Computer Networks 57 (2013) 2597–2614

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Single password authentication

Tolga Acar [a,1], Mira Belenkiy [b], Alptekin Küpçü [c,*,2]

[a] Intel Corporation, Bellevue, WA, USA
[b] Microsoft Research, Redmond, WA, USA
[c] Koç University, İstanbul, Turkey

ARTICLE INFO

Article history:
Received 18 November 2012
Received in revised form 11 May 2013
Accepted 23 May 2013
Available online 5 June 2013

Keywords:
Password-based authentication
Dictionary attacks
Malware
Honeypots
Privacy
Mobile

ABSTRACT

Users frequently reuse their passwords when authenticating to various online services. Combined with the use of weak passwords or honeypot/phishing attacks, this brings high risks to the security of the user's account information. In this paper, we propose several protocols that can allow a user to use a *single password* to authenticate to multiple services securely. All our constructions provably protect the user from *dictionary attacks* on the password, and cross-site impersonation or honeypot attacks by the online service providers.

Our solutions assume the user has access to either an *untrusted* online cloud storage service (as per Boyen [16]), or a *mobile* storage device that is trusted until stolen. In the cloud storage scenario, we consider schemes that optimize for either storage server or online service performance, as well as anonymity and unlinkability of the user's actions. In the mobile storage scenario, we minimize the assumptions we make about the capabilities of the mobile device: *we do **not** assume synchronization, tamper resistance, special or expensive hardware, or extensive cryptographic capabilities*. Most importantly, the user's password remains *secure even after the mobile device is stolen*. Our protocols provide another layer of security against malware and phishing. To the best of our knowledge, we are the first to propose such various and provably secure password-based authentication schemes. Lastly, we argue that our constructions are relatively *easy to deploy*, especially if a few single sign-on services (e.g., Microsoft, Google, and Facebook) adopt our proposal.

1. Introduction

A recent study [30] found that the average user logs into 25 different online services in the course of a three month period. The same study found that the average user has only seven passwords, and reuses them for about three accounts on average. To make things worse, users frequently forget their passwords, and try to login via trial-and-error. This means that a malicious online service would learn not only a user's password to that service, but to many other services, possibly also through a cross-site impersonation attack. Even "trusted" services may mount such attacks. For example, in 2004, the CEO of Facebook allegedly used Facebook login data to access the private emails of some business rivals and journalists [22].

In a *honeypot* attack, Bob would create an online service, Bob.com, and convince Alice to create an account on it. Bob.com might even provide some useful service, but the real goal of the honeypot would be to harvest as many usernames and passwords as possible. Bob would then try to use this information to login to other services such as bank accounts. Even if Bob.com was honest, hackers might try to break into Bob.com user database and steal user login data. Bob.com might also leak the information accidentally (e.g., lost company laptops). Analysis of a

* Corresponding author. Tel.: +13476257587.
E-mail addresses: tolga.acar@intel.com (T. Acar), mira.belenkiy@gmail.com (M. Belenkiy), akupcu@ku.edu.tr (A. Küpçü).
[1] Work done at Microsoft Research.
[2] Work done mainly at Microsoft Research.

1389-1286/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comnet.2013.05.007

breach suggests that the user passwords then can be recovered easily via dictionary attacks by the hackers [38].

Even if Bob.com is an honest online service with strong security precautions, Alice still has to worry about potential malware running on her computer. She can become the victim of a phishing attack (where a malicious website impersonates Bob.com), a key logger running on a public terminal, or a virus on her friend's computer.

In this work, we present several solutions to the problem of secure password authentication. Our solutions have three key features:

1. Alice has exactly a *single password* that she can use with all online services (hence the name single-password authentication – SPA).
2. No online service ever learns Alice's password, or any *deterministic* function of Alice's password. In particular, no online service ever learns enough information to impersonate Alice with any other service.
3. Alice's user experience is simple and similar to the typical password login experience she is already used to.

We accomplish these three goals with the help of a storage device. We consider two scenarios. In the first scenario, the storage device is actually an online cloud service – Carol.net. Alice distrusts Carol.net as much as she distrusts Bob.com. Yet, as long as Carol and Bob do not collude, Alice's password is safe. For simplicity, we describe our protocols assuming Carol and Bob do not collude (e.g., Carol is Microsoft and Bob is Google), but in Section 7 we show how to relax this assumption. In our second scenario, where Alice has a trusted mobile device (e.g., a smartphone), Alice's password and online accounts are safe even if an adversary (except Bob, since otherwise it means the storage and the online service are colluding) steals her mobile device. Note that such a helper storage means that Alice may use any computer to login to her accounts (in contrast to password managers that require installation on each computer Alice would like to use).

Our cloud-based solutions are based on Boyen's Hidden Credential Retrieval (HCR) protocols [16]. HCR lets a user securely store a *random* value on an untrusted server. Due to the properties of the HCR scheme, only the user who knows a short password can retrieve the data. An adversary who does not know the password cannot launch a dictionary attack to retrieve the secret data because the adversary has no way to test whether the data it retrieves is the correct random value via offline means. The adversary may then try to mount an online attack, but Bob.com will block his attempts after several unsuccessful tries, since by assumption they are not colluding.

Our mobile SPA constructions only assume that the client computer has a keyboard, monitor, and connection to the Internet. Furthermore, we make minimal assumptions about the mobile device capabilities (a keyboard, a display, and a camera/microphone or SMS/Internet connection). Instead of trying to implement heavy-weight cryptography on the mobile device, we use only simple cryptographic primitives such as symmetric encryption and MAC. We leverage a crucial observation in our constructions: It is easy to pass a relatively large amount of data ($\geqslant$128-bits) from the client computer or the server to the mobile device.

Related Work: Establishing a secure authenticated channel is a well-studied problem [5]. Authenticated key exchange (AKE) goes back to the Diffie-Hellman authenticated key exchange [26]. Once the server and the client establish a shared secret key, it is straightforward to create a secure communication channel. Today, common protocols like SSL give the choice of client-side authentication and/or server-side authentication.

We are interested in mutually authenticated channels that require the client to memorize only a small amount of information, i.e. a password. In Password-Authenticated Key Exchange (PAKE), the client and the server start with a shared password known to both. The earliest example of PAKE is Bellovin and Merritt's Encrypted Key Exchange (EKE) [6], and there are many variations [59,39]. Since we are worried about malicious servers and cross-site impersonation, we require that **the server never learns the client's password**.

Asymmetric Password-Authenticated Key Exchange (APAKE), attempts to remedy this problem. Only the client knows the password, while the server stores a one-way function of the password [53,7,32]. However, *APAKE schemes are still vulnerable to dictionary attacks* by the server, or by hackers stealing information from the server.

Boyen [16] shows that any password-based authentication protocol that involves only two parties (the client and the server) is vulnerable to a dictionary attack by the server. The server can always try every single possible password to see if it allows a successful authentication. The best that a PAKE/APAKE scheme could hope for is to increase the cost of the dictionary attack. Boyen's HPAKE scheme [17] lets the client control the cost of a dictionary attack; the client chooses a security parameter τ, and performs $\theta(\tau)$ work during registration and authentication. However, as long as τ is polynomial, which must be the case to have an efficient client, so is the cost to the server to launch a dictionary attack.

One way to overcome the inherent limitations of a two-party password-based authentication protocol is to add more parties to the protocol. This can be done by having the client authenticate with multiple servers. Some systems, e.g., by Ford and Kaliski [31], require all the servers to participate in every authentication, while others systems, e.g., by MacKenzie et. al. [45], require only a subset of k out of n servers to participate. All such schemes require a prior setup where the servers exchange keys. In our Cloud SPA constructions **the cloud storage and the online service do not need to interact in any way, or even be aware of the others' presence**.

Another option is to add a trusted mobile device that the user carries [52,46]. Devices such as smart cards eliminate entirely the need for password-based authentication. However, they *do not scale* well to multiple independent online services. Dedicated hardware devices (such as key fobs) are usually tied to one online service (e.g., corporate network login for employees); any such solution would result in the user carrying many such devices. In addition, smart card readers are not standard on all machines and may not be present on public terminals (such as those in hotels

or airports). The same problem holds for other hardware token devices that require a physical or wireless communication channel with the client computer. Many such solutions assume tamper-proof hardware, and mainly target phishing attacks rather than dictionary attacks. Our mobile SPA protocols assume only standard hardware, and are **provably secure even if the mobile device is stolen** and its contents are revealed (no tamper-resistance required).

Our constructions focus on achieving *authentication* rather than *authenticated key exchange*. There exist many protocols for achieving a secure channel between two parties given an initial set of secrets; TLS, SSH, and Kerberos are pervasive examples. Instead of creating a new secure channel protocol that is unlikely to be adopted, we propose practical schemes that are *easily deployable* with existing infrastructure.

Finally, *other attempts to provide single-password authentication fail to provide complete security against dictionary attacks* by the server or hackers hacking into the server, since they lack formal definitions and proofs [60,33]. For example, the SPP scheme [33] assumes the users can remember long and random passwords; otherwise the scheme is *insecure* against dictionary attacks. It is possible to use only the information known to the server to mount a dictionary attack, since the server also stores the randomness used in the hash function (see our observations below). Unfortunately Imperva analysis shows that half of the user passwords are already susceptible to dictionary attacks [38]. In the SOKE scheme, the authors admit that using the same password for multiple servers makes it even easier to mount dictionary attacks [1]. Recent industrial solutions trying to secure the server-side password databases [56] make it harder but not impossible to mount dictionary attacks. The closest formalization to our technique is the virtual soft-token idea [55], but again without provable security against dictionary attacks. Our constructions are all provably secure against dictionary attacks, and **it is possible that the single password that is used is simple, as long as it is hard to guess** (i.e. secure against social attacks) (e.g., not so obvious as a birth date).

While we provide provable guarantees against many common attacks, we do not fully protect against man-in-the-middle attacks or malware. If the adversary can successfully mount a man-in-the-middle attack (e.g., via attacking SSL, secure DNS, or certificates, or by installing malware on the machine used), the the adversary may steal a single session in our mobile-based solutions. This is a problem inherent in today's world, and would require modifications in network protocols with support from browsers and operating systems. Yet, we make sure to **protect the user's persistent, long-term password even under successful man-in-the-middle attacks or malware**.

Our contributions may be summarized as follows:

1. We formally define single-password authentication schemes, and security against attacks by a malicious server, or a malicious storage, or an external adversary, both in *cloud-* and *mobile-based scenarios*.
2. We present, to the best of our knowledge, the first *provably-secure* single-password authentication schemes with various *performance/privacy/usability* considerations.
3. Our cloud-based solutions do not require that multiple servers need to communicate with, or even know each other, and can be *easily deployed via independent vendors*.
4. Our mobile-based solutions can work with current *standard* cell-phones or smart-phones, and remain *secure even when the device is stolen*.
5. Overall, we propose methods to (completely or partially) protect against **dictionary attacks, honeypot attacks, cross-site scripting attacks, phishing attacks, and malware**. To the best of our knowledge, such a comprehensive proposal on password-based authentication have not been done before.

To enable these contributions, we make three key observations:

1. Password-based encryption is inherently susceptible to dictionary attacks. Therefore, simply using a master password to encrypt all other passwords do not constitute a solution. The key idea we use is the fact that **password-based encryption is insecure unless the encrypted input is indistinguishable from random**.
2. While *offline* dictionary attacks are very easy to mount and fast, *online* dictionary attacks are inherently slow and easy to protect against. The standard mechanism in use today (limiting the number of unsuccessful attempts) is enough to successfully thwart *online* dictionary attacks. Thus, in our solutions, **we make sure that the only way of testing correctness of the decryption of a password-based encryption is *online*** through trying to login to the server (i.e. in cryptographic terms, the number of queries that can be made to a distinguishing oracle is limited to some constant).
3. Current authentication protocols, where there is no randomization by the client or the server, are doomed to be susceptible to dictionary attacks (by the server or hackers hacking into the server), since the server has a deterministic function of the client's password (see also [35,15]). To overcome such a limitation, **the authentication protocol must be a challenge-response type protocol** where the server never learns any *deterministic* function of the client's password.

2. Preliminaries

We say that $neg(k)$ is a negligible function in k if $\forall c$ constant: $\exists K$ finite: $\forall k > K$: $neg(k) < k^{-c}$. The notation $m \leftarrow \{0,1\}^k$ denotes picking a value m uniformly at random from the space of all k-bit strings. 1^k denotes a k-bit string of ones. The symbol $\oplus$ denotes *bitwise exclusive or* operation. We write $\langle a,b\rangle$ to denote a database with columns of the types of a and b. For simplicity, we sometimes pick a row (a,b) from the database $\langle a,b\rangle$ that matches a particular value a.

A function that can be executed by a single party is denoted $\mathsf{Function}(input) \rightarrow (output)$. To denote a two-party protocol, we write $\{\mathsf{Alice}(input_A), \mathsf{Bob}(input_B)\} \rightarrow \{\mathsf{Alice}(output_A), \mathsf{Bob}(output_B)\}$. This means that Alice executes the protocol with $input_A$ and receives $output_A$ and Bob executes the protocol with $input_B$ and receives

$output_B$. If one of the players has no input (or output), we may omit writing.

Since users can typically remember only a short low-entropy password, we will often define security in terms of two parameters: k which will be large (e.g., 80–128), and ℓ, which will be smaller (e.g., 30–40). We will use $m \ll k$ to emphasize that the value m is much smaller than the value k. The notation $\mathsf{ProbGuess}(N)$ represents the probability that an adversary can guess the user's password in N tries. In some cases, $\mathsf{ProbGuess}(N)$ will present an inherent upper bound on the security of the authentication scheme, attacked through social measures or dictionary attacks.

Definition 1 (*Probability of Guessing*). Let Adv be a probabilistic polynomial time (PPT) adversary that has some apriori knowledge of how a user chooses a password during UserGen. We define:

$$\mathsf{ProbGuess}(N) = \Pr[(name, pwd) \leftarrow \mathsf{UserGen}(1^\ell); (pwd'_1, \ldots, pwd'_N) \leftarrow \mathsf{Adv}(1^\ell, name) : pwd \in \{pwd'_1, \ldots, pwd'_N\}].$$

Since *pwd* need not be chosen from a uniform distribution, $\mathsf{ProbGuess}(N) \geqslant N/2^\ell$.

One may consider an alternative definition where the adversary gets to adaptively query a yes/no oracle with N different passwords, and then has to output a guess. This definition is *equivalent* to Definition 1. The optimal strategy for an adversary without a yes/no oracle is to output the N passwords with which it would have queried the oracle had it said "no" each time. The optimal strategy for an adversary with a yes/no oracle is to keep querying until either the oracle says "yes" or it runs out of guesses (limited to N). Since the adversary stops when he finds the correct password, this definition is *equivalent* to the *adaptive* version, because the adversary may assume the oracle's answer is "no" without loss of generality.

Note that we do not restrict the probability of guessing a password in any manner. Thus, it may even include *social attacks*, which, in general, have very high probabilities of guessing the password correctly. Obviously, no password-based authentication mechanism can be foolproof to such attacks. But, what we provably show is that our system makes sure the advantage of the adversary in addition to any such attack is *negligible*.

Digital Signature schemes are made up of three PPT algorithms: $\mathsf{SigKeyGen}(1^k)$ generates a secret signing key *ssk* and a public verification key *svk*. $\mathsf{Sign}(ssk,msg)$ generates a signature *sig* on the message *msg* using the secret key *ssk*. $\mathsf{SigVerify}(svk,msg,sig)$ outputs accept if *sig* is a valid signature on *msg* given the public verification key *svk*, outputs reject otherwise.

A digital signature scheme must be secure against an (adaptive) existential forgery attack. The adversary is given the verification key *svk* and an oracle that will sign any message of the adversary's choice, adaptively. No PPT adversary should be able to output a valid message-signature pair (*msg*,*sig*) such that the verification function $\mathsf{SigVerify}(svk,msg,sig)$ will accept and the oracle has not previously given the adversary any signature on *msg*.

Message Authentication Code (MAC) is the symmetric key version of a digital signature scheme. It has the same protocols as above, except $\mathsf{MACKeyGen}(1^k)$ outputs just one key (i.e. $ssk = svk$). The definition of security is the same as for digital signatures, except that the adversary does not see *svk*, since it is secret.

Blind Signature is an extension of digital signatures that allows a user to receive a signature on a message without revealing the message to the signer. BSigKeyGen generates a signing key *bsk* and a verification key *bvk*. The signing algorithm is denoted as $\mathsf{BSign}(bsk,msg)$, which corresponds to the interactive protocol $\{\mathsf{Signer}(bsk), \mathsf{Receiver}(msg)\} \rightarrow \{\mathsf{Receiver}(sig)\}$. To verify the resulting signature, one runs $\mathsf{BSigVerify}(bvk,msg,sig)$. The security properties are the same as for digital signatures, with the addition that BSign is a secure two party computation scheme (i.e. the inputs of the parties remain private, and the output is given to the user). Our constructions require *unique blind signatures*, such as those due to Boldyreva [10], that allow *only one valid signature per key-message pair* (*bsk*,*msg*). We furthermore require that the blind signer does not learn the signature (in addition to not learning the message). The Boldyreva [10] blind signature has this property also.

Symmetric Encryption schemes consist of three PPT algorithms: $\mathsf{EncKeyGen}(1^k)$ generates a secret key *esk*. The encryption algorithm $\mathsf{Encrypt}(esk,msg)$ encrypts the message *msg* using the secret key *esk* and outputs a ciphertext *ctext*. The decryption function $\mathsf{Decrypt}(esk,ctext)$ uses the secret key *esk* to decrypt the ciphertext *ctext* and outputs the original message *msg*. For security definitions, see e.g., [40].

Encryption schemes typically need strong (e.g., 128-bit) secret keys to ensure security. Our constructions frequently use short ℓ-bit passwords as the encryption key. As a result, one inherently expects less security from such a scheme. One way to get around this dilemma is to encrypt k-bit random messages, in the hopes that the inherent "randomness" of the message will prevent a dictionary attack. However, it seems there is no straightforward reduction from secure password based encryption to semantic security. Below we present what it means for a password-based encryption scheme to be secure, borrowing ideas from the definition of security of symmetric encryption.

Definition 2 (*Secure Password-based Encryption*). A password-based encryption scheme is secure against dictionary attacks **under random messages** if $\forall$ PPT adversaries Adv the following holds:

$$\Pr[msg \leftarrow \{0,1\}^k; (pwd_0, pwd_1, state) \leftarrow \mathsf{Adv}(1^k, 1^\ell); b \leftarrow \{0,1\}; ctext \leftarrow \mathsf{Encrypt}(pwd_b, msg); b' \leftarrow \mathsf{Adv}(1^k, state, ctext) : b = b'] = 1/2 + neg(k)$$

Consider a commutative encryption scheme, where for any choice of random coins r, $\mathsf{Encrypt}(esk,msg,r) = \mathsf{Encrypt}(msg,esk,r)$. In this case, *semantic security of an encryption scheme does imply secure password-based encryption* (we do not provide a full proof here for the sake of space, but we hope the reader sees that it is relatively straightforward). One example of such an encryption scheme is the one-time pad, where $\mathsf{Encrypt}(esk,msg) = esk \oplus msg$.

Note that, even though we defined secure password-based encryption using a single message, it can be easily extended to a multi-message definition. A standard hybrid argument is enough to show that the single-message security implies multi-message security. Briefly, assume adversary $\mathcal{A}$ breaks single-message security. Then, adversary $\mathcal{B}$, given n messages, guesses a value j between *1* and n, and sets the single message for $\mathcal{A}$ to be the jth message. $\mathcal{B}$ then forwards to its challenger the same passwords $\mathcal{A}$ provides. Upon receiving n challenge ciphertexts, $\mathcal{B}$ forwards the jth ciphertext to $\mathcal{A}$ as the challenge. Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs. It is easy to see that probability of success for $\mathcal{B}$ is at least $1/n$ times the probability of success of $\mathcal{A}$, and thus if $\mathcal{A}$ succeeds in breaking the single-message security with non-negligible probability, then $\mathcal{B}$ breaks the multi-message security with non-negligible probability.

Throughout our paper, we assume that *the secret keys for the digital signature schemes and message authentication codes used are indistinguishable from random values*. For example, in DSS [50] the signature private key is a random integer up to the order of the group that is being used, and for MAC constructions using HMAC [49], the key is a random string of length equal to the block length of the hash function. Luckily, mostly the block lengths of encryption and MAC schemes are compatible, and the orders of the groups used in DSS are mostly a multiple of that block size, therefore preventing padding that makes the decryption of the encrypted signature/MAC key distinguishable. In particular, **no deterministic padding scheme should be used**, since padding can help the adversary distinguish the decryption from random.

Whenever necessary, hash functions are employed to make the lengths match, and are used as *random oracles* as in password-based encryption. We assume that *SSL connections are always in use* between the client and the server or the storage, and therefore the messages never leak to a third party. Moreover, both *the server and the storage provider limit the number of attempts* to prevent online dictionary attacks (see [14] for real world analysis of these assumptions).

2.1. Hidden credential retrieval

Boyen [16] presents a scheme for storing and retrieving data from an untrusted online storage provider. The user has a trusted client, and is capable of remembering only a short password. The storage provider is assumed to be potentially malicious. Our constructions use a *modified version* of Boyen's scheme as a building block:

$\mathsf{Store}\ \{\mathsf{Client}(1^k, pwd, data)\} \rightarrow \{\mathsf{Storage}(id, ctext, bsk)\}$

1. The client starts by generating two key-pairs; one for blind signature $(bsk, bvk) \leftarrow \mathsf{BSigKeyGen}(1^k)$ and one for digital signature $(ssk, svk) \leftarrow \mathsf{SigKeyGen}(1^k)$.
2. Afterward, the client computes a blind signature value $sig \leftarrow \mathsf{BSign}(bsk, \mathsf{Hash}(pwd))$ and encrypts the data using hash of this signature: $ctext \leftarrow data \oplus \mathsf{Hash}(sig)$.
3. The client sends her blind signature signing key, the encrypted data, and an identifier $(id, ctext, bsk)$ to the storage.

$\mathsf{Retrieve}\ \{\mathsf{Client}(pwd), \mathsf{Storage}(ctext, bsk)\} \rightarrow \{\mathsf{Client}(data)\}$

1. The client and the storage execute the blind signature protocol. The storage acts as the signer using bsk as the signing key. The client acts as the receiver, using $\mathsf{Hash}(pwd)$ as the message. The client gets the signature $sig = \mathsf{BSign}(bsk, \mathsf{Hash}(pwd))$ as its output.
2. The storage sends the ciphertext $ctext$ to the client.
3. The client computes the decryption of the ciphertext as $data \leftarrow ctext \oplus \mathsf{Hash}(sig)$.

Theorem 1 (Boyen's Storage Protocol [16]). *Any PPT adversary that makes T impersonation queries succeeds in recovering the password with the following probability*

$$\Pr[\text{Adversary obtains password}] \leqslant \mathsf{ProbGuess}(T) + neg(k)$$

Boyen defines impersonation queries in terms of an adversary either impersonating the storage to the client ("insider" attack) or impersonating the client to the storage ("outsider") attack. The adversary also has access to oracles for testing passwords. We have simplified the notation to consider all types of queries as an impersonation query. In Boyen's original definition [16], the adversary wins if it is able to recover the stored credential, called sk, which is a k-bit secret key. This is equivalent to learning the password, since given the credential, it is possible to launch an off-line dictionary attack and learn the password, or given the password, it is easy to learn the credential.

As Boyen brilliantly notes, any mechanism between Alice the client and Carol the storage provider during the retrieval phase must *not* output any success or failure signals to either party. Any such indicator output will enable Carol to perform offline dictionary attacks. Furthermore, methods other than blind signatures can also be used in this phase (see [16] for a good overview of such methods).

3. Single password authentication

An SPA protocol lets a user Alice register and authenticate with multiple online services using the same persistent password. In doing so, Alice may employ an untrusted cloud provider, or a trusted mobile device, as storage. The goal of an SPA protocol is to protect Alice's long-term password from online services, storage providers, and external adversaries.

Fig. 1 presents an overview of our protocols. The goal of the **registration phase** is to let Alice register with Bob and store her secret securely at Carol. First, Alice generates a random strong secret key (e.g., k-bits), and a verification mechanism associated with it (e.g., asymmetric keys for a digital signature scheme or symmetric keys for a MAC scheme). Then, she registers the verification mechanism with the server. Optionally, she can contribute with her password (without revealing it to Bob), and Bob can contribute with his own secrets, and Alice can get some random-looking identifier at the end of the protocol.

In the same phase, Alice also needs to register with Carol. The idea is that Alice will store her strong secret that looks random, encrypted using her password, on the storage provider with a random-looking identifier.

Furthermore, the choice of the identifier is important in terms of privacy of Alice. We will present several options for our protocols, presenting a performance-privacy trade-off.

Once the registration with the server and the storage is done, the **authentication phase** may take place as many times as requested. For Alice to login to Bob.com, first Alice identifies herself to Bob, then Bob will challenge her, and to respond to that challenge Alice needs to retrieve her strong secret from Carol. Depending on the scheme, Alice can run the *authenticate* protocol with Bob and the *retrieve* protocol with Carol in parallel. Yet, in some schemes, Alice first needs to obtain the random-looking identifier with the help of Bob, and use it to retrieve-and-decrypt her secret from Carol. Finally, Alice responds to Bob's challenge using the secret she retrieved from Carol.

We use slightly different trust models in the case of cloud storage versus a mobile storage device. In both scenarios, Alice completely trusts her computer (browser) during the *registration phase* of the protocol. However, in the mobile storage scenario, the browser is assumed to be adversarial and may collude with the online service during the *authentication phase*. This is to model situations where (1) Alice might be using an untrusted (public) terminal and (2) the online service might send malicious code to Alice's browser. As a result, we have to distinguish between the user Alice and the computer client she uses.

3.1. SPA protocol

An SPA protocol has three types of players: Clients who want to use a password to access services, servers who register and authenticate clients, and storage providers who store data for the clients and assist with the registration/authentication process.

An SPA protocol consists of the following algorithms:

UserGen: $\{\mathsf{Client}(1^\ell)\} \rightarrow \{\mathsf{Client}(name, pwd)\}$. This algorithm is run by the client to generate a username *name* and an ℓ-bit password *pwd*.

Register: $\{\mathsf{Client}(1^k, name, pwd, servername), \mathsf{Server}(1^k, servername)\} \rightarrow \{\mathsf{Client}(sk, id), \mathsf{Server}(name, state)\}$. Using this two-party protocol, the client registers with the server. At the end, the client gets as output a secret key *sk* and a unique identifier *id*. The server stores (*name*, *state*) in its database $\langle name, state\rangle$.

Store: $\{\mathsf{Client}(pwd, sk, id)\} \rightarrow \{\mathsf{Storage}(id, data)\}$. The client uses its password *pwd* and the output it got from the registration protocol to compute *data* (generally an encryption of *sk*). The client sends (*id*, *data*) to the storage provider for safe keeping. At this point, the client may forget/erase (*id*, *data*, *sk*), but remembers (*name*, *pwd*).

PreAuth: $\{\mathsf{Client}(name, pwd, servername), \mathsf{Server}(servername, \langle name, state\rangle)\} \rightarrow \{\mathsf{Client}(id, chal), \mathsf{Server}(state, chal)\}$. The client uses its username *name* and password *pwd* to retrieve its identifier *id* from the server. The server retrieves the state associated with the client's username from its database. The server sends a challenge *chal* to the client, and remembers it.

Retrieve: $\{\mathsf{Client}(id, pwd, chal), \mathsf{Storage}(\langle id, data\rangle)\} \rightarrow \{\mathsf{Client}(sk)\}$. The client uses its identifier *id* and password *pwd* to retrieve its strong secret key *sk* from the storage provider. The storage provider uses its database $\langle id, data\rangle$ as input, and gets no output.

Authenticate: $\{\mathsf{Client}(sk, chal), \mathsf{Server}(state, chal)\} \rightarrow \{\mathsf{Server}(\mathsf{accept/reject})\}$. The client uses its strong secret key *sk* to prove to the server that it owns the account corresponding to *state*, by respond-

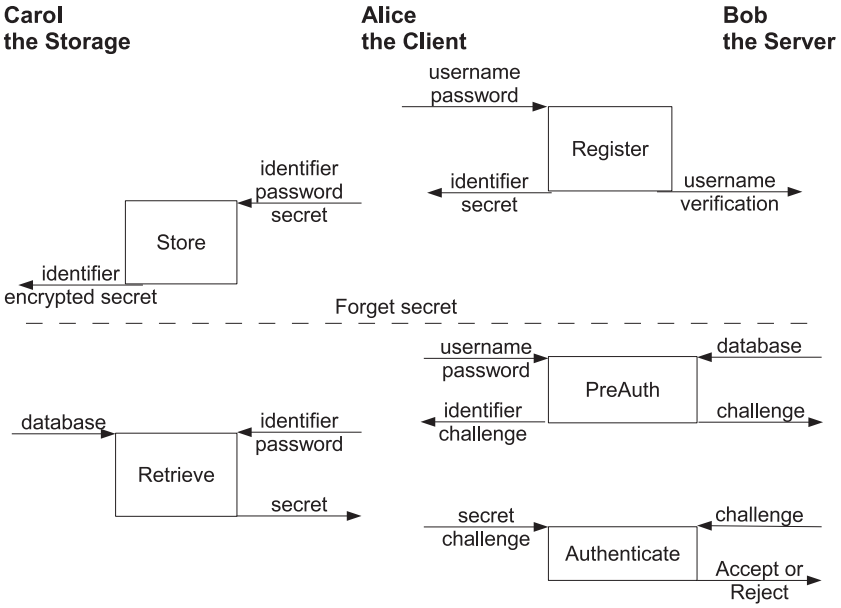

Fig. 1. SPA protocol. The registration phase is presented above the dashed line, and below it is the authentication phase.

ing to the challenge *chal*. The server outputs accept or reject.

It is important to note that **all our definitions require that the adversary has negligible advantage in the security parameter k and not the weaker password-security parameter ℓ**. This is the key property of our definitions and schemes that provably prevent dictionary attacks.

3.2. Secure Cloud SPA

A Cloud SPA uses storage providers that are available online. We assume that when the client accesses the cloud storage provider, the client is guaranteed to connect to the storage provider to which it intends. This may be achieved through secure DNS systems and SSL connections. In addition, we assume the client's computer is trusted (i.e. free of malware). Later, when we present mobile SPA, we show how the user can be protected even if her computer is infected.

Both the server and the storage provider may be malicious; however, they may *not collude* (we present a solution to the collusion problem later). Besides, to prevent online dictionary attacks, as explained in Theorem 1, we assume that the server limits the number unsuccessful Authenticate attempts and the storage provider limits the number of Retrieve requests that a client may make (note here that the storage provider does not know whether or not those requests were successful, but needs to limit their frequency anyway to thwart online dictionary attacks). We now formalize these notions.

Definition 3 (*Secure Cloud SPA*). A Cloud SPA protocol is secure if it provides Cloud Honeypot Security and Cloud Storage Security (defined below).

(T,N)-**Cloud Honeypot Security Game**. In this game, a malicious server tries to learn an honest client's password. There is one honest client who has access to N different honest cloud storage providers. The adversary plays the role of the server. It can ask the client to execute Register, PreAuth, and Authenticate using any of the cloud storage providers. The adversary is also allowed to execute the Retrieve protocol directly with any storage provider, up to T times per provider. (There is no point to allowing the adversary to execute Store because the adversary can simulate the result of any Store request without the assistance of a storage provider). In the following game, there are three phases, performed in the presented order, where the challenger plays the roles of the client and the storage providers.

Setup. The challenger creates username and password $(name, pwd) \leftarrow \mathsf{UserGen}(1^\ell)$. The challenger sends *name* to the adversary.

Play. The adversary interacts with the client and storage provider via the following four protocols. The challenger maintains a *sessionid* that uniquely identifies each interaction.

PlayRegister(*servername*,*i*). The client must register with a server chosen by the adversary. The adversary generates *servername* and sends it to the challenger. Then the challenger and adversary execute Register, where the challenger plays an honest client with input $(1^\ell, name, pwd, servername)$ and the adversary plays the server. After the Register protocol completes, the challenger simulates the Store protocol between the client and the storage by computing $data \leftarrow \mathsf{Store}(pwd, sk, id)$ and storing $(id, data)$ in database $\mathsf{Storage}[i]$. If $\mathsf{Storage}[i]$ already has an entry with *id*, the challenger overwrites it.

PlayPreAuth(*servername*,*sessionid*). The adversary asks the client to run the PreAuth protocol and plays the server role. The challenger plays the role of an honest client and executes the PreAuth protocol with input $(name, pwd, servername)$. After the PreAuth protocol completes, the challenger stores $(sessionid, id, chal)$ in its PreAuth database.

PlayAuthenticate(*sessionid*,*i*). The adversary asks the client to run the Authenticate protocol. The challenger looks up the *id* associated with the *sessionid* in its PreAuth database. Then the challenger simulates the Retrieve protocol between the client and the storage provider $\mathsf{Storage}[i]$, obtaining *sk*. The challenger then uses *sk* to execute the Authenticate protocol with the adversary.

PlayRetrieve(i). The adversary interacts directly with the ith storage provider. For each $i \in [1, N]$, the adversary is limited to call PlayRetrieve(i) at most T times. The challenger plays the role of an honest storage provider and executes Retrieve using the database $\langle id, data \rangle$ of $\mathsf{Storage}[i]$ as its input.

Output. The adversary outputs his guess pwd' for the password. The adversary wins if $pwd' = pwd$.

Note that the adversary can interact with N cloud storage provider T times, and has one more try during the *Output* phase of the game ($TN + 1$ guesses in total). After each Retrieve interaction, the adversary can check whether or not he guessed the password correctly, therefore making adaptive guesses. Since *pwd* is a short (i.e. ℓ-bit) secret, the adversary can guess it with probability $\mathsf{ProbGuess}(TN + 1) \geqslant (TN + 1)/2^\ell$.

Definition 4 (*(T,N)-Cloud Honeypot Security*). We say that a Cloud SPA has (T,N)-Cloud Honeypot Security if no PPT adversary can win the (T,N)-Cloud Honeypot Security game with probability more than $\mathsf{ProbGuess}(TN + 1) + neg(k)$.

(T,N)-**Cloud Storage Security Game**. In this game, a malicious storage provider tries to impersonate an honest user to an honest server. The adversary plays the role of the storage provider, while the challenger plays the role of the client and N different servers. The adversary can interact with the client polynomially-many times (asking the client to store and retrieve), and can make T PreAuth and Authenticate queries to each server.

Setup. The challenger computes $(name, pwd) \leftarrow \mathsf{UserGen}(1^{\ell})$, but discards the *name*.

Play. The adversary interacts with the client and online service via the following three protocols.

PlayStore(i,*name*,*servername*). The client must register with a server chosen by the adversary using a username chosen by the adversary, and store the result. The challenger simulates the Register protocol between the client and the ith server to compute (*sk*,*id*,*state*). The client's input is (*name*,*pwd*,*servername*) and the ith server's input is (*servername*). The challenger then runs Store(*pwd*,*sk*,*id*) with the adversary, where the adversary obtains (*id*,*data*). The challenger locally stores (*servername*,*name*,*state*) in the database Server[i] overwriting *state* if an entry with (*servername*,*name*) already exists.

PlayRetrieve(i,*name*,*servername*). The client must run the PreAuth protocol with the ith server and the Retrieve protocol with the adversary. First, the challenger looks up (*servername*, *name*, *state*) in the database Server[i]. Then it computes (*id*,*chal*) by executing PreAuth simulating both the client and the ith server. The client's input in this execution is (*name*, *pwd*, *servername*), and the ith server's input is (*servername*, *name*, *state*). Then the challenger plays the role of the client and executes Retrieve with the adversary using input (*id*, *pwd*, *chal*). The challenger stores (*servername*, *name*, *state*, *chal*) in its Authenticate[i] database (no overwriting of duplicate entries).

PlayAuthenticate(i,*name*,*servername*,*chal*). The adversary tries to authenticate with the ith server. The challenger looks up (*servername*, *name*, *state*, *chal*) in its Authenticate[i] database and outputs reject if it is not there. Then the challenger executes the Authenticate protocol with the adversary; the challenger plays the role of ith server with input (*name*, *state*, *chal*, *servername*). The challenger outputs accept or reject.

Output. The adversary wins the game if the challenger ever outputs accept during PlayAuthenticate.

Definition 5 (*(T,N)-Cloud Storage Security*). We say that a Cloud SPA has (T,N)-Cloud Storage Security if no PPT adversary can win the (T,N)-Cloud Storage Security game with probability more than $\mathsf{ProbGuess}(TN+1) + neg(k)$.

3.3. Secure Mobile SPA

A mobile SPA assumes that the user has access to a trusted storage device (e.g., a cell-phone). The user completely trusts the device while it is in his possession. Therefore, the user can even enter his persistent password into the device. However, the device might be lost or stolen at any point; and at that point a malicious adversary might try to use the data stored on the device to access the user's online services, or even to try to learn the persistent password of the user. The trusted device can perform computations on the user's behalf. This is important if the user does not trust the terminal he/she is using to authenticate with the online service (e.g., accessing his bank account from a hotel computer).

Definition 6 (*Secure Mobile SPA*). A mobile-based SPA protocol is secure if it has the Mobile Honeypot Security and Mobile Storage Security properties.

Mobile Honeypot Security Game. This game is similar to the Cloud Honeypot Security game where the adversary acts as a malicious online service.

Setup. The adversary chooses two passwords, pwd_0 and pwd_1. The challenger flips a bit b and sets pwd_b as its password.

Play. The adversary may invoke all the same queries as during the Cloud Honeypot Security game *except* the PlayRetrieve query.

Output. The adversary outputs a bit b'. It wins if $b = b'$.

Definition 7 (*Mobile Honeypot Security*). A Mobile SPA protocol has the Mobile Honeypot Security property if no PPT adversary can win the Mobile Honeypot Security game with probability more than $1/2 + neg(k)$.

Winning the Mobile Honeypot Security game merely requires distinguishing two passwords, while the Cloud Honeypot Security game requires full password recovery. The reason is that in the Cloud Honeypot Security game, an adversary can query an online cloud storage provider with pwd_0 and pwd_1 to see which of them is valid.

On another note, our honeypot security games, as is, define security in terms of password recovery or password distinguishing, while intentionally leaving man-in-the-middle attacks out. As pointed out in the introduction, we do not fully protect against man-in-the-middle attacks, but limit their potential to session hijacking rather than full long-term secret leaks.

(T,N)-**Mobile Storage Security Game.** This game simulates what happens if the storage device is stolen by an adversary. It is similar to the Cloud Storage Security game. In the beginning of the *Play* stage, the storage device is honest. The adversary does not get to see the outcome of any PlayStore or PlayRetrieve query. At some point, the adversary may choose to corrupt the storage device. The adversary gains access to all the information on the storage device once it is corrupt. However, at that point, the challenger ceases to interact with that device (modeling the real-world scenario that the user stops using her cell-phone once it is stolen). Specifically, the challenger no longer simulates calls to Store or Retrieve, which means the challenger will terminate early during the PlayRegister and PlayAuthenticate queries. The adversary wins if it can convince the online service to output accept during PlayAuthenticate.

Definition 8 (*(T,N)-Mobile Storage Security*). A Mobile SPA protocol has the (T,N)-Mobile Storage Security property if no PPT adversary can win the (T,N)-Mobile Storage Security game with probability more than $\mathsf{ProbGuess}(TN+1) + neg(k)$.

3.4. Privacy

We consider two notions of privacy:

Anonymity. A malicious storage provider cannot learn Alice's username at a particular server.

Unlinkability. A malicious storage provider cannot link a Store and a Retrieve request, or even two Retrieve requests, when they come from the same user.

Note that Alice already registers a username with the online service, and uses the same username so that her logins can be linked to provide the service (e.g., one needs a fixed email address to obtain meaningful service most of the time). Therefore, we are not trying to protect Alice's privacy against Bob. Anonymous authentication methods [23,18,44,21,4] can be employed if that is desired.

Definition 9 (*Privacy of an SPA protocol*). An SPA protocol is **anonymous** if all PPT adversaries have negligible advantage in winning the Anonymity game below. The protocol is called **unlinkable** if Carol the storage provider cannot link two requests made using the same identifier.

(T)-**Anonymity Game**. In this game, the adversary plays the role of Carol the storage provider, whereas the challenger plays the role of Alice the user and Bob the server. Carol provides Alice with two user names $name_1$,$name_2$. Alice flips a fair coin and gets the bit b. She then registers and authenticates with Bob with the user name $name_b$, and with her choice of password generated using UserGen. Carol can interact with Bob T times. At the end, Carol outputs a bit b'. The adversary wins if $b' = b$. Call her probability of winning $\Pr_W$. Her advantage is then $\Pr_W - 1/2 - \mathsf{ProbGuess}(T)$.

A weaker anonymity notion can be a (T,N)-Anonymity game where Alice registers with her choice of user name with N servers, and Carol learns it after T interactions with each one of the servers (just as the analogy between Mobile Honeypot Security game and Cloud Honeypot Security game: indistinguishability vs. recovery).

4. Cloud SPA constructions

We present three constructions of a Cloud SPA protocol. The *first* construction is very **efficient for the server**, the *second* construction is very **efficient for the storage**, and the *third* construction provides optimal **privacy for the client**.

Our constructions use modified versions of Boyen's hidden credential retrieval protocol [16] as building blocks. During registration, the client generates a strong signature key pair (ssk,svk). The client sends the verification key svk to the online service and an encryption of the signing key ssk to the storage provider. To authenticate, the client retrieves the encrypted signing key, decrypts it, and uses it to sign a challenge generated by the online service.

In our first construction, we employ Boyen's protocol directly, with the slight addition of identifiers. Alice stores an identifier with the storage provider; during Retrieve, and she uses the identifier to tell the storage provider which ciphertext to retrieve. Boyen's Retrieve protocol involves a blind signature operation. In our second construction, we move this blind signature from the storage to the server instead. Finally, our third construction requires an oblivious transfer (OT) or private information retrieval (PIR) protocol during Retrieve.

In terms of privacy, our first construction computes the identifier as a hash of Alice's username and the name of the online service. These are short values that Alice can remember. However, the storage provider can use this identifier to launch a dictionary attack and learn Alice's username at a particular online service (not the password, of course), and thus the protocol is *neither anonymous nor unlinkable*. In our second construction, identifiers are random values stored at the server; the storage cannot learn Alice's username (*anonymous*), but she can still link Store and Retrieve requests (*not unlinkable*). Our third construction uses OT/PIR to provide unlinkability, thus giving Alice complete privacy (*both anonymity and unlinkability*).

All our protocols are provably secure according to the definitions in Section 3.1. Security proofs for our Cloud SPA schemes assume that the digital signature used is existentially unforgeable and its secret key is computationally indistinguishable from random; the blind signature used is secure, unique, and private (i.e. the signer does not learn the signature); the password-based encryption used is secure as per Definition 2; the oblivious transfer scheme that is used is secure; and the hash function is modeled as a Random Oracle. Furthermore, we assume that the communication channels are secure and server-authenticated (e.g., via SSL).

4.1. Server-optimal Cloud SPA

Our first construction (see Fig. 2) is the *most efficient for the server*. The drawback is that it is *neither anonymous nor unlinkable*.

Register: $\{\mathsf{Client}(1^k, name, pwd)\} \rightarrow \{\mathsf{Client}(ssk, bsk), \mathsf{Server}(name, svk)\}$

1. The client computes two key-pairs; one for blind signatures $(bsk, bvk) \leftarrow \mathsf{BSigKeyGen}(1^k)$ and one for digital signatures $(ssk, svk) \leftarrow \mathsf{SigKeyGen}(1^k)$.
2. The client sends $(name, svk)$ to the server. The client keeps (ssk, bsk).

Store: $\{\mathsf{Client}(pwd, bsk, ssk)\} \rightarrow \{\mathsf{Storage}(id, ctext, bsk)\}$

1. The client computes a blind signature value $sig \leftarrow \mathsf{BSign}(bsk, \mathsf{Hash}(pwd))$ and encrypts her secret key ssk using hash of the signature as key $ctext \leftarrow \mathsf{Encrypt}(\mathsf{Hash}(sig), ssk)$.
2. The client computes the identifier $id \leftarrow \mathsf{Hash}(name, servername)$ via a hash function.
3. The client sends the identifier, the encrypted secret signing key, and the blind signature key $(id, ctext, bsk)$ to the storage. At this point, the client may forget all hard-to-remember values (i.e. ssk, bsk, id, $ctext$).

PreAuth: $\{\mathsf{Client}(name, pwd), \mathsf{Server}(\langle name, svk\rangle)\} \rightarrow \{\mathsf{Client}(id, chal), \mathsf{Server}(svk, chal)\}$

1. The client sends *name* to the server.
2. The server sends the client a random challenge $chal \leftarrow \{0,1\}^k$.
3. The client receives the challenge *chal* from the server and computes the identifier $id \leftarrow \mathsf{Hash}(name, servername)$.
4. The server looks up the verification key *svk* associated with *name* in its database and outputs (*svk*, *chal*) to use for authentication.

Retrieve: $\{\mathsf{Client}(id, pwd, chal), \mathsf{Storage}(\langle id, ctext, bsk\rangle)\} \rightarrow \{\mathsf{Client}(response)\}$

1. The client sends the identifier *id* to the storage.
2. The storage looks up the (*ctext*, *bsk*) associated with *id*. The client and the storage execute the blind signature protocol. The storage acts as the signer using *bsk* as the signing key. The client acts as the receiver, using $\mathsf{Hash}(pwd)$ as the message. The client gets a blind signature $sig \leftarrow \mathsf{BSign}(bsk, \mathsf{Hash}(pwd))$ on the hash of her password as its output.
3. The storage sends *ctext* to the client.
4. The client decrypts the ciphertext using the blind signature to obtain her secret signing key $ssk \leftarrow \mathsf{Decrypt}(\mathsf{Hash}(sig), ctext)$ and outputs the response to the challenge $response \leftarrow \mathsf{Sign}(ssk, chal)$.

Authenticate: $\{\mathsf{Client}(response), \mathsf{Server}(svk, chal)\} \rightarrow \{\mathsf{Server}(\mathsf{accept/reject})\}$

1. The client sends the response *response* to the server.
2. The server accepts iff the response verifies using the registered verification key of the client (i.e. $\mathsf{SigVerify}(svk, chal, response) = 1$).

Theorem 2. *Server-optimal Cloud SPA is a secure Cloud SPA scheme.*

Proof. Since the client's interaction with the cloud storage provider during Store and Retrieve is identical to that in Boyen HCR protocol, Cloud Honeypot Security follows. Thus, given an adversary Adv that can defeat the Cloud Honeypot Security of our construction, we can create a reduction that attacks Boyen HCR scheme:

1. The challenger generates an ℓ-bit password *pwd* and a *k*-bit credential *cred*. The challenger simulates a Store request to the Boyen HCR storage provider. The reduction gets $(1^\ell, 1^k)$ as input.
2. To answer a $\mathsf{PlayRegister}(servername, \mathrm{i})$ query of the adversary Adv, the reduction generates a key pair (*ssk*, *svk*). Then, it sends *svk* to the adversary Adv and tells the challenger to store *ssk*. The reduction records *ssk*.
3. To answer a $\mathsf{PlayPreAuth}(servername)$ query of Adv, the reduction sends *name* to Adv. The adversary Adv returns a challenge *chal*, which the reduction stores.
4. To answer a $\mathsf{PlayAuthenticate}(sessionid, \mathrm{i})$ query, the reduction returns $sig \leftarrow \mathsf{Sign}(ssk, chal)$ to the adversary.
5. To answer a $\mathsf{PlayRetrieve}(i)$ query, the reduction simply passes messages between the adversary Adv and the challenger.
6. Eventually, Adv returns a guess *pwd*′ and the reduction uses it as its output.

The reduction succeeds with the same probability as the adversary, which by Theorem 1 is $\mathsf{ProbGuess}(TN + 1) + neg(k)$.

Cloud Storage Security follows immediately from Theorem 1. □

Privacy: The cloud storage provider can mount a dictionary attack on $id \leftarrow \mathsf{Hash}(name, servername)$ to learn Alice's username associated with a server (using a *name* and

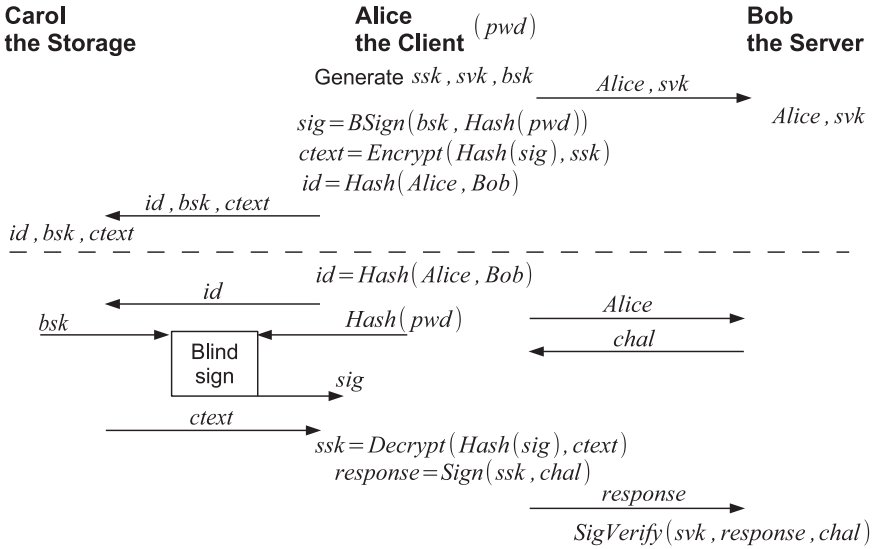

Fig. 2. Server-optimal Cloud SPA.

servername dictionary instead of a password dictionary). Such attacks are realistic (see an attack on MicroID where the hash of the user's email was attacked [28]) but can be avoided if Alice chooses a random *id* during registration, and retrieves it from the online service during PreAuth, as we will see in our second construction.

4.2. Storage-optimal Cloud SPA

This version of our protocol (see Fig. 3) is the *most efficient for the cloud storage provider*, and provides *anonymity* (while still being linkable). The main difference is that the server stores the blind signature key *bsk* and performs the blind signature operation instead of the storage provider.

Register: $\{\mathsf{Client}(1^k, name, pwd)\} \rightarrow \{\mathsf{Client}(ssk, bsk), \mathsf{Server}(name, svk, bsk)\}$

1. The client computes two key-pairs; one for blind signatures $(bsk, bvk) \leftarrow \mathsf{BSigKeyGen}(1^k)$ and one for digital signatures $(ssk, svk) \leftarrow \mathsf{SigKeyGen}(1^k)$
2. The client sends $(name, svk, bsk)$ to the server. The client's private output is (ssk, bsk).

Store: $\{\mathsf{Client}(pwd, bsk, ssk)\} \rightarrow \{\mathsf{Storage}(id, ctext)\}$

1. The client sets the identifier $id \leftarrow \mathsf{BSign}(bsk, \mathsf{Hash}(pwd))$ by simulating the blind signing protocol using her own knowledge of *bsk*.
2. The client encrypts her key using her password $ctext \leftarrow \mathsf{Encrypt}(\mathsf{Hash}(pwd), ssk)$.
3. The client sends $(id, ctext)$ to the storage. At this point, the client may forget all hard-to-remember values (i.e. $ssk, bsk, id, ctext$).

PreAuth: $\{\mathsf{Client}(name, pwd), \mathsf{Server}(\langle name, svk, bsk\rangle)\} \rightarrow \{\mathsf{Client}(id, chal), \mathsf{Server}(svk, chal)\}$

1. The client sends *name* to the server.
2. The server looks up the (svk, bsk) associated with the *name* in its database. The client and the server then execute the blind signature protocol. The server acts as the signer using *bsk* as the blind signing key. The client acts as the receiver, using $\mathsf{Hash}(pwd)$ as the message. The client gets the identifier $id \leftarrow \mathsf{BSign}(bsk, \mathsf{Hash}(pwd))$ as its output.
3. The server sends the client a random challenge $chal \leftarrow \{0,1\}^k$.
4. The client remembers $(id, chal)$. The server remembers $(svk, chal)$.

Retrieve: $\{\mathsf{Client}(id, pwd, chal), \mathsf{Storage}(\langle id, ctext\rangle)\} \rightarrow \{\mathsf{Client}(response)\}$

1. The client sends the identifier *id* to the storage.
2. The storage looks up the *ctext* associated with the *id* in its database and sends *ctext* to the client.
3. The client computes the decryption of the ciphertext to obtain her secret key $ssk \leftarrow \mathsf{Decrypt}(\mathsf{Hash}(pwd), ctext)$ and computes the response $response \leftarrow \mathsf{Sign}(ssk, chal)$.

Authenticate: Same as in Server-optimal Cloud SPA (the client sends the response to the server, the server verifies it with *svk*).

Theorem 3. *Storage-optimal Cloud SPA is a secure Cloud SPA scheme.*

Proof. Cloud Honeypot Security is proven by reduction to Boyen's HCR protocol, similar to the previous protocol. Thus, given an adversary Adv that can defeat the Cloud Honeypot Security of our second construction, we can create a reduction that attacks Boyen HCR scheme:

1. The challenger generates an ℓ-bit password *pwd* and a *k*-bit credential *cred*. The challenger simulates a Store request to the Boyen HCR storage provider. The reduction gets $(1^\ell, 1^k)$ as input.
2. To answer a PlayRegister(*servername*,i) query of the adversary Adv, the reduction generates two key pairs (ssk, svk, bsk, bvk) as usual. It then sends (svk, bsk) to the Adv, and tells the challenger to store *ssk*. The reduction records *ssk*.
3. To answer a PlayPreAuth(*servername*) query, the reduction executes the blind signature protocol with Adv using Hash(1) as its message. The reduction records the $(id, chal)$ that it gets from the Adv. Due to the properties of a blind signature scheme, Adv cannot tell that the reduction uses Hash(1) as its input instead of Hash(*pwd*) (otherwise the reduction may use the adversary to break the security of the blind signature scheme).
4. To answer a PlayAuthenticate(*sessionid*,*i*) query, the reduction returns $sig \leftarrow \mathsf{Sign}(ssk, chal)$ to the adversary.
5. To answer a PlayRetrieve(i) query, the reduction asks the challenger to execute the Retrieve protocol using "1" as its password. The reduction forwards the *ctext* it obtains from the storage to the adversary.
6. Eventually, Adv returns a guess pwd' and the reduction uses it as its output.

It is clear that the reduction wins whenever the adversary wins. Due to Theorem 1, the reduction may succeed with probability $\mathsf{ProbGuess}(TN + 1) + neg(k)$.

Cloud Storage Security follows trivially from Boyen [16] since the storage provider gets even less information than the Boyen storage provider.

Privacy: Storage-optimal Cloud SPA is anonymous; the storage provider does not learn any information about Alice's username, since the identifier is chosen at random. However, it is not unlinkable because Alice uses the same identifier for every Store and Retrieve request.

4.3. Privacy-optimal Cloud SPA

We can provide both *anonymity* and *unlinkability* by employing oblivious transfer (OT) or private information retrieval (PIR) techniques [8,19,20,29,41,24,11,47,42], in particular, the Oblivious Keyword Search [51] that allows efficient OT with any keyword as an index rather than only consecutive integers. Privacy-optimal Cloud SPA (see

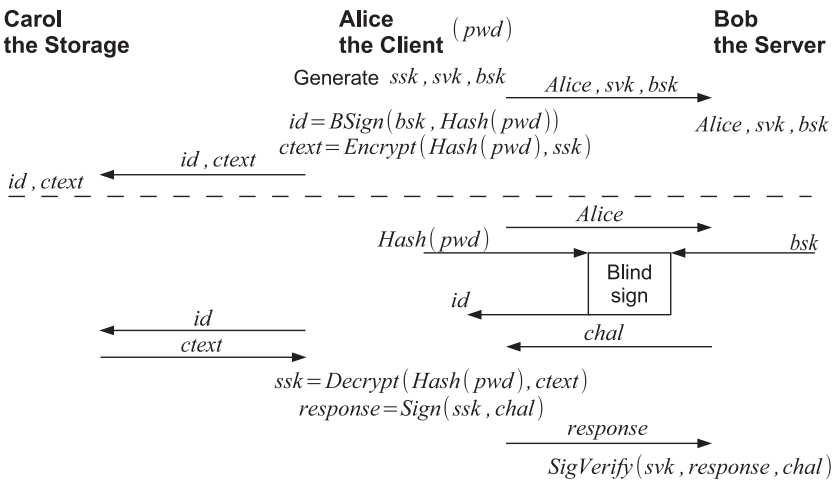

Fig. 3. Storage-optimal Cloud SPA.

Fig. 4) is identical to Storage-optimal Cloud SPA, except that during Retrieve, the client and storage provider execute a PIR/OT. As a result, Alice learns her *ctext* without revealing her *id*. This prevents the storage provider from linking Store and Retrieve requests (and even two Retrieve requests), hence providing **both anonymity and unlinkability**.[3]

Theorem 4. *Privacy-optimal Cloud SPA is a secure Cloud SPA scheme.*

Proof. The security proof follows trivially from Storage-optimal Cloud SPA proof, whereas the anonymity and unlinkability directly follow from the properties of oblivious transfer, and hence a full proof is omitted. □

5. Mobile SPA construction

Our mobile SPA construction requires the ability to establish a channel between the online service and the storage provider (i.e. the mobile device). However, we want to minimize setup and assumptions about the hardware on the mobile device. We require some of the following input/output capabilities from the mobile device:

Local human output: Capability to output a small amount of information to the user. Examples include outputting a 5–8 character value through a display or a speaker.
Local human input: Capability to obtain a small amount of information from the user. Examples include Alice inputting a 5–8 character value through a keypad or a touchpad.
Local machine input: Capability to obtain a larger amount of information (i.e. *k*-bits, which would be hard for a human to input) from another machine. For example, the device could use its camera to scan and decode a barcode displayed on Alice's monitor, or use its microphone to record and analyze audio, or connect to Alice's computer through Bluetooth/Wi-fi/USB.
Remote machine input: Capability to obtain a larger amount of information (i.e. *k*-bits, which would be hard for a human to input) from another machine. This can be through a direct Internet connection, an indirect Internet connection (through Bluetooth/Wi-fi/USB), or SMS.

Our mobile construction requires *local human input and output* capabilities from the mobile device. We additionally require *local* or *remote* machine input capabilities.

A naive solution for the mobile SPA protocol would be for Alice to lock her mobile device with a password/pattern/PIN, and then store her login password (even in an encrypted fashion) on it. However, if an adversary captures her mobile device, the login password is easily leaked, even when encrypted by the device. Recent research on popular smartphones with popular operating systems shows that an attack can recover the stored data within as little as 6 min [36,2].

In our solution, the mobile device stores *ssk* encrypted using Alice's persistent password as the encryption key, *without* relying on any PIN or similar mechanism to lock the device. During authentication, Alice enters her password via some local human input mechanism, and forwards the online service's challenge via some local or remote machine input mechanism. The device performs the necessary computation to respond to the challenge, and then *erases* the password once authentication is complete. In the case the device is stolen and the robber is *not* Bob, Alice's password and the secrets are not disclosed. If Bob manages to steal the device, then it is as if the storage provider and the server are colluding, and hence we cannot protect against dictionary attacks (see Section 7 for possible measures against collusion).

We will not worry about the use of identifiers in the mobile SPA protocols, as Alice is the only user of her mobile device; she can select the appropriate account on the

[3] Anonymous communication techniques such as TOR [27] should be employed on top of our scheme to provide even IP-level anonymity and unlinkability.

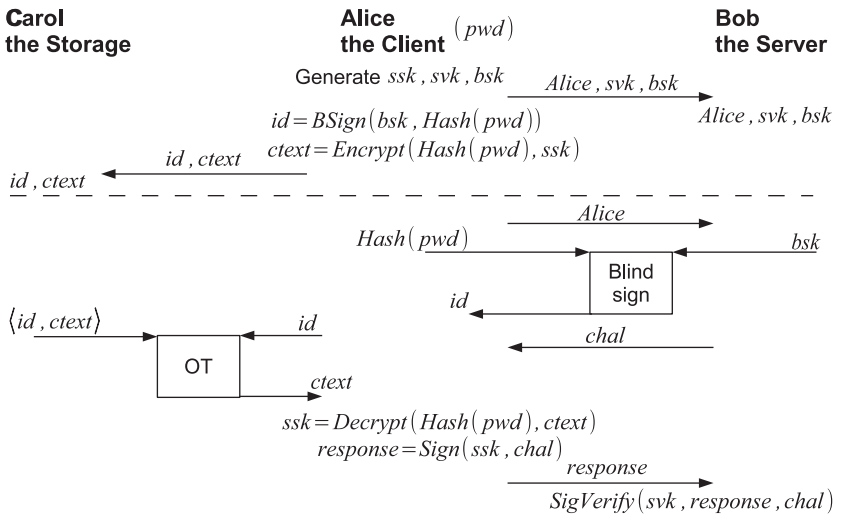

Fig. 4. Privacy-optimal Cloud SPA.

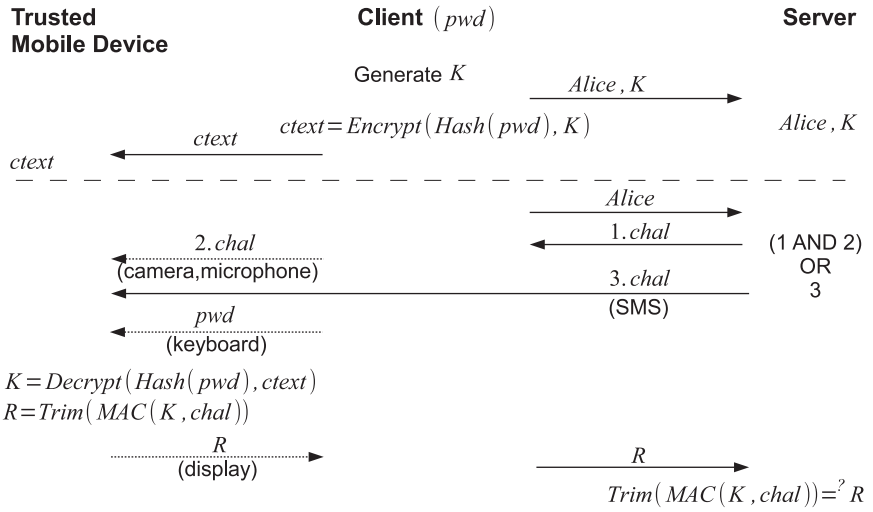

Fig. 5. Mobile SPA flavors.

device directly. Alternatively, Alice may pick a random *id* for every server, send it to the server during registration, and then during authentication the online service could prepend an *id* to the challenge and the mobile device could use it to look up the appropriate stored ciphertext. This way, **anonymity** of Alice is protected even when the device is stolen, since identifiers are random values (and unlinkability is not necessary in the mobile SPA scenario).

All flavors of our protocol are provably secure according to the definitions in Section 3.1. Security proofs for our mobile SPA schemes assume that the MAC scheme used is unforgeable and its secret keys are computationally indistinguishable from random; the password-based encryption used is secure as per Definition 2; and the hash function is modeled as a Random Oracle. Furthermore, we assume that the communication channels are secure and authenticated (e.g., SSL connection between the client and the server, and physical security of the client-device interaction).

Our construction, shown in Fig. 5, assumes the helper device has **local human input and output, and local or remote machine input capabilities**. During PreAuth, the online service sends the client a challenge. This can be done via the device's camera or microphone, or even using SMS since *the communication channel between the mobile device and the server does not need to be secure*. The device MACs the challenge using Alice's secret key *ssk* to generate a response *response*′. Then the device calls a function $\mathsf{Trim}(1^{m \ll k}, response')$ to abbreviate the MAC from the usual k bits to m bits, and outputs the result in human-friendly form (e.g., displays m-bits as 5–8 characters). Alice uses this trimmed response as her one-time password. As long as the online service limits the number of guesses a user

Table 1
Performance upper bounds for our various proposals during the *authentication phase*. Computation refers to the total time spent (in ms), and communication refers to the total amount sent (in bytes), throughout the authentication phase.

	Server-optimal		Storage-optimal		Mobile	
	Comp.	Comm.	Comp.	Comm.	Comp.	Comm.
Alice the client	7	100	6	100	0	25
Bob the server	1	10	2.5	30	1	10
Carol the storage	1.5	36	0	16	2	0

can make to answer a particular challenge, **protection against even active real-time attacks** is provided.

Register: $\{\mathsf{Client}(1^k, name, pwd)\} \rightarrow \{\mathsf{Client}(K), \mathsf{Server}(name, K)\}$

1. The client computes a MAC key $K \leftarrow \mathsf{MACKeyGen}(1^k)$.
2. The client sends $(name, K)$ to the server. The client keeps for storage K.

Store: $\{\mathsf{Client}(pwd, K)\} \rightarrow \{\mathsf{Storage}(ctext)\}$

1. The client sends the encrypted MAC key $ctext \leftarrow \mathsf{Encrypt}(\mathsf{Hash}(pwd), K)$ to the storage, and forgets it.

PreAuth: $\{\mathsf{Client}(name, pwd), \mathsf{Server}(\langle\ name, K\rangle)\} \rightarrow \{\mathsf{Client}(chal), \mathsf{Server}(K, chal)\}$

1. The client sends *name* to the server.
2. The server looks up the key K associated with the *name* in its database. It picks a random challenge $chal \leftarrow \{0,1\}^k$, and sends it to the client as readable by the mobile device (e.g., a barcode image is displayed on the client's screen or an audio file is played using the client's speakers). Alternatively, the challenge may be sent to the registered mobile device of the user directly (e.g., via SMS).
3. The server remembers $(K, chal)$ while the client obtains *chal*.

Retrieve: $\{\mathsf{Client}(pwd, chal), \mathsf{Storage}(ctext)\} \rightarrow \{\mathsf{Client}(response)\}$

1. If the remote input mechanism was not directly used in PreAuth step (e.g., the mobile device did not receive the challenge directly in an SMS), then the user provides the storage device with the challenge *chal* via the local machine input mechanism (e.g., the user takes a photo of the barcode image on the client's screen using the camera of the mobile device, or records the audio file the client is playing using the microphone of the mobile device, who then recognizes the challenge embedded into that image or audio).
2. The user provides the storage device with the password *pwd* using the local human input mechanism (e.g., types in via the mobile device's hardware/touch keyboard).
3. The storage device recovers the key $K \leftarrow \mathsf{Decrypt}(\mathsf{Hash}(pwd), ctext)$. It computes the response $response' \leftarrow \mathsf{MAC}(K, chal)$, and trims it to get the short one-time password $response = \mathsf{Trim}(1^{m \ll k}, response')$. Finally, it outputs *response* in human-friendly form using local human output mechanism (e.g., displays it on the screen).

Authenticate: $\{\mathsf{Client}(response), Server(K, chal)\} \rightarrow \{\mathsf{Server}(\mathsf{accept/reject})\}$

1. The user types *response* into the client (e.g., the user reads the response displayed on the mobile device, and types it exactly to the client), which sends it over to the server.
2. The server accepts iff $\mathsf{Trim}(1^{m \ll k}, \mathsf{MAC}(K, chal)) = response$.

Theorem 5. *Mobile SPA is a secure mobile SPA scheme.*

Proof. Mobile Honeypot Security is trivial since the view of the online service is completely *independent* of the password *pwd*.

Mobile Storage Security follows from unforgeability of MACs and security of the password-based encryption scheme. Once the mobile device is corrupted, all it has is a (database of) ciphertext values $ctext = \mathsf{Encrypt}(\mathsf{Hash}(pwd), K)$. If the adversary is able to create a valid MAC on the challenge *chal*, then it either has forged a MAC or violated the security of the password-based encryption scheme. We omit the full reduction in consideration for space. □

6. Performance analysis

In this section, we provide a performance analysis using approximate performance numbers. Our schemes employ standard primitives (such as AES, DSA, and SHA-1 [25,50,48]), all of whom operate under 1 ms, even on mobile devices[4]. The most expensive parts of our schemes are the blind signature operations as well as the oblivious transfer.

López-García et. al. [43] implemented Boldyreva blind signatures [10] on a 2 GHz machine. For the blind signature protocol run jointly between Alice and Bob/Carol, they note that Alice needs to spend about 3 ms in total and the signer (Bob or Carol depending on our protocol) needs to spend about 1.5 ms. Table 1 presents the total computation and communication costs using the numbers presented above.[5] Note that the 1 ms performance for standard primitives is definitely over-counting—especially since the servers have better processors, and thus the table

[4] We are encrypting 128 or 160 bytes of data, and a 450 MHz processor can achieve about 0.15 microseconds per byte performance [58].

[5] In the Mobile SPA scenario, if Bob sends a QR code, his communication is about 2 KB, but Carol still need not employ any network communication.

should be seen as an upper bound on the performance. Since registration is a one-time process, we emphasize the performance for the authentication phase.

As for storage requirements, observe that the client never needs to store any long-term information in any of our protocols. This is necessary for Alice to be able to use different computers to access her accounts. For the server, clearly any server already needs to store a database of usernames and passwords (or hashes of passwords). Our Server-Optimal Cloud SPA protocol requires the server to store a digital signature signing key (128 bytes for DSA if everybody use the same group), and our Mobile SPA protocol require a MAC key to be stored, instead of the password. Our Storage-Optimal and Privacy-Optimal Cloud SPA protocols require the server to store an additional blind signature signing key (about 115 bytes for Boldyreva blind signature [10,12,13,43]). The cloud storage provider needs to store a database of identifiers, ciphertexts, and possibly blind signature signing keys (for the server-optimal protocol). To give a ballpark upper-bound, assume the server and the storage each store 300 bytes per user. Then for 10 million users, they each need 2.8 GB of data, which can be easily handled using today's storage capabilities (even in RAM).

Finally, note that in our Mobile SPA construction, Alice should not reuse the same key K for different servers, otherwise any such server can impersonate Alice against other servers. Therefore, this construction requires storage on the mobile device linear in the number of servers. If linear storage is not desired, then one may only store a pseudorandom function key at the device, and then use the (*name*,*servername*) pair as input to the pseudorandom function to re-generate the same MAC key every time that is needed. This presents a trade-off between storage and computation. Yet, since a MAC key is 128 bytes, then **with just 1 MB of storage**, the mobile device can store authentication information for **more than 8000 servers** (i.e. websites). Therefore, using current standard mobile devices, *linear storage is not an issue*.

7. Extensions and conclusion

We presented a complete system for single-password authentication, and provided multiple flavors of our system under different performance-privacy-usability considerations. Our solutions deal with server/storage efficiency, and client privacy, as well as mobile-device capability issues.

Our schemes thwart honeypot attacks since the server can no longer learn the client's password, or any deterministic function of it. Similarly, a phishing website will not be able to obtain the user's password (though may still obtain credit card information). Further measures against phishing may rely on combining our mobile SPA protocols with some phishing prevention protocols working on mobile devices [52,46].

Moreover, **malware or malicious code damage is minimized** using our schemes. In our Cloud SPA model, malicious code may lead to leakage of Alice's password. But using our mobile SPA scheme, *only the session information is leaked*. Thus, Alice's long-term secrets (e.g., password) remains safe using a mobile helper device, even when she does not trust the PC she is using during authentication (e.g., **using a public terminal**), and **even when the mobile device is stolen**. The best an *active* adversary can do is to gain control of the session, but no future sessions. An adversary that is *passive* during the session learns no useful information.

To protect Alice from such attacks, we *must* enlist the aid of the browser and/or operating system (OS). Just as modern operating systems present the user with an OS-generated window warning about potentially dangerous interactions, the browser and/or OS should obtain Alice's password in a secure window and run the SPA protocol on her behalf. This requires a change to the browser code or a plug-in. But unlike previous browser extensions [54,34,61], our protocols provide provable security against dictionary attacks.

In the mobile SPA setting, indeed a one-time password is given to the browser instead of the long-term password to limit the risk. In cases where TLS is not feasible or the user is fooled into connecting to an impersonating server even over TLS, and assuming the sessions are short-lived compared to the time it takes to crack this one-time password, the adversary is forced to deploy an active real-time attack, or the adversary must store all the encrypted session information and perform an offline attack. Even under such an attack, **mobile SPA protects Alice's long-term secret and password**, and prevents the adversary from impersonating Alice. Note that *using a one-time password together with the password on the same device*, as in many current scenarios (e.g., Internet banking), still may *leak the password* on an infected device; that is why Alice never enters her password on an untrusted device in our system. Furthermore, in most current systems, the one-time password is sent as an SMS in clear, which means an *active* attacker is almost guaranteed to succeed. In our system, only the challenge is sent in clear, and the one-time password is constructed using the password, which means even an active attacker must keep guessing it online. When the mobile SPA is integrated with the browser, rather than being part of a potentially malicious web page code, it protects the per session secret as well.

Our constructions assume that *the online service and the storage provider do not collude* (e.g., they are not operated by the same company). Otherwise, as shown by Boyen [16], it is impossible to prevent a dictionary attack. Alice can **protect herself from collusion of the storage and the server** by using threshold secret-sharing schemes [57,9], or password-protected secret-sharing schemes [3], to employ multiple storage providers. The basic idea is to split her secret key ssk (or K) into m secrets ssk_i and store each one at a different storage provider (operated by a different company) in encrypted form.[6] Alice can successfully authenticate as long as a (k out of n) quorum of storage providers are available, and now such a quorum would

[6] As for the secret keys, the secret shares themselves should be random values, and this is the case in popular secret-sharing schemes [57].

need to collude with the online service in order to learn Alice's password. For example, the storage is distributed among Google, Microsoft, Facebook, Amazon, and Yahoo, then we expect that collusion does not happen in practice, since now these companies must collude together against the user.

Another important property of our protocol is that, each client can choose her own security parameter and cryptographic primitive employed. For example, Alice can decide to share her signing key among 10 storage servers and use AES-256, whereas Amanda can decide that it is enough to use a single storage and AES-128. We expect to see a variety of online services and storage providers offering different levels of security (e.g., stronger security and privacy level for important services like banking).

Our constructions minimize the amount of modification that needs to be made to existing online services. While a service needs to execute some novel code to achieve SPA, it can be performed at the javascript/CGI script level. All subsequent communication proceeds as usual. Moreover, Alice's view in terms of her login experience need not change in the Cloud SPA setting, possibly with some help from browsers. Furthermore, if Kerberos-like token-based (possibly anonymous) credential schemes or single-signon services (where authentication and service providers are separate [37]) are used, then **only the client and the authentication software need to be modified, while the server software may remain intact**. SPA would run only between the user and the Kerberos server leaving the rest of the protocol unmodified. Such services that are widely-used today include Windows Active Directory services running at enterprise networks, Facebook accounts used to login to many other sites, and Open ID (e.g., Google accounts).

Considering all the benefits of our constructions (**provable security against dictionary attacks and honeypots, anonymity and unlinkability measures, mobility, extra protection against malware and phishing**) as well as relative **ease of deployment** as discussed above, we truly hope that our schemes will be available soon as browser extensions, mobile phone applications, and implemented on popular single-signon services such as Microsoft Passport, Google Accounts, and Facebook.

References

[1] M. Abdalla, E. Bresson, O. Chevassut, B. Möller, D. Pointcheval, Provably secure password-based authentication in tls, in: ASIACCS, 2006.

[2] A.J. Aviv, K. Gibson, E. Mossop, M. Blaze, J.M. Smith, Smudge attacks on smartphone touch screens, in: WOOT, 2010.

[3] A. Bagherzandi, S. Jarecki, N. Saxena, Y. Lu, Password-protected secret sharing, in: ACM CCS, 2011.

[4] M. Belenkiy, M. Chase, M. Kohlweiss, A. Lysyanskaya, P-signatures and noninteractive anonymous credentials, in: R. Canetti (Ed.), TCC 2008: 5th Theory of Cryptography Conference, San Francisco, CA, USA, March 19–21, 2008, vol. 4948, Lecture Notes in Computer Science, Springer, Berlin, Germany, 2008, pp. 356–374.

[5] M. Bellare, D. Pointcheval, P. Rogaway, Authenticated key exchange secure against dictionary attacks, in: B. Preneel (Ed.), Advances in Cryptology – EUROCRYPT 2000, Bruges, Belgium, May 14–18, 2000, Lecture Notes in Computer Science, vol. 1807, Springer, Berlin, Germany, 2000, pp. 139–155.

[6] S.M. Bellovin, M. Merritt, Encrypted key exchange: password-based protocols secure against dictionary attacks, in: 1992 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, 1992, pp. 72–84.

[7] S.M. Bellovin, M. Merritt, Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password file compromise, in: V. Ashby (Ed.), ACM CCS 93: 1st Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3–5, 1993, ACM Press, 1993, pp. 244–250.

[8] C.H. Bennett, G. Brassard, C. Crépeau, M.-H. Skubiszewska, Practical quantum oblivious transfer, in: J. Feigenbaum (Ed.), Advances in Cryptology – CRYPTO '91, Santa Barbara, CA, USA, August 11–15, 1992, Lecture Notes in Computer Science, vol. 576, Springer, Berlin, Germany, 1992, pp. 351–366.

[9] G.R. Blakley, Safeguarding cryptographic keys, in: NCC, 1979.

[10] A. Boldyreva, Threshold signatures, multisignatures and blind signatures based on the gap-Diffie–Hellman-group signature scheme;;, in: Y. Desmedt (Ed.), PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, USA, January 6–8, 2003, Lecture Notes in Computer Science, vol. 2567, Springer, Berlin, Germany, 2003, pp. 31–46.

[11] D. Boneh, E. Kushilevitz, R. Ostrovsky, W.E. Skeith III, Public key encryption that allows PIR queries, in: A. Menezes (Ed.), Advances in Cryptology – CRYPTO 2007, Santa Barbara, CA, USA, August 19–23, 2007, Lecture Notes in Computer Science, vol. 4622, Springer, Berlin, Germany, 2007, pp. 50–67.

[12] D. Boneh, B. Lynn, H. Shacham, Short signatures from the Weil pairing, in: C. Boyd (Ed.), Advances in Cryptology – ASIACRYPT 2001, Gold Coast, Australia, December 9–13, 2001, Lecture Notes in Computer Science, vol. 2248, Springer, Berlin, Germany, 2001, pp. 514–532.

[13] D. Boneh, B. Lynn, H. Shacham, Short signatures from the Weil pairing, Journal of Cryptology 17 (4) (2004) 297–319.

[14] J. Bonneau, S. Preibusch, The password thicket: technical and market failures in human authentication on the web, in: WEIS, 2010.

[15] M.K. Boyarsky, Public-key cryptography and password protocols: the multi-user case, in: ACM CCS, 1999.

[16] X. Boyen, Hidden credential retrieval from a reusable password, in: W. Li, W. Susilo, U.K. Tupakula, R. Safavi-Naini, V. Varadharajan (Eds.), ASIACCS 09: 4th Conference on Computer and Communications Security, Sydney, Australia, March 10–12, 2009, ACM Press, 2009, pp. 228–238.

[17] X. Boyen, Hpake: password authentication secure against cross-site user impersonation, in: CANS 2009: Conference on Cryptology and Network Security, Springer-Verlag, Berlin Heidelberg, 2009, pp. 279–298.

[18] S.A. Brands, Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy, MIT Press, Cambridge, MA, USA, 2000.

[19] G. Brassard, C. Crépeau, J.-M. Robert, All-or-nothing disclosure of secrets, in: A.M. Odlyzko (Ed.), Advances in Cryptology – CRYPTO'86, Santa Barbara, CA, USA, August 1987, Lecture Notes in Computer Science, vol. 263, Springer, Berlin, Germany, 1987, pp. 234–238.

[20] C. Cachin, S. Micali, M. Stadler, Computationally private information retrieval with polylogarithmic communication, in: J. Stern (Ed.), Advances in Cryptology – EUROCRYPT'99, Prague, Czech Republic, May 2–6, 1999, Lecture Notes in Computer Science, vol. 1592, Springer, Berlin, Germany, 1999, pp. 402–414.

[21] J. Camenisch, A. Lysyanskaya, An efficient system for non-transferable anonymous credentials with optional anonymity revocation, in: B. Pfitzmann (Ed.), Advances in Cryptology – EUROCRYPT 2001, Innsbruck, Austria, May 6–10, 2001, Lecture Notes in Computer Science, vol. 2045, Springer, Berlin, Germany, 2001, pp. 93–118,.

[22] N. Carlson, in: 2004, mark zuckerberg broke into a facebook user's private email account. Business Insider, March 2010. <http://www.businessinsider.com/how-mark-zuckerberg-hacked-into-the-harvard-crimson-2010-3>.

[23] D. Chaum, Showing credentials without identification: signatures transferred between unconditionally unlinkable pseudonyms, in: F. Pichler (Ed.), Advances in Cryptology – EUROCRYPT'85, Linz, Austria, April 1986, Lecture Notes in Computer Science, vol. 219, Springer, Berlin, Germany, 1986, pp. 241–244.

[24] B. Chor, E. Kushilevitz, O. Goldreich, M. Sudan, Private information retrieval, Journal of ACM 45 (6) (1998) 965–981.

[25] J. Daemen, V. Rijmen, The Design of Rijndael: AES – the Advanced Encryption Standard, Springer, 2002.

[26] W. Diffie, M.E. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (6) (1976) 644–654.
[27] R. Dingledine, N. Mathewson, P. Syverson, Tor: the second-generation onion router, in; USENIX Security, 2004.
[28] C.C. Erway, Microid considered Harmful (to Privacy), Technical Report CS-08-09, Brown University, 2008.
[29] S. Even, O. Goldreich, A. Lempel, A randomized protocol for signing contracts, in: D. Chaum, R.L. Rivest, A.T. Sherman (Eds.), Advances in Cryptology – CRYPTO'82, Santa Barbara, CA, USA, 1983, Plenum Press, New York, USA, 1983, pp. 205–210.
[30] D. Florencio, C. Herley, A large-scale study of web password habits, in: WWW '07: Proceedings of the 16th international conference on World Wide Web, New York, NY, USA, 2007, ACM, 2007, pp. 657–666.
[31] W. Ford, B.S. Kaliski Jr., Server-assisted generation of a strong secret from a password, in: WETICE '00: Proceedings of the 9th IEEE International Workshops on Enabling Technologies, Washington, DC, USA, 2000, IEEE Computer Society, 2000, pp. 176–180.
[32] C. Gentry, P. MacKenzie, Z. Ramzan, A method for making password-based key exchange resilient to server compromise, in: C. Dwork (Ed.), Advances in Cryptology – CRYPTO 2006, Santa Barbara, CA, USA, August 20–24, 2006, Lecture Notes in Computer Science, vol. 4117, Springer, Berlin, Germany, 2006, pp. 142–159.
[33] M.G. Gouda, A.X. Liu, L.M. Leung, M.A. Alam, Spp: an anti-phishing single password protocol, Computer Networks 51 (13) (2007) 3715–3726.
[34] J.A. Halderman, B. Waters, E.W. Felten, A convenient method for securely managing passwords, in: WWW, 2005.
[35] S. Halevi, H. Krawczyk, Public-key cryptography and password protocols, ACM TISSEC 2 (1999) 230–268.
[36] J. Heider, M. Boll, Lost iphone? Lost Passwords! Practical Consideration of IOS Device Encryption Security. Technical Report, Fraunhofer Institute for Secure Information Technology, 2011.
[37] IETF. RFC 5849, The OAuth 1.0 Protocol, April 2010.
[38] Imperva, Consumer Password Worst Practices, 2010.
[39] D.P. Jablon, W. Ma, Strong password-only authenticated key exchange, ACM Computer Communications Review 26 (1996) 5–26.
[40] J. Katz, Y. Lindell, Introduction to Modern Cryptography, Chapman & Hall/CRC, 2007.
[41] E. Kushilevitz, R. Ostrovsky, Replication is NOT needed: SINGLE database, computationally-private information retrieval, in: 38th Annual Symposium on Foundations of Computer Science, Miami Beach, Florida, October 19–22, 1997, IEEE Computer Society Press, 1997, pp. 364–373.
[42] H. Lipmaa, New communication-efficient oblivious transfer protocols based on pairings, in: T.-C. Wu, C.-L. Lei, V. Rijmen, D.-T. Lee (Eds.), ISC 2008: 11th International Conference on Information Security, Taipei, Taiwan, September 15–18, 2008, Lecture Notes in Computer Science, vol. 5222, Springer, Berlin, Germany, 2008, pp. 441–454.
[43] L. López-García, L. Martínez-Ramos, F. Rodríguez-Henríquez, A comparative performance analysis of several blind signature schemes, in: CCE, 2008.
[44] A. Lysyanskaya, R.L. Rivest, A. Sahai, S. Wolf, Pseudonym systems, in: H.M. Heys, C.M. Adams (Eds.), SAC 1999: 6th Annual International Workshop on Selected Areas in Cryptography, Kingston, Ontario, Canada, August 9–10, 2000, Lecture Notes in Computer Science, vol. 1758, Springer, Berlin, Germany, 2000, pp. 184–199.
[45] P.D. MacKenzie, T. Shrimpton, M. Jakobsson, Threshold password-authenticated key exchange, in: M. Yung (Ed.), Advances in Cryptology – CRYPTO 2002, Santa Barbara, CA, USA, August 18–22, 2002, Lecture Notes in Computer Science, vol. 2442, Springer, Berlin, Germany, 2002, pp. 385–400,.
[46] M. Mannan, P. Van Oorschot, Using a personal device to strengthen password authentication from an untrusted computer, in: FC, 2007.
[47] R. Meier, B. Przydatek, On robust combiners for private information retrieval and other primitives, in: C. Dwork (Ed.), Advances in Cryptology – CRYPTO 2006, Santa Barbara, CA, USA, August 20–24, 2006, Lecture Notes in Computer Science, vol. 4117, Springer, Berlin, Germany, 2006, pp. 555–569,.
[48] NIST. Secure Hash Standard (sha). FIPS PUB 180-1, 1995.
[49] NIST. The Keyed-Hash Message Authentication Code (hmac). FIPS PUB 198, 2002.
[50] NIST. Digital Signature Standard (dss). FIPS PUB 186-3, 2009.
[51] W. Ogata, K. Kurosawa, Oblivious keyword search, Journal of Complexity 20 (2-3) (2004) 356–371. <http://eprint.iacr.org/2002/182/>.
[52] B. Parno, C. Kuo, A. Perrig, Phoolproof phishing prevention, in: FC, 2006.
[53] G.B. Purdy, A high security log-in procedure, ACM Communications 17 (1974) 442–445.
[54] B. Ross, C. Jackson, N. Miyake, D. Boneh, J.C. Mitchell, Stronger password authentication using browser extensions, in: USENIX Security, 2005.
[55] R. Sandhu, M. Bellare, R. Ganesan, Password-enabled pki: virtual smartcards versus virtual soft tokens, in: 1st Annual PKI Research Workshop, 2002, pp. 89–96.
[56] M.J. Schwartz, Rsa launches database breach prevention tool, October 2012. <https://www.informationweek.com/security/encryption/rsa-launches-database-breach-prevention/240008730>.
[57] A. Shamir, How to share a secret, ACM Communications 22 (11) (1979) 612–613.
[58] S. Traboulsi, M. Sbeiti, D. Szczesny, A. Showk, A. Bilgic, High-performance and energy-efficient sliced aes multi-block encryption for lte mobile devices, in: ICCSN, 2011.
[59] T. Wu, The secure remote password protocol, in: In Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium, 1998, pp. 97–111.
[60] Y. Yang, F. Bao, Enabling use of single password over multiple servers in two-server model, in: IEEE CIT, 2010.
[61] K.-P. Yee, K. Sitaker, Passpet: convenient password management and phishing protection, in: SOUPS, 2006.

Tolga Acar earned his BS and MS degrees in Computer Engineering and Computer Science from Istanbul Technical University in 1992 and 1994, and his PhD in Computer Engineering from Oregon State University in 1998. He works in cryptography, operating system and network security, and high-performance arithmetic. He worked at Novell on various cryptographic infrastructures, key management systems, authentication and authorization components, network security, and trusted systems. At Microsoft, he worked on cryptography, policy languages, operating system and network security in Windows and hosted services, certification programs, cryptography and security research, U-Prove, and other security research areas. Dr. Acar is a member of International Association for Cryptologic Research (IACR) and a senior IEEE member. He participated in IEEE P1363 public key cryptography standards, IETF, and NIST security and cryptography standardization efforts. He is now a security architect at Intel Corporation.

Mira Belenkiy graduated from the Department of Computer Science at Brown University in 2008 with her Ph.D. degree. Her work focuses on finding cryptographic solutions to problems in privacy and security. Her recent work includes anonymous credentials, electronic cash, fair exchange, secret sharing, and privacy and accountability in peer-to-peer networks. She has worked at the cryptography incubation team at Microsoft Research. She has been awarded a Department of Homeland Security Fellowhip in 2004.

Alptekin Küpçü has received his Ph.D. degree from Brown University Computer Science Department in 2010. Since then, he has been working as an assistant professor at Koç University College of Engineering, and leading the Cryptography, Security & Privacy Research Group he has founded. His research mainly focuses on applied cryptography, and its intersection with cloud security, privacy, peer-to-peer networks, game theory, and mechanism design. He has led the development of the Brownie Cashlib cryptographic library, which is available as open source online. He is a member of IACR, ACM, and IEEE. Dr. Küpçü has various accomplishments including 2 patents pending, and has been part of 6 funded research projects up to now, for 4 of which he was the principal investigator. For more information, visit http://crypto.ku.edu.tr