach against adversarial servers (e.g. phishing, man-in-the-middle, honeypot). Moreover, it is directly vulnerable to offline dictionary attacks by hackers obtaining the login server database, since recomputing the output of the deterministic function and comparing against the database enables such offline dictionary attacks.
- ***Secure but non-portable approach*:** Store a verification key at the login server, where the corresponding secret key is blinded by the user's password and stored on the user's machine or USB device. When the keys are generated independent of the password, such an approach will protect the user against the mentioned attacks. Unfortunately, this approach is not user friendly in the contemporary setting where each user owns multiple devices or employs public terminals, as this incurs portability issues (even with USB storage employed, note that not all devices have USB ports). [1]
- ***Secure and usable approach*:** Store the verification key at the login server and securely store the corresponding secret key blinded by some function of user password at storage provider(s) *different* from the login server. Such a *storage provider* can be a cloud storage or mobile device (as opposed to a non-portable location above). This was first observed and the first solution was constructed by Acar et al. [1] (with their patent application dating 2010

[1] Non-cloud-based password managers also fall into this setting.

[2]), and later also used by Jarecki et al. [17] and also Bicakci et al. [4].[2] We make the following observations regarding these three works: Moreover, there are two useful observations we have investigated in terms of server side changes;

1. **Secret is a salt:** Salt is a well-known approach to slow offline dictionary attacks. Instead of server-side salt, consider that the *user* generates a random salt (that is of the size of the security parameter) per login server during the registration (Bicakci et al. [4], PwdHash [22]). Then, the user computes a hash of *password* and *salt* as $hash(salt||password)$ and sends the hash output to the login server together with username. This salt needs to be securely binded to the user password with the help of storage provider(s). This approach requires no change at the login server since the hash output can be considered as the user's server-specific password.
2. **Secret is a key of an authentication scheme:** The user generates a key depending on the authentication scheme (e.g. digital signature, message authentication code) per login server. The user sends the verification key of the authentication scheme to the login server together with the username for registration, and stores the secret key at the storage provider(s) blinded with the password. This approach requires changes on the login server side to verify a signature or MAC.

In both above observations, the user should generate a random secret that is bounded to the verification key sent to the login server, and securely store it at the storage provider.

The only known provably secure (against offline dictionary attacks) and usable password based authentication or PAKE systems are the Acar et al. [1] (with their patent application dating 2010 [2]) and Jarecki et al. [17] solutions (Bicakci et al. [4] present a solution idea briefly, but without a formal security proof). They include a single storage provider (which is either a cloud storage or a mobile device) for the secret storage. The underlying assumption is that the protocol is secure unless the login server and the storage provider are corrupted by the same adversary. Thus, when the same adversary corrupts both the login server and the storage provider, or they simply collude, they can mount an offline dictionary attack to find the user password. Any weaker adversary can always perform an *online attack*; however, online attacks are not big threats due to several reasons. Firstly, they are inherently many orders of magnitude slower compared to offline attacks due to network delays. Secondly, the honest login servers and storage providers will block the adversary after several unsuccessful attempts, or limit the rate of such attempts.

For the first time, we present a **single password authentication (SPA)** protocol that can employ possibly *more than one* storage provider (any combination of cloud or mobile devices). In our **Threshold SPA** solution, we employ a total of n storage providers, and a threshold $1 \leq t \leq n$. This setting serves

[2] Cloud-based password managers are related to this setting, but almost all of them employ the user master password for authenticating with their own servers, hence having the same insecurities there.

two purposes. Firstly, for an adversary to be able to successfully mount an offline dictionary attack, he must corrupt the login server *in addition to* t storage providers. Secondly, to login, the user must access t storage providers out of n; thus availability can be balanced against security easily by setting these parameters. While the underlying techniques are different, in terms of security, the previous solutions correspond to setting $t = n = 1$.

Delving deeper, Acar et al. [1] discuss in one paragraph that it is possible to convert their solution to employ more than one storage provider using secret sharing. Unfortunately, when done in a straightforward manner, this results in an insecure solution in the sense that corrupting the login server and *only one* storage provider will still enable the adversary to perform an offline dictionary attack. As mentioned earlier, they employ the storage provider to store the secret key needed for authentication. This secret key is protected by the password. When this secret is simply shared among multiple storage providers, there needs to be a mechanism to prevent the login server to obtain these shares from the storage providers, as otherwise the adversarial login server can simply request these shares and perform an offline dictionary attack. To prevent this, only the entity knowing the password should be able to retrieve these shares, meaning that there needs to be another password based authentication mechanism also done at the storage provider side, creating a recursive problem, needing further storage providers indefinitely. If traditional password based authentication approaches are employed instead, then this password based authentication subprotocol will enable the login server to collude with only one storage provider to mount the offline dictionary attack, since the user only has a single password and hence the same authentication is employed at every storage provider. Moreover, their one paragraph extension idea lacks any formal proof where we give a formal proof for Threshold SPA for the first time.

In our solution, while the user still has only a single password, **our protocol provably ensures that the following adversaries will be completely unsuccessful**:

- An adversary that controls *threshold-many* the storage providers (but not the login server).
- An adversary that controls less than threshold-many storage providers *in addition to* the login server.
- An adversary who successfully mounts a phishing, man-in-the-middle, or honeypot attack.

We assume that if any party is corrupted by an adversary once, it remains corrupted (which is realistic, since the adversary can install backdoors on a system once corrupted). Our solution only fails against a local adversary (such as a keylogger on the user's computer), since it can sniff the password. Our **contributions** are as follows:

1. We **define** a general structure of threshold single password authentication (Threshold SPA) systems and propose the **first construction** with a formal proof.

2. We formally present **ideal and real world** definitions for security of Threshold SPA protocols, for the first time in the literature.
3. We **formally prove** the security of our Threshold SPA solution via ideal-real simulation, showing impossibility of **offline dictionary attacks**.
4. Our Threshold SPA method 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** evaluation numerically, showing that our techniques are easily applicable with today's hardware.
6. Our construction does **not require any change** at the login server side and can work with a variety of storage providers (e.g. mobile devices and cloud providers).

Overview: Our solution achieves full threshold security as follows: We create a salt as the secret *independent* of the password *pwd* and compute verification information as the hash of salt, password, and login server domain name *ls* ($Hash(salt||pwd||ls)$). Then, the verification information is shared with the login server whereas the salt is secret shared. But, these shares are not directly sent to the storage providers. We employ a layer of encryption to hide each share, and this encryption employs an oblivious pseudorandom function (OPRF) F output over the password *pwd* as its key. Moreover, each storage provider will use a different key k_i for the OPRF, and thus each share will be encrypted with the corresponding $F_{k_i}(pwd)$ as the key. This means, the storage providers also hold essentially no information regarding the password. This is true because as first observed by Acar et al. [1] (for this discussion simply assume one time pad is employed for encryption, though they discuss in further detail how a block cipher can be employed), when a random value (such as the secret key or shares) is encrypted with the password or any deterministic value derived from the password (such as hash or OPRF over the password), an offline dictionary attack is impossible without knowing what the decryption needs to reveal since any password in the dictionary would yield a valid plaintext.

Later, for authentication, the user interacts with threshold-many servers using the OPRF protocol, reconstructs the original secret salt after decryption with her correct password, and then computes the verification information as $Hash(salt||pwd||ls)$ and sends it along with her username to the login server. Only when the login server and at least threshold-many storage providers collude they can reconstruct the secret salt by trying different passwords, and calculate the hashes offline. Otherwise, offline dictionary attacks are impossible. Consider, for example, the adversarial login server potentially colluding with $t-1$ storage providers. He needs to interact online through the OPRF protocol with at least one honest storage provider to be able to reconstruct the secret salt, which can be rate/attempt limited. Moreover, consider that threshold-many storage providers are colluding while the login server is honest. Even though they can reconstruct a secret salt, since all passwords yield to a valid (in terms of format) secret salt, they can only try to authenticate online with the login server to verify whether

or not the secret salt they constructed is the correct one. Again, this can easily be rate/attempt limited.

2 Related Work

Traditional password-based authentication takes place between two parties (a *user* and a *login server*). However, Boyen [5] showed that any password-based authentication between a user and a login server is vulnerable to an *offline dictionary attack* by the login server (or hackers obtaining its database). Tatlı [24] improved offline dictionary attacks on password hashes to find some additional passwords assumed to be strong and complex.

Ford et al. [14] suggest a *password hardening* protocol where the user, holding a weak-password *pwd*, interacts with one or more servers by blinding the password to create a secret credential (to decrypt, or authenticate herself to a login server, etc.) from shares received by the hardening server(s) (which is like storage providers). The hardening server(s) cannot learn anything about the password and the secret unless all of them collude. During the authentication, for each login server, the user runs the password hardening protocol to retrieve the same secret as in the registration by communicating with hardening servers. The solution proposed do not have a formal proof and requires interaction with all of the servers to be able to reconstruct the secret. MacKenzie et al. [20] propose a threshold PAKE where the password is secure unless threshold-many servers collude. [20] requires servers to know each other. [18] proposes to create a password file storing false passwords called *honeywords* per user account. In case the adversary steals the password file, and mistakenly employs a honeyword, the system is alerted.

Mannan et al. [21] propose to secure user's password from untrusted user computer (malicious browser) assuming the server holds the user password. Camenish et al. [10] distribute the password verification over multiple servers to secure the password against server compromise where the server keeps the hash of username and password $hash(username||password)$. PwdHash [22] produce a different password for each login server by simply computing the hash of the password and login server domain where the hash is a pseuodorandom function of the domain keyed with the user password $H(password, domain) = PRF_{password}(domain)$ and the server stores the hash value. The discussed solutions do not provide security against offline dictionary attack in case the server database is compromised. Increasing the number of parties by adding storage provider(s) is one way to help prevent offline dictionary attacks.

Acar et al. [1] (with their patent application dating 2010 [2]) present the first provably secure single password authentication protocols where the user employs a cloud or mobile storage provider to keep her secret to prevent offline dictionary attacks. The user's password is secure against offline dictionary attacks unless the storage provider and the server are colluding. Their mobile device based solution inputs the password to the device, and hence provides security against *malware* on the public terminal. They provide security against *phishing* indirectly because

the user identifier used at the storage provider depends on the server name. Since the phishing site name is *different* from the actual login server name, the retrieval of the user secret fails. Our main differences are to enable a fully secure threshold construction for the first time, without requiring any changes at the login server, making our solution much easier to deploy.

Following Acar et al. [1], Jarecki et al. [17] provided a device enhanced password authenticated key exchange protocol employing a mobile device storage. Similar to [1], their protocol is secure against offline dictionary attacks assuming the login server and the mobile device are not colluding. They provide a recovery procedure in case the device is lost. They leave threshold authentication as future work, which is what we achieve.

Bicakci et al. [4] discuss briefly a single password solution employing a storage provider for unique blind signatures (similar to Acar et al. [1]), but they neither delve into the details of their solution nor present a security proof. Nevertheless, we are influenced by their work in terms of requiring no change at the login server, and achieve to distribute the storage provider for the first time.

3 Preliminaries

Let $\lambda \in N$ be 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 $\lambda' \in N$ such that $\forall \lambda > \lambda'$ $negl(\lambda) < 1/poly(\lambda)$.

Hash Function: A hash function H 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 psuedorandom function (PRF) F is a function that takes two inputs: a secret function 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 distinguishers given oracle access. An Oblivious PRF (OPRF) [15] is a protocol between two parties (*sender* and *receiver*) that securely computes $F_k(x)$ where the k and x are the inputs of *sender* and *receiver*, respectively, such that the sender learns *nothing* from the interaction and the receiver learns $F_k(x)$.

Symmetric Encryption Scheme: 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. The decryption algorithm $Dec_{sk}(c)$ uses the secret sk to decrypt the ciphertext c, and outputs the original message msg. The encryption scheme we use need to be semantically secure.

Threshold Secret Sharing (TSS): consists of two PPT algorithms $< s_1, ..., s_n > \leftarrow TSS(S)$ to create the n shares of the secret S, and for a threshold t we use $S \leftarrow TSSRecon(s_1, ..., s_t)$ to reconstruct the original secret. We employ

the methodology of Shamir [23]. The security is that less than threshold many shares provide theoretically no information regarding the original secret.

4 Threshold Single Password Authentication

In a Threshold SPA protocol, 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 authenticate with these login servers. For this purpose, the users store some secret information (that is needed for authentication with the login servers) at one or more ***storage providers***. The main objective of a Threshold SPA solution is to protect the user's password against offline dictionary attacks by the storage providers, the login servers, and many other adversaries (including phishing sites). Figure 1 shows an overview of the **registration** and **authentication** phases of a Threshold SPA protocol, considering a single user who registers with a login server and stores the secret at n storage providers. Threshold in this context refers to the fact that the user must communicate with some subset (defined by the threshold) of storage providers to facilitate authentication with the login server. It furthermore refers to the security of the solution: An offline dictionary attack is possible only when the adversary controls the login server *and* at least threshold many storage providers.

The **registration** phase is for the user to register with the login server and store the secret among storage providers. The user registers with the login server whose domain is ls using a low-entropy password pwd (only secure against online attacks). The login server obtains the user's verification information vk and identifier $userID$ such that the login server can authenticate the legitimate user whenever the user wants to login. The user further stores some secret information $share_i$ with the storage providers, in a distributed manner. Some identifier $storUID_i$ is associated with this secret to facilitate later retrieval. More formally we have the following multi-party protocol:

Registration:

1. **The user's inputs** are a user name $userID$ for the login server whose domain is ls, and a password pwd.
2. **Each storage provider receives as output** an identifier $storUID_i$ and a share $share_i$ and stores the data received in the database. This share is what the user wants to store among the storage providers depending on the Threshold SPA protocol. The identifier is employed for later retrieval of the stored share.
3. **The login server receives as output** an identifier $userID$ and a server verification information based on user's password vk of the user, and stores them in his database. The verification information vk is used by the login server to verify the user during the authentication phase.

The **authentication** phase is for the user who remembers the user name $userID$ and the password pwd to authenticate herself to the login server with domain ls by interacting with *threshold-many* (t) storage providers to retrieve

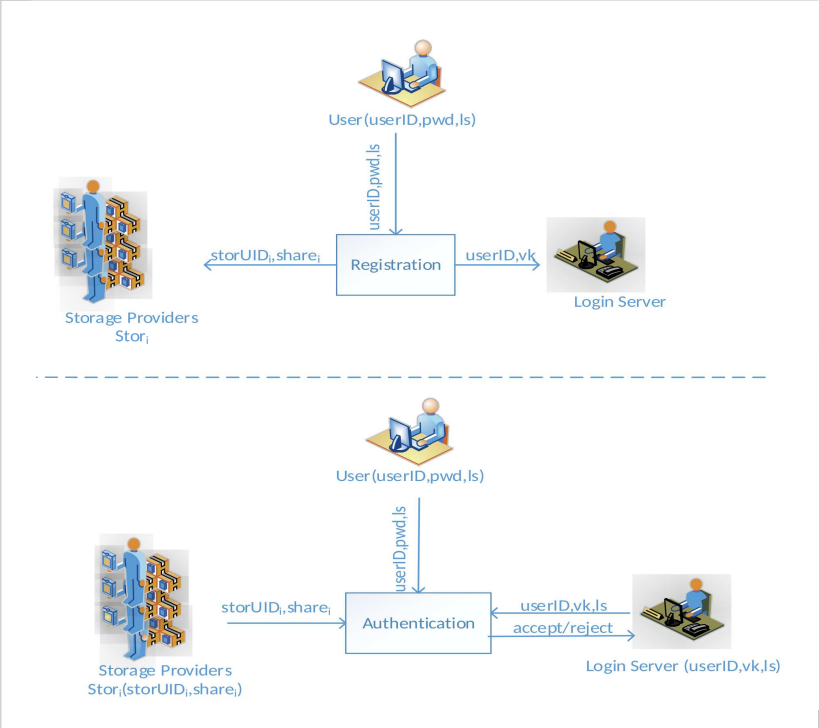

Fig. 1. Threshold-SPA Overview. The registration and authentication protocols are separated by the dashed line.

and reconstruct the secret needed for authentication. Of course, in general it is possible that $t = n$ and hence all storage providers may need to be contacted.

Authentication:

1. **The user's inputs** are as before: the user name *userID*, the password *pwd*, and the domain ls of the login server to authenticate with.
2. **The login server's inputs** include the user identifier *userID*, as well as the verification information *vk* corresponding to the user, and its domain ls.
3. **Each storage provider's inputs** are the share $share_i$ that they hold for that user and the identifier $storUID_i$ of that user.
4. **The login server outputs** accept or reject. The domain name ls is employed to prevent phishing/man-in-the-middle attacks.

4.1 Security Definition

We define the *ideal world* and the *real world* for a Threshold SPA protocol, in the spirit of Canetti [11].

Ideal World: The ideal world consists of a user $\mathcal{U}$, a login server $\mathcal{LS}$, *n-many* storage providers $\mathcal{SP} = (Stor_1, Stor_2, \ldots, Stor_n)$ (realize that $\mathcal{SP}$ denotes the

set of storage providers), and the universal trusted party $\mathcal{TP}$ **(which is not a real entity, and only exists in the ideal world)**.

Registration :

1. $\mathcal{U}$ sends $< userID, pwd >$ to $\mathcal{TP}$.
2. $\mathcal{TP}$ computes the necessary steps to obtain the shares $share_i$ and identifiers $storUID_i$, and the verification information vk.
3. $\mathcal{TP}$ sends $< userID, vk >$ to $\mathcal{LS}$ and $< storUID_i, share_i >$ to each storage provider in $\mathcal{SP}$.

Authentication:

1. $\mathcal{U}$ sends $< userID, pwd >$ to $\mathcal{TP}$.
2. $\mathcal{TP}$ sends $userID$ to $\mathcal{LS}$ for login request.
3. $\mathcal{TP}$ sends $storUID_i$ to *at least threshold-many* storage providers in $\mathcal{SP}$ for retrieving the secret shares (wlog. assume all storage providers are employed).
4. $\mathcal{SP}$ send their shares $share = \{share_1, share_2, \ldots, share_t\}$.
5. $\mathcal{TP}$ calculates the verification information vk using the shares from the $\mathcal{SP}$ and the pwd from $\mathcal{U}$, and sends vk to $\mathcal{LS}$.

Real World: The real world consists of a user $\mathcal{U}$, a login server $\mathcal{LS}$, and storage providers $\mathcal{SP} = (Stor_1, Stor_2, \ldots, Stor_n)$. There is no universal trusted party $\mathcal{TP}$ for a real world protocol π for the threshold-single password authentication. The parties $\mathcal{U}$, $\mathcal{LS}$, and $\mathcal{SP}$ are involved in the real execution of the protocol π.

Definition 1 (Secure Threshold Single Password Authentication). *Let π be a probabilistic polynomial time (PPT) protocol for a threshold single password authentication. We say that π is secure if for every non-uniform PPT real world adversary $\mathcal{A}$ attacking π, there exists a non-uniform PPT ideal world simulator $\mathcal{S}$ such that for both registration and authentication phases, the real and ideal world interactions and outputs are computationally indistinguishable;*

$$\{IDEAL_{\mathcal{S}(aux)}(userID, pwd, ls, \lambda)\} \equiv_c \{REAL_{\pi,\mathcal{A}(aux)}(userID, pwd, ls, \lambda)\}$$

where $aux \in \{0,1\}^*$ denotes the auxiliary input, and λ is the security parameter.

Note that such an ideal world definition assumes secure and authenticated channels between parties. Furthermore, as there is only a single login server in the ideal world, it does not include phishing (this is why ls domain is not part of the ideal world). But it provides security against offline dictionary attacks. In Section 5.2 we discuss the security of our solution for attacks like phishing not covered by this ideal model definition.

5 Threshold SPA Construction

Our Threshold SPA construction is represented visually in Figure 2 (*registration phase*) and Figure 3 (*authentication phase*). It is also described below.

Registration:

1. **The user**
   (a) generates a random $salt$ as $salt \leftarrow \{0,1\}^\lambda$

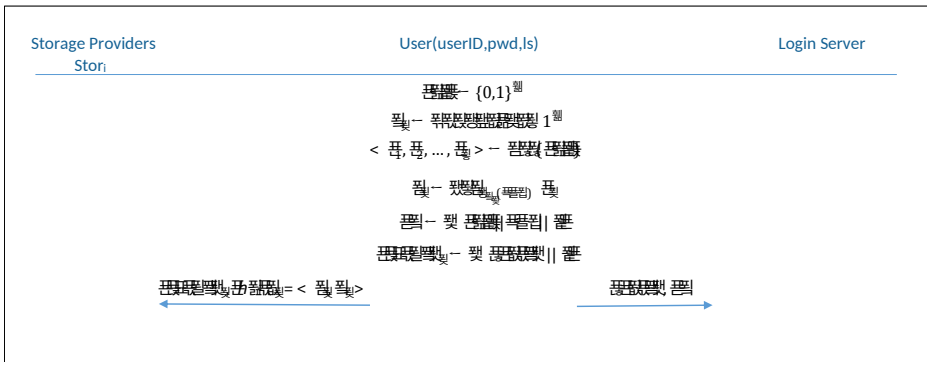

Fig. 2. Threshold SPA Construction Registration Phase

(b) generates one $OPRF$ key per storage provider as $k_i \leftarrow OPRFKeyGen(1^\lambda)$.
(c) runs threshold secret sharing construction scheme on $salt$ to create the secret share for each storage provider $< s_1, s_2, ..., s_n > \leftarrow TSS(salt)$.
(d) encrypts each share using *oblivious pseudorandom function* of the password pwd using generated $OPRF$ key of each storage provider obtaining $c_i \leftarrow Enc_{F_{k_i}(pwd)}(s_i)$.
Remark: Since the secret shares are random bitstrings, offline dictionary attacks on these encryptions are impossible. Therefore, in our solution, even all the storage providers,without the help of the login server, they cannot break the security.
(e) computes verification information for the login server via a collusion-resistant hash function as $vk = H(salt||pwd||ls)$
Remark: Salt is a randomstring with size of security parameter. For that reason, the login server, without colluding at least t-*many* storage provider, cannot perform a successful dictionary attack.
(f) computes the same identifier for all storage providers via a collusion-resistant hash function as $storUID_i \leftarrow H(userID||ls)$.
Remark: This identifier is only used to retrieve the correct values from the storage providers that serve multiple clients. Remember that ls is the domain name of the server the user is registering/connected to (e.g. ls=paypal.com).
(g) sends $< userID, vk = H(salt||pwd||ls) >$ to the login server, and $< storUID_i, share_i = (c_i, k_i) >$ to each storage provider.
(h) can forget all the data she computed that are cumbersome for her to remember (e.g. K, $k_{i=1,..,n}$).

2. **The login server** receives $< userID, vk = H(salt||pwd||ls) >$, and stores the pair in his database.
3. **Each storage provider** receives $< storUID_i, share_i = (c_i, k_i) >$ and stores in the database.

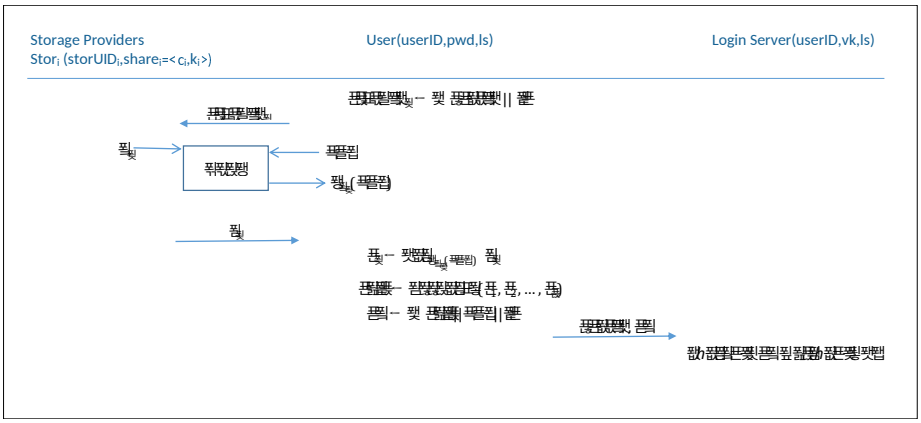

Fig. 3. Threshold SPA Construction Authentication Phase

Authentication:

1. **The user** who is trying to authenticate with the login server with domain ls computes the same storage identifier $storUID_i \leftarrow H(userID||ls)$ and sends it to at least *t-many* storage providers, and sends $userID$ to login server.
2. **Each storage provider** finds the associated $< share_i = (k_i, c_i) >$ with $storUID_i$.
3. **The user and each storage provider jointly** execute the oblivious pseudorandom function (OPRF) protocol. Each storage provider acts 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.
4. **Each storage provider** sends c_i to the user.
5. **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 $salt \leftarrow TSSRecon(s_1, s_2, ..., s_t)$.
   Remark: Even when at least threshold-many storage providers collude and reconstruct the ciphertext *salt* of the original secret salt by trying different passwords in the dictionary, they still need to try the resulting salts *online* against the login server, since each password in the dictionary would result in a valid *salt* when decrypting shares S.
6. **The user** computes the verification information as $vk = H(salt||pwd||ls)$ and sends $< userID, vk = H(salt||pwd||ls) >$ to the login server.
7. **The login server** looks up the verification information vk associated with *userID*, and it accepts the response if and only if the vk sent by the user same as the vk in the database.[3]

[3] Or a hashed version of vk can be stored in the database, as usual.

Remark: The domain name of the login server ls in the hash is to prevent a phishing/man-in-the-middle attacks. This attack prevention is discussed in Section 5.2 in details.

5.1 Security Proof

Theorem 1. *Our Threshold SPA protocol is secure according to Definition 1 against any non-uniform PPT adversary $\mathcal{A}$ corrupting* ***the login server $\mathcal{LS}$ and (t-1) many storage providers*** *$\mathcal{SP}_c$, assuming that the threshold secret sharing construction is secure, encryption scheme is semantically-secure, the oblivious pseudorandom function is secure, and the hash function is collision resistant.*

Proof. The simulator $\mathcal{S}$ simulates honest parties in the real world (which are the user $\mathcal{U}$ and *n-t+1* storage providers denoted by $\mathcal{SP}_h = \{Stor_{i_h}\}$ where $i_h = t, ..., n$ wlog. since all storage providers in our solution are identical) and corrupted parties in the ideal world (which are the login server $\mathcal{LS}$ and *t-1* storage providers denoted by $\mathcal{SP}_c = \{Stor_{i_c}\}$ where $i_c = 1, ..., t-1$). $\mathcal{S}$ behaves as follows:

Registration Phase:

1. The user $\mathcal{U}$ in the ideal world sends <*userID,pwd*> to $\mathcal{TP}$
2. The third party $\mathcal{TP}$:
   (a) generates a random *salt* as $salt \leftarrow \{0,1\}^\lambda$
   (b) generates one *OPRF* key per storage provider as $k_i \leftarrow OPRFKeyGen(1^\lambda)$.
   (c) runs threshold secret sharing construction scheme on *salt* to create the secret share for each storage provider $< s_1, s_2, ..., s_n > \leftarrow TSS(salt)$.
   (d) encrypts each share using *oblivious pseudorandom function* of the password *pwd* using generated *OPRF* key of each storage provider obtaining $c_i \leftarrow Enc_{F_{k_i}(pwd)}(s_i)$.
   (e) computes verification information for the login server via a collusion-resistant hash function as $vk = H(salt||pwd||ls)$
   (f) computes the same identifier for all storage providers via a collusion-resistant hash function as $storUID_i \leftarrow H(userID||ls)$.
   (g) sends $< userID, vk = H(salt||pwd||ls) >$ to the login server (which is the simulator $\mathcal{S}$), and $< storUID_i, share_i = (c_i, k_i) >$ to each storage provider.
3. $\mathcal{S}$ receives $< userID, vk = H(salt||pwd||ls), \{storUID_i, share_i = (c_i, k_i)\}_{i=1,...,t-1} >$ from $\mathcal{TP}$.
   Remark: Since $\mathcal{S}$ simulates $\mathcal{LS}$ and $\mathcal{SP}_c = \{Stor_{i_c}\}_{i_c=1,...,t-1}$ in the ideal world, $\mathcal{S}$ receives whatever they receive from $\mathcal{TP}$. Because of the symmetry of the actions of the storage providers in our construction, which ones are corrupted by the adversary does not change anything in the proof as long as the number of corrupted storage providers is below the threshold.
4. $\mathcal{S}$ sends $< userID, vk = H(salt||pwd||ls) >$ to the adversarial $\mathcal{LS}$ in the *real world*.

5. $\mathcal{S}$ follows the protocol as a user choosing a random password pwd' from the dictionary and a secret share s_{i_c}' for each corrupted storage provider, and sends $< storUID_{i_c}, share_{i_c} = (c_{i_c}', k_{i_c}) >$ where $c_{i_c}' = Enc_{F_{k_i}(pwd')}(s_{i_c}')$ to each adversarial storage provider $\{Stor_{i_c}\}_{i_c=t,...,n}$.
   Remark: Adversarial storage providers receive encrypted shares of random values with the random password pwd'. There is no efficient way for adversarial storage providers to distinguish this from real behavior since one more storage provider needs to be corrupted to mount a successful offline dictionary attack. For our protocol, all $storUID_i$ values are the same.
6. $\mathcal{S}$ stores all the data in its database.

Authentication Phase:

1. **The user** who is trying to authenticate with the login server with domain ls in the ideal world computes the same storage identifier $storUID_i \leftarrow H(userID||ls)$ and sends it to at least *t-many* storage providers, and sends $userID$ to login server.
2. $\mathcal{S}$ receives $< \{storUID_i\}_{i=1,...,t-1} >$ from $\mathcal{TP}$.
   Remark: In general, since $\mathcal{TP}$ may pick any (threshold size) subset of storage providers to work with, and so not all adversarial storage providers may need to be contacted. We are assuming the most powerful adversary here, therefore suppose that all adversarial storage providers are contacted.
3. $\mathcal{S}$ sends $storUID_{i_c}$ to each storage provider $Stor_{i_c}$ where $i_c = 1, ..., t-1$.
   Remark: While $\mathcal{S}$ could already contact the $\mathcal{TP}$ regarding the storage providers at this point (since it already possesses the necessary shares), this may be distinguishable by the adversary. It is possible that the adversarial storage providers will not provide correct values in the real world, and hence the real authentication may fail. The simulator must ensure in that case that the ideal authentication also fails. The following steps are hence necessary for indistinguishability.
4. $\mathcal{S}$ executes the OPRF protocol with each $\{Stor_{i_c}\}_{i_c=1,...,t-1}$ using the password pwd', and receives $p_{i_c} = F_{k_{i_c}}(pwd')$ and also c_{i_c} from each real $Stor_{i_c}$.
5. $\mathcal{S}$ checks whether or not each $\{Stor_{i_c}\}_{i_c=1,...,t-1}$ used the correct corresponding $share_{i_c} = (c_{i_c}, k_{i_c})$ values. $\mathcal{S}$ already possesses the correct values obtained from $\mathcal{TP}$ during registration in the database. For each p_{i_c}, c_{i_c} received from $Stor_{i_c}$, $\mathcal{S}$ does the following: Using the corresponding c_{i_c}', k_i stored in its database during registration, it computes $p_i = F_{k_i}(pwd')$ locally and checks whether or not $p_{i_c} = p_i$ and $c_{i_c} = c_{i_c}'$. There are two cases for each $Stor_{i_c}$:
   (a) **Case 1: Correct** $share_{i_c} = (c_{i_c}, k_{i_c})$ **employed by the adversary in the real protocol**. $\mathcal{S}$ detects this by verifying that $p_{i_c} = p_i$ and $c_{i_c} = c_{i_c}'$. Therefore, $\mathcal{S}$ sends (c_i, k_i) in its database to $\mathcal{TP}$ where c_i, k_i was sent by $\mathcal{TP}$ during the registration.
   (b) **Case 2: Incorrect** $share_{i_c} = (c_{i_c}, k_{i_c})$ **employed by the adversary in the real protocol**. $\mathcal{S}$ detects this by verifying that $p_{i_c} \neq p_i$ or $c_{i_c} \neq c_{i_c}'$.

i. If $p_{i_c} = p_i$ and $c_{i_c} \neq c'_{i_c}$, $\mathcal{S}$ sends (c_{i_c}, k_i) to $\mathcal{TP}$, where k_i was in its database.
ii. If $p_{i_c} \neq p_i$, $\mathcal{S}$ generates a random OPRF key $k'_i \neq k_i$, and sends (c_i, k'_i) to $\mathcal{TP}$ where c_i sent by $\mathcal{TP}$ during registration.

Remark: Even though $\mathcal{S}$ does not have any knowledge about k_{i_c} used by $Stor_{i_c}$, he can easily understand if each $Stor_{i_c}$ used the correct input k_i by computing the OPRF locally using k_i in the database. Then, if incorrect k_{i_c} or c_{i_c} are employed in the real protocol, $\mathcal{S}$ also sends incorrect values to $\mathcal{TP}$, in which case both the real and ideal responses will fail.

6. $\mathcal{TP}$ calculates and sends the verification information vk and $userID$ to $\mathcal{S}$ based on the $\{c_i, k_i\}_{i=1,\ldots,t-1}$ received from $\mathcal{S}$, together with (at least) one (c_i, k_i) pair from one of the remaining *n-t+1* honest storage providers to reach the threshold t.
   Remark: $\mathcal{TP}$ employs the ideal user provided password in the ideal world. Therefore, if the adversarial storage providers in the real world acted honestly meaning that the simulator provided correct c_i, k_i pairs, then the calculated verification information will be valid, since it is computed using the actual password. On the other hand, if the storage providers acted maliciously in the real world, $\mathcal{S}$ would have detected this in the previous step, and would have provided wrong pairs to $\mathcal{TP}$ in the ideal world, so in both worlds the response will be invalid.
7. $\mathcal{S}$ forwards $< userID, vk >$ to the adversarial $\mathcal{LS}$ in the real world.

Claim. The view of adversary $\mathcal{A}$, controlling the login server $\mathcal{LS}$ and *t-1* storage providers $\mathcal{SP}_c$, in his interaction with the simulator $\mathcal{S}$ is indistinguishable from the view of his interaction with a real honest party.

Proof. $\mathcal{S}$ acts differently while sending shares c'_i calculated based on randomly chosen pwd' instead of sending actual c_i (sent by $\mathcal{TP}$) calculated based on actual password pwd and executing the $OPRF$ with the $Stor_{i_c}$ using the password pwd' chosen randomly because $\mathcal{S}$ does not have the correct password. If $\mathcal{A}$ can distinguish these behaviors, then we can construct another adversary $\mathcal{A}'$ which breaks either the OPRF construction or TSS construction. We skip this relatively straightforward reductions for the sake of space, but intuitively;

1. **Reduction 1:** The OPRF security ensures that the sender (the adversarial storage providers) cannot distinguish the receiver (the simulated user) input, whether it is the actual password pwd or another randomly chosen password pwd'. Such a reduction will be a hybrid proof, where if at least one adversarial storage provider distinguishes the simulator from the real user, that can be used to distinguish the OPRF receiver input.
2. **Reduction 2:** The TSS security ensures that less than threshold many providers cannot reconstruct the secret and also cannot check if the shares are indeed related to the same secret. Intuitively, if adversarial storage providers can distinguish the simulator, who employs random secret shares during the registration, from the real user, then that can be used to break the security of the underlying threshold secret sharing scheme.

Moreover, even though $\mathcal{A}$ knows the verification information $vk = H(salt||pwd||ls)$ and $\{c_i, k_i\}_{i=1,...,t-1}$ from the registration, $\mathcal{A}$ cannot perform an offline dictionary attack on the password because he needs one more (c_i, k_i) to reach the threshold t to reconstruct the secret *salt*. This part can be information theoretically secured if an information theoretically secure threshold secret sharing scheme (e.g. Shamir [23]), semantically secure encryption scheme and collision resistant hash function are employed.

Theorem 2. *Our Threshold-SPA protocol is secure according to Definition 1 against any non-uniform PPT adversary $\mathcal{A}$ corrupting* ***threshold-many storage providers*** *$\mathcal{SP}_c$, assuming that the threshold secret sharing construction is secure, encryption scheme is semantically-secure, the oblivious pseudorandom function is secure, and the hash function is collision resistant.*

Proof. The simulator $\mathcal{S}$ simulates honest parties (which are the login server $\mathcal{LS}$ and the user $\mathcal{U}$) in the real world and corrupted parties (which are n storage providers denoted by $\mathcal{SP}_c = \{Stor_{i_c}\}_{i_c=1,...,n}$) in the ideal world. $\mathcal{S}$ behaves as follows:

Registration Phase:

1. $\mathcal{S}$ receives $< storUID_i, share_i = (c_i, k_i) >$, where $i = 1, ..., n$ from $\mathcal{TP}$. $\mathcal{S}$ follows the protocol as a user choosing a random password pwd' from the dictionary and a secret share s_{i_c} for each corrupted storage provider, and sends $< storUID_{i_c}, share_{i_c} = (c'_{i_c}, k_{i_c}) >$ where $c'_{i_c} = Enc_{F_{k_i} pwd'}(s'_{i_c})$ to ℓ-*many* adversarial storage providers $\{Stor_{i_c}\}_{i_c=1,...,\ell}$ and for the rest, it sends $< storUID_{i_c}, share_{i_c} = (c_i, k_i) >$ in the real world
2. $\mathcal{S}$ stores the all the data in its database.

Authentication Phase:

1. $\mathcal{S}$ receives $\{storUID_i\}_{i=1,...,n}$ from $\mathcal{TP}$.
   Remark: As $\mathcal{TP}$ may choose a (threshold size) subset of storage providers, we assume the worst case to give the maximum power to the adversary controlling all storage providers and all their values are employed in the protocol.
2. $\mathcal{S}$ sends $storUID_{i_c}$ to each $Stor_{i_c}$ where $i_c = 1, ..., n$.
3. $\mathcal{S}$ executes OPRF protocol with each $\{Stor_{i_c}\}_{i_c=1,...,n}$ using the password pwd', and receives $p_{i_c} \leftarrow OPRF(pwd', k_{i_c})$ and c_{i_c} from each $Stor_{i_c}$ in real.
4. $\mathcal{S}$ checks whether or not each $\{Stor_{i_c}\}_{i_c=1,...,n}$ used the correct corresponding $share_{i_c} = (c_{i_c}, k_{i_c})$ values. $\mathcal{S}$ already holds the correct corresponding values during registration in the database. For each (p_{i_c}, c_{i_c}) received from $Stor_{i_c}$, $\mathcal{S}$ does the following: Using the corresponding c_i, k_i stored in its database during registration, it computes $p_i = F_{k_i}(pwd')$ locally and checks whether or not $p_{i_c} = p_i$, $c_{i_c} = c_i$ for corresponding $n - \ell$ $Stor_{i_c}$ and $c_{i_c} = c'_{i_c}$ for ℓ many $Stor_{i_c}$. There are two cases for each $Stor_{i_c}$:
   (a) **Case 1: Correct** $share_{i_c} = (c_{i_c}, k_{i_c})$ **employed by the adversary in the real protocol**. $\mathcal{S}$ detects this by verifying that $p_{i_c} = p_i$ and $c_{i_c} = c'_{i_c}$ for ℓ-many $Stor_{i_c}$ and $c_{i_c} = c_i$ for n-ℓ-many $Stor_{i_c}$. Therefore, $\mathcal{S}$ sends (c_i, k_i) in its database to $\mathcal{TP}$.

(b) **Case 2: Incorrect** $share_{i_c} = (c_{i_c}, k_{i_c})$ **employed by the adversary in the real protocol**. $\mathcal{S}$ detects this by verifying that $p_{i_c} \neq p_i$ or $c_{i_c} \neq c'_{i_c}$ for ℓ-many $Stor_{i_c}$ and $c_{i_c} \neq c_i$ for n-ℓ-many $Stor_{i_c}$.
   i. If $p_{i_c} = p_i$ and $c_{i_c} \neq c_i$ are sent by $\alpha - many\ Stor_{i_c}$ and $p_{i_c} = p_i$ and $c_{i_c} \neq c'_{i_c}$ are sent by $\beta - many\ Stor_{i_c}$, $\mathcal{S}$ sends $\alpha+\beta$ many (c_{i_c}, k_i) to $\mathcal{TP}$,where k_i was in its database, in case $\alpha+\beta \geq n-t+1$. Otherwise, meaning that $\alpha + \beta < n - t + 1$, $\mathcal{S}$ sends $\alpha + \beta$ many (c_i, k_i) and $n - (\alpha + \beta)$ many (c_{i_c}, k_i) to $\mathcal{TP}$.
   ii. If $p_{i_c} \neq p_i$, $\mathcal{S}$ generates a random OPRF key $k'_i \neq k_i$, and sends (c_{i_c}, k'_i) to $\mathcal{TP}$.

Remark: Even though $\mathcal{S}$ does not have any knowledge about k_{i_c} used by $Stor_{i_c}$, he can easily understand if each $Stor_{i_c}$ used correct input k_i by computing the OPRF locally using k_i in the database. Assume $\alpha+\beta$ adversarial storage providers employed incorrect values. Then, $\mathcal{S}$ also sends exactly $\alpha+\beta$ incorrect values to $\mathcal{TP}$. If $\alpha+\beta \geq n-t+1$ meaning that less than threshold-many correct values were employed, the response will be invalid in the ideal world to the ideal login server, as well as the real response. On the other hand, if at least t values were correct in the real protocol ($\alpha+\beta < n-t+1$), responses in ideal and real worlds will both be valid.

5. $\mathcal{S}$ will not receive anything from $\mathcal{TP}$, and hence halts.

Claim. The view of adversary $\mathcal{A}$, controlling all n-storage providers $\mathcal{SP}_c$, in his interaction with the simulator $\mathcal{S}$ is indistinguishable from the view of his interaction with a real honest party.

Proof. $\mathcal{S}$ acts differently while sending ℓ-many shares c'_i calculated based on randomly chosen pwd' instead of sending actual c_i (sent by $\mathcal{TP}$) calculated based on actual password pwd and executing the $OPRF$ with the $Stor_{i_c}$ using the password pwd' chosen randomly because $\mathcal{S}$ does not have the correct password. If $\mathcal{A}$ can distinguish this behavior, then we can construct another adversary $\mathcal{A}'$ which breaks either the OPRF construction (as in *Theorem 1*) or password based encryption scheme.

If adversarial storage providers can distinguish the simulator, who employs ℓ random secret shares and $n - \ell$ actual shares during the registration, from the real user, than it can distinguish actual secret share c_i based on pwd from chosen random share c'_{i_c} based on pwd', that can be used to break the security of the underlying encryption scheme. Moreover, $\mathcal{A}$ can compute $s_{i_c} \leftarrow Dec_{F_{k_{i_c}}(pwd^*)}(c_{i_c})$ for each pwd^* in the dictionary, then compute the threshold secret sharing reconstruction algorithm to reconstruct the $salt^* \leftarrow TSSRecon(s_1, s_2, ..., s_n)$. For $\mathcal{A}$ to verify if $salt^*$ (and hence pwd^*) is correct, he needs to have actual verification information $vk = H(salt||pwd||ls)$ to compare, which he does not have, since only the login server has that information.

5.2 Further Analysis

Phishing protection: We consider a strong phishing attack with man-in-the-middle between the user and the login server 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'=paypat.com). Therefore, during registration, the user computed $storUID_i \leftarrow H(userID||ls)$, but now for authentication, $storUID'_i \leftarrow H(userID||ls')$ values are computed instead. Thus, honest storage providers will not proceed with the OPRF protocol if a phishing domain ls' is used. Even when all storage providers are corrupted by the phishing attacker and the correct salt is obtained, remember that the original registered $vk \leftarrow H(salt||pwd||ls)$, whereas during attack, the user will send $vk' \leftarrow H(salt||pwd||ls')$ to the attacker. This means the phishing/man-in-the-middle attacker cannot authenticate with the original login server on the user's behalf. Furthermore, because of the security of salt, the adversary cannot obtain any information about the user password, unless threshold-many storage providers are also corrupted.

Handling different domains of the same login server: Ross et al. [22] suggest an approach that enables recognizing that *amazon.com.de* and *amazon.co.uk* accounts belong to the same login server and one registration is indeed enough. Using the same approach for setting ls values, we can also enable the user to authenticate with any one of the valid domains of the login server.

Remembering the storage providers: The human user is not required to remember the storage providers. There are several easy solutions. As addressed by Camenisch et al. [9], the user can remember only a few storage providers who can help direct to other storage providers. Alternatively, a browser extension or a mobile device may remember the list of storage providers employed. Finally, if all storage providers in the whole system are employed by all users, such a public list can be employed, and t of them may be contacted by the user for any given authentication attempt. Observe that publicly listing storage providers does not affect cryptographic security. Our ideal model allows the adversary to know all the storage providers. Therefore, their identities are not hidden when protecting against offline dictionary attacks.

6 Performance Evaluation

In this section, we discuss performance evaluation for the user and storage providers. Since the login server acts the same as current servers, we did not discuss its efficiency. Performance measurement is processed on a standard laptop machine with Intel Core(TM) i7-5600U CPU 2.60 GHz, 8.00 GB RAM, and 64-bit OS. For our implementation, we choose HMAC [25], AES[12], OPRF in [17], and TSS [23] with various thresholds. Table 1 shows the computational performance of the authentication and registration phases. For the registration, the storage providers *do not* compute anything, only receive and store some value. Finally, the user can communicate with the storage providers in parallel, which decreases the network round trip to 1.5 rounds per authentication, which should be added to the login total time in practice.

Table 1. Performance evaluation of Threshold SPA (in milliseconds).

	User (Reg.)	User(Auth.)	Storage Provider	Login Total
1-1 Threshold	0.85	1.14	0.35	1.50
3-6 Threshold	2.84	2.83	0.70	3.53
5-10 Threshold	4.46	3.99	1.30	5.30

7 Conclusion

Recent studies [1, 4, 17] introduced cloud or mobile storage providers to secure passwords against offline dictionary attacks currently prevalent in password-based authentication systems. They provided solutions that ensure that as long as the adversary does not corrupt the login server and the storage provider together, offline dictionary attacks will be prevented. For the first time, in this paper, we provide novel techniques to ensure that multiple storage providers can be employed, and the adversary now must corrupt the login server *and threshold-many* storage providers to be able to mount an offline dictionary attack. We provided an ideal and real world security definition and presented an ideal-real simulation proof. We further ensure phishing, man-in-the-middle, and honeypot attacks are also thwarted. Lastly, our construction employs efficient symmetric key primitives and can easily work with today's hardware, even on mobile devices.

Acknowledgements

We acknowledge the support of TÜBİTAK (the Scientific and Technological Research Council of Turkey) under project numbers 114E487 and 115E766, European Union COST Action IC1306, and the Royal Society of UK Newton Advanced Fellowship NA140464.

References

1. T. Acar, M. Belenkiy, and A. Küpçü. Single password authentication. *Computer Networks*, 57(13):2597–2614, 2013.
2. M. Belenkiy, T. Acar, H. Morales, and A. Küpçü. Securing passwords against dictionary attacks. 2015. US Patent 9,015,489.
3. 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*, pages 72–84. IEEE, 1992.
4. 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.
5. X. Boyen. Hidden credential retrieval from a reusable password. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pages 228–238. ACM, 2009.

6. X. Boyen. HPAKE: Password authentication secure against cross-site user impersonation. In *Cryptology and Network Security*, pages 279–298. Springer, 2009.
7. V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *Advances in Cryptology—Eurocrypt 2000*, pages 156–171. Springer, 2000.
8. J. Camenisch, R. R. Enderlein, and G. Neven. Two-server password-authenticated secret sharing uc-secure against transient corruptions. Public-Key Cryptography–PKC 2015, 2015.
9. J. Camenisch, A. Lehmann, A. Lysyanskaya, and G. Neven. Memento: How to reconstruct your secrets from a single password in a hostile environment. In *Advances in Cryptology–CRYPTO 2014*, pages 256–275. Springer, 2014.
10. J. Camenisch, A. Lehmann, and G. Neven. Optimal distributed password verification. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 182–194. ACM, 2015.
11. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13(1):143–202, 2000.
12. J. Daemen and V. Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.
13. D. Florencio and C. Herley. A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web*, pages 657–666. ACM, 2007.
14. 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). Proeedings. IEEE 9th International Workshops on*, pages 176–180. IEEE, 2000.
15. M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference*, pages 303–324. Springer, 2005.
16. S. Jarecki, A. Kiayias, and H. Krawczyk. Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model. In Advances in Cryptology - ASIACRYPT 2014. Springer Berlin Heidelberg, 2014., 2014.
17. 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*, pages 177–188. ACM, 2016.
18. A. Juels and R. L. Rivest. Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 145–160. ACM, 2013.
19. J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *Advances in Cryptology—EUROCRYPT 2001*, pages 475–494. Springer, 2001.
20. P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. In *Advances in Cryptology—CRYPTO 2002*, pages 385–400. Springer, 2002.
21. M. Mannan and P. C. van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In *Financial Cryptography and Data Security*, pages 88–103. Springer, 2007.
22. B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. In *Usenix security*, pages 17–32. Baltimore, MD, USA, 2005.

23. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
24. E. I. Tatli. Cracking more password hashes with patterns. *Information Forensics and Security, IEEE Transactions on*, 10(8):1656–1665, 2015.
25. J. M. Turner. The keyed-hash message authentication code (hmac). *Federal Information Processing Standards Publication*, 2008.