SESS: A Security-Enhanced Secret Storage Scheme for Password Managers

Hao Fang, Hu Aiqun, Le Shi and Tao Li
School of Information Science and Engineering
Southeast University
Nanjing, China

***Abstract*—Password-based authentication mechanism is used widely for its simplicity. However, due to the rapidly growth of computation power nowadays, low-entropy passwords and data protected by such passwords become more and more easy to attack. For against offline dictionary attacks with memorizable passwords, one may use interactive protocols, making offline guessing impossible. Then only online guessing, of which the times can be limited, remains for the attacker. In this paper we propose such a scheme, for securely storing high-entropy keys and other secret data shared in local storage and servers in the cloud, employing only low-entropy password. Even if one of the two parties in the protocol is cracked, the security of valuable data is still guaranteed. Our scheme is merely based on the assumption that one-way function exists, and is also easy to implement.**

Keywords*—*password, authentication, secret share

I. Introduction

There are two practical cryptographic authentication mechanisms: the one based on the Public Key Infrastructures (PKI), and the one based on passwords. The former is more professional and can achieve a very high security level, guaranteed by cryptographic hard problems to avoid at least brute-force cracking. However it also means a heavy architecture that depends on certification authorities and anyone who wants to show his identity must holds a long private key. Though the private key is never sent directly in any processes of authentication, the key holder must find a secure place to store his private key. However such a secure place is usually not so secure as it is expected.

The latter, the password-based authentication mechanism is a little old fashioned, but has a massive user base for its simplicity. It only requires the one who wants to show his identity to memorize a relatively short key, without the help of trusted storage. In fact it works in the past years by employing cryptographic hash functions to obfuscate the cleartext passwords, and by using salts to amplify the difficulty of making inverse hash table. But nowadays, though the computing power is still not able to solve cryptographic hard problems, it is sufficient to brute-force crack hashed (short) passwords. For example, a typically random-generated 7 bytes long password composed of all printable ASCII characters like 2Z(&u!0 (or equivalently a 20 bytes long normal word) can be cracked in 6 minutes by a normal Desktop PC. In fact, to obtain an enough strength of password as of an 128 bits symmetric key, one needs a 22 bytes long case sensitive alphanumeric password or a 39 bytes long numeral password, which is not easier to memorize than a real binary key directly.

To overcome this vulnerability, it is practically suggested to use a password manager to generate and store strong-enough password for each password-authenticated application, and to leverage the risk of brute-force attacks on varied applications to a master key for the authentication for the password manager itself and for encrypt the data that consist of the generated passwords. Though this method is widely used, there still exist vulnerabilities. In fact there are two main-stream modes of password manager: the one is to store users' passwords in local file, and the other one is to store them "in the cloud" by some providers; but it is common that local files are lost (for example, a password manager in a lost smart phone), as well as that databases in the cloud are leaked (as many news reported, for example [1]). Once an attacker obtains such valuable data, he will try to crack the data by guessing the master key. So how complicated should a master key is? If the master key is required to be complicated enough as we stated above, it is still hard to memorize and hence not very practical. Then if the master key is a simple password, the security of users' accounts is hardly guaranteed.

A. Our Approach

We then observe that though it is possible that a certain server is attaked and then leaks users' data, and it is also possible that a user loses his/her mobile device with the data, however, that it is truly rare that both events happen concurrently. So we can share the secret data (which contains the machine-generated long passwords) between the service provider (the server) and the user (the client, for example, a smart phone or a laptop). Basically it is the combination of the two ways of storing secret data as we mentioned above (locally and in the cloud). Intuitively, the leakage of the partial data either in the server or in the client is useless, except someone gets the other half. Further, the master-key-related information is also shared (comparing to traditional password-authenticated key exchange protocols, which usually require the server also holds the long-time (master) password/key), so that hacking any party, if not both, is helpless for the attacker to obtain the master key. In fact when the client needs to recover the secret data, it starts a session to the server, with the master-key-related information having been stored locally and the master key kept in the user's brain, and then the server will send its partial data back to the client if the authentication is passed, without learning what the real master key is. We achieve this by using one-way function and the one-time pad technique *very carefully*.

978-1-4673-7687-7/15/$31.00 ©2015 IEEE

B. Related Work

Our approach is in fact in a simplified scenario that comes from the password-protected secret share (PPSS) schemes [2], [3], [4], [5]. In a PPSS scheme, the user shares its secret data with several servers but does not keep any data himself except a password. PPSS schemes guarantee that if not all the servers are corrupted (or the number of corrupted servers is less than a certain threshold), then the secret data and the password is safe. A PPSS scheme with two servers will fit our scenario if the user himself runs one of the servers. However employing a PPSS scheme directly is costly, for they usually use lots of somewhat heavy zero-knowledge proofs.

We also want to point out that the PPSS model is quite different from other password-based authentication models (such as PAKE, password-authenticated key exchange) for it does not require the server knows pre-shared passwords. Then it gains the benefit that the server will even not be able to leak the users' passwords naturally if it is corrupted, for it never knows them.

C. Our Contributions

In this work, we construct a lightweight security-enhanced secret storage scheme that guarantees the privacy of secret data and the master key if either the server or the client is corrupted, which, comparing to current PPSS schemes that are suitable for this scenario, has a more concise settings and is based on more general assumptions.

II. Preliminary

We use the notation $x \xleftarrow{\$} X$ to denote x is chosen uniformly at random from the set X, and use λ to denote the security parameter throughout. We use the negligible notation $f(\lambda) \leq \mathsf{negl}(\lambda)$ to show a function $f(\lambda) = O(\lambda^{-c})$ for every constant $c > 0$.

We say an encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is *perfectly secure*[6] if for any two plaintext messages m_1, m_2 and any c,

$$\Pr[\mathsf{Enc}_k(m_1) = c] = \Pr[\mathsf{Enc}_k(m_2) = c]$$

The One-Time Pad(OTP) scheme is a symmetric encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as follows:

- $k \leftarrow \mathsf{Gen}$: $k \xleftarrow{\$} \{0,1\}^N$
- $c \leftarrow \mathsf{Enc}_k(m)$: $c = k \oplus m$
- $m \leftarrow \mathsf{Dec}_k(c)$: $m = k \oplus c$

where $k, c, m \in \{0,1\}^N$.

Theorem 1. *Let* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be the OTP scheme with key length* $N \geq 1$ *and assume we use it to encrypt a message* $m \in \{0,1\}^N$. *Then one-time pad scheme is perfectly secure.[6]*

We say a function $h : \{0,1\}^* \to \{0,1\}^\lambda$ is one-way, if for any probabilistic polynomial time algorithm $\mathcal{A}$,

$$\Pr_{x \xleftarrow{\$} \{0,1\}^*}[f(\mathcal{A}(f(x))) = f(x)] \leq \mathsf{negl}(\lambda)$$

where the probability comes from the choice of x and the randomness of $\mathcal{A}$. It is believed that one-way function exists, for example, the discrete logarithm in groups with prime order and cryptographic hash functions like SHA-256.

III. Definitions

A Security-Enhanced Secret Storage (SESS) scheme consists of two subprotocols $(\mathsf{Setup}, \mathsf{Retrieve})$, both involving 2 parties: a local client $\mathcal{U}$, and a server $\mathcal{S}$.

- Protocol $\mathsf{Setup}(uid, pw, K)$ is executed by $\mathcal{U}$, and $\mathcal{S}$, where $\mathcal{U}$, given its identity uid, a password pw, secret data K, outputs two pieces of password-and-private-data-related information w, w' respectively sent to $\mathcal{S}$ and stored locally.
- Protocol $\mathsf{Retrieve}(uid, pw')$ is executed by $\mathcal{U}$ and $\mathcal{S}$, where $\mathcal{U}$, given its identity uid and a password guess pw' as inputs, interacts with $\mathcal{S}$ and identify himself. $\mathcal{U}$ finally outputs the recovered private data K' from the data given by $\mathcal{S}$ and the local storage if the server authenticate the identity of $\mathcal{U}$. Otherwise it outputs an abort symbol $\perp$.

The *correctness* on a SESS scheme requires $\mathcal{U}$ finally outputs $K' = K$ when $pw' = pw$ and all parties follows the protocol.

The (client-side) *soundness* of a SESS scheme depicts the functionality that anyone who do not have the correct password will hardly obtain the secret data even if she corrupts the client. That it to say, the adversary's best strategy to obtain the secret data should be guessing password online.

Formally, we say a SESS scheme $\Pi = (\mathsf{Setup}, \mathsf{Retreive})$ is $(\mathcal{D}, q)$-*sound*, if we have

$$\mathbf{Adv}^{\mathsf{SS}}_{\mathcal{A},\Pi}(\lambda) = |\Pr[\mathbf{Exp}^{\mathsf{SS}}_{\mathcal{A},\Pi}(\lambda) = 1] - \frac{1}{2}| \leq \frac{q_r}{|\mathcal{D}|} + \mathsf{negl}(\lambda)$$

where $\mathcal{A}$ is any probabilistic polynomial time algorithm, $\mathcal{D}$ is the dictionary space from which passwords are chosen, q_r is the maximum number of adversary's allowed on-line guessing times and the security experiment $\mathbf{Exp}^{\mathsf{SS}}_{\mathcal{A},\Pi}(\lambda)$ is a probabilistic polynomial time algorithm as follows:

1: The adversary $\mathcal{A}$ chooses $K_0 \xleftarrow{\$} \{0,1\}^\lambda$ and $K_1 \xleftarrow{\$} \{0,1\}^\lambda$ and sends them to the challenger.
2: The challenger randomly chooses $b \xleftarrow{\$} \{0,1\}$ and $pw \xleftarrow{\$} \mathcal{D}$, runs $\mathsf{Setup}(0, pw, K_b)$ to get (w, w'), and then sends w' (the client side data) to the adversary.
3: The adversary $\mathcal{A}$ is allowed to run at most q_r times Retreive protocol, then outputs $b' \in \{0,1\}$.
4: The experiment returns 1 if and only if $b' = b$, otherwise it returns 0.

Note that this definition of the client-side security also implies any adversary could not obtain the correct password whether via offline attack or via limited online accesses, because if the adversary knows the correct password, she will easily impersonate the legal user and get the user's secret data.

The (server-side) *password-privacy* of a SESS scheme depicts the functionality that anyone who corrupts the server will never obtain the correct password (as well as the secret data), even if the client interacts with the server using the

correct password. It amounts to saying that the best strategy of an adversary to output the correct password is to guess one randomly.

Formally, we say a SESS scheme $\Pi = (\mathsf{Setup}, \mathsf{Retreive})$ is $\mathcal{D}$-*password-private*, if we have

$$\mathbf{Adv}^{\mathsf{SPP}}_{\mathcal{A},\Pi}(\lambda) = \Pr[\mathbf{Exp}^{\mathsf{SPP}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \frac{1}{|\mathcal{D}|} + \mathsf{negl}(\lambda)$$

where $\mathcal{A}$ is any probabilistic polynomial time algorithm, $\mathcal{D}$ is the dictionary space from which passwords are chosen, and the security experiment $\mathbf{Exp}^{\mathsf{SPP}}_{\mathcal{A},\Pi}(\lambda)$ is a probabilistic polynomial time algorithm as follows:

1: The challenger randomly choose $pw \xleftarrow{\$} \mathcal{D}$ and $K \xleftarrow{\$} \{0,1\}^\lambda$, run $\mathsf{Setup}(0, pw, K)$ to get (w, w') and then send w to the adversary as if the adversary is the server.
2: The adversary $\mathcal{A}$ is allowed to call $\mathcal{O}^{\mathsf{Retreive}}(\cdot)$ polynomial times then outputs $pw' \in \{0,1\}^t$, where $\mathcal{O}^{\mathsf{Retreive}}(\cdot)$ with input 0 (resp., 1) sends a retrieve message to the adversary using a incorrect password (resp., the correct password).
3: The experiment returns 1 if and only if $pw' = pw$, otherwise it returns 0.

As mentioned before, we do not allow $\mathcal{A}$ corrupts both parties, as it is a trivial case that a SESS scheme guarantees nothing. We limit the times of calling Retreive instances from $\mathcal{A}$ in $\mathbf{Exp}^{\mathsf{SS}}_{\mathcal{A},\Pi}(\lambda)$ because online password guesses are easy to detect and prevent.

IV. Our Protocol

The overview of Protocol Setup and Protocol Retrieve is given in Fig. 1. Throughout the protocols, we assume the dictionary space $\mathcal{D} \subset \{0,1\}^t$ where $t \leq \lambda$. We explain them as follows:

A. Protocol Setup.

Assume that the password of $\mathcal{U}$ is $pw \in \mathcal{D}$, the secret data of $\mathcal{U}$ is a string $K \in \{0,1\}^N$ (where we assume $N \geq \lambda$, without loss of generality) and uid is the client identifier chosen by $\mathcal{U}$. Let $\mathsf{pad}(\cdot) : \{0,1\}^t \to \{0,1\}^\lambda$ be the function padding the password to length λ with 0. Then the protocol Setup runs as follows:

1) Client $\mathcal{U} \leftarrow uid||pw||K$:
   1: Choose $p_1 \xleftarrow{\$} \{0,1\}^\lambda$ and $K_2 \xleftarrow{\$} \{0,1\}^N$;
   2: Calculate $p_2 = \mathsf{pad}(pw) \oplus p_1$ and $K_1 = K_2 \oplus K$;
   3: Send $w = uid||p_1||K_1$ to $\mathcal{S}$;
   4: Store $w' = uid||p_2||K_2$ in the local storage.
2) Server $\mathcal{S} \leftarrow uid||p_1||K_1$:
   1: If uid is not in its database, store $uid||p_1||K_1$ in the database, then send $uid||\mathsf{setup\_ok}$ to $\mathcal{U}$;
   2: Otherwise send $uid||\mathsf{setup\_failed}$ to $\mathcal{U}$.

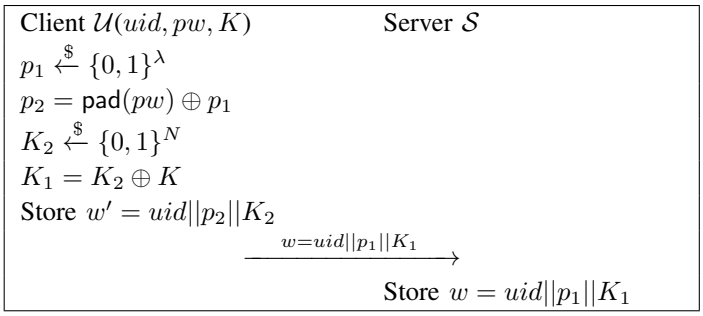
B. Protocol Retrieve.

Assume that the password guess of $\mathcal{U}$ is $pw' \in \mathcal{D}$, and uid is the client identifier chosen by $\mathcal{U}$. Let $h(\cdot) : \{0,1\}^\lambda \to \{0,1\}^\lambda$ be an one-way function. Then the protocol Retrieve runs as follows:

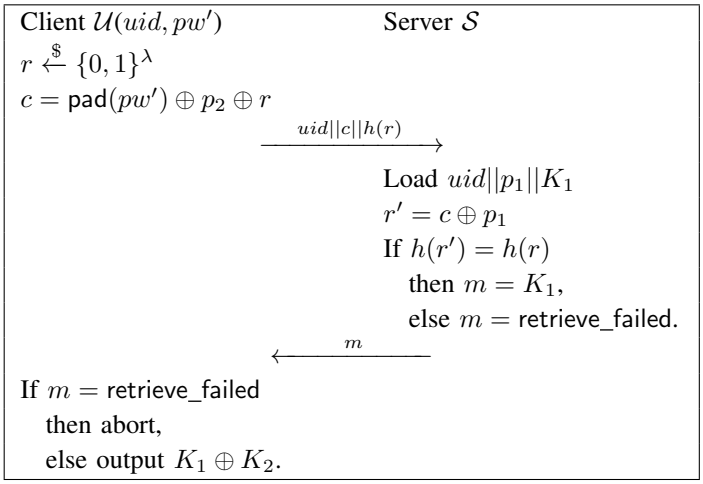
1) Client $\mathcal{U} \leftarrow uid||pw'$:
   1: if there is no such item $uid||*||*$ in the local storage then abort, otherwise load $uid||p_2||K_2$ into memory.
   2: Choose $r \xleftarrow{\$} \{0,1\}^\lambda$;
   3: Calculate $c = \mathsf{pad}(pw') \oplus p_2 \oplus r$;
   4: Send $uid||c||h(r)$ to $\mathcal{S}$.
2) Server $\mathcal{S} \leftarrow uid||c||h(r)$:
   1: If uid is not in its database, send $m = uid||\mathsf{retrieve\_failed}$ to $\mathcal{U}$ and abort.
   2: Otherwise load $uid||p_1||K_1$ into memory;
   3: Calculate $r' = c \oplus p_1$;
   4: If $h(r') = h(r)$ then send $m = uid||K_1$ to $\mathcal{U}$;
   5: Otherwise send $m = uid||\mathsf{retrieve\_failed}$ to $\mathcal{U}$.
3) Client $\mathcal{U} \leftarrow m$:
   1: If m is $uid||\mathsf{retrieve\_failed}$ then abort;
   2: Otherwise output $K' = K_1 \oplus K2$.

We want to point out that the implementation of the behaviors of the server will be a little more complicated than what is listed above, basically for limiting the number of query times and refreshing it. However as we already restrict the query ability of adversaries in security models, we simply ignore them here.

We also want to point out that the scheme itself do not guarantees the integrity of secret data, but it is easy to achieve by appending cryptographic hash value of the original secret data



(a) Protocol Setup



(b) Protocol Retrieve

Fig. 1. Overview of Setup and Retrieve Protocols

and verifying it accordingly. In fact we leave this functionality to password managers that call SESS. Besides, for avoiding man-in-the-middle attack, the client should authenticate the server at first. Practically this can be done via the standard `https` protocol, and we omit the details.

C. Analysis

Claim 1 (Correctness)**.** *SESS is correct for any security parameter λ, any password length t and any secret data length N.*

Proof: To proof the correctness, we need to show the retrieve protocol above can let the server test whether or not the password guess (pw') matches the original one (pw). Firstly we observe the following equations:

$$r' = c \oplus p_1 = \underbrace{p_1 \oplus p_2}_{\mathsf{pad}(pw)} \oplus \mathsf{pad}(pw') \oplus r = \mathsf{pad}(pw) \oplus \mathsf{pad}(pw') \oplus r$$

Hence $r' = r$ if and only if $pw' = pw$. Secondly, the server will see the condition $h(r') = h(r)$ is true if $r' = r$. Combining these two observations, we clearly see that the correctness holds. ∎

Claim 2 (Soundness)**.** *If there exists any one-way function $f : \{0,1\}^* \to \{0,1\}^\lambda$ with the possibility $\mathsf{negl}(\lambda)$ to compute the inverse, then for all probabilistic polynomial time adversary $\mathcal{A}$, any dictionary space $\mathcal{D} \subset \{0,1\}^t$, any allowed number of query times q_r and any secret data length $N \geq \lambda$, the advantage $\mathbf{Adv}^{\mathsf{SS}}_{\mathcal{A},\Pi}(\lambda)$ satisfies:*

$$\mathbf{Adv}^{\mathsf{SS}}_{\mathcal{A},\Pi}(\lambda) \leq \frac{q_r}{|\mathcal{D}|} + \mathsf{negl}(\lambda)$$

Informally we see that directly cracking p_2 and K_2 makes no sense by viewing them as OTP ciphertext. Then the adversary has to interact with the server. Except guessing password online, the adversary can only impersonate the client by sending faked c^* to the server, and expecting the server computes $r^* = c^* \oplus p_1$ where $r^* = r$. However, by considering p_1 as a OTP key and r as the corresponding ciphertext, it means the adversary must do an OTP decrypt $c^* = \mathsf{Dec}_{p_1}(r) = r \oplus p_1$ without the key p_1, which is obviously successful only in probability $1/2^\lambda$.

Proof: We prove the claim by enumerating all the possible strategies:

1) If the adversary decides to crack K_2 directly, due to that it is a ciphertext protected by the OTP key K_1 which the adversary has not known, the advantage is $1/2^N$.
2) If the adversary decides to crack the password from p_2, due to that it is a ciphertext protected by the OTP key p_1 that the adversary has not known, the advantage is $1/2^\lambda$. Note that if the guess is wrong, the adversary will get no further information about the correct password.
3) If the adversary decides to guess the password randomly, the advantage is $q_r/|\mathcal{D}|$. Again, note that if the guess is wrong, the adversary will get no further information about the correct password.
4) If the adversary decides to cheat the server without the knowledge about the correct password, then the adversary generates c^* by some algorithm $\mathcal{B}$ with input r and sends $uid||c^*||h(r)$ to the server:
   a) If the adversary can successful let $r' = r$ in probability ϵ, then $\mathcal{B}$ will be able to decrypt any OTP ciphertext with length λ in ϵ, which deduces that $\epsilon = 1/2^\lambda$.
   b) If the adversary can successful let $r' = r^*$ where $h(r^*) = h(r)$ and $r^* \neq r$ in probability ϵ, then $\mathcal{B}$ will be at least able to inverse the one-way function in ϵ, which deduces that $\epsilon \leq \mathsf{negl}(\lambda)$.

By summing up the possibility of success in each case, we conclude the result. ∎

Claim 3 (Password-Privacy)**.** *If there exists any one-way function $f : \{0,1\}^* \to \{0,1\}^\lambda$ with the possibility $\mathsf{negl}(\lambda)$ to compute the inverse, then for all probabilistic polynomial time adversary $\mathcal{A}$, any dictionary space $\mathcal{D} \subset \{0,1\}^t$, any secret data length $N \geq \lambda$, the advantage $\mathbf{Adv}^{\mathsf{SPP}}_{\mathcal{A},\Pi}(\lambda)$ satisfies:*

$$\mathbf{Adv}^{\mathsf{SPP}}_{\mathcal{A},\Pi}(\lambda) \leq \frac{1}{|\mathcal{D}|} + \mathsf{negl}(\lambda)$$

To show the password-privacy, we also start from the non-interactive case. Obviously, the adversary cannot get any information about pw because p_1 is generated randomly. Then the adversary needs to let the client do some password guesses. In real world, this can be done by just waiting. When the client does such a guess pw', the adversary will see $r' = \mathsf{pad}(pw) \oplus \mathsf{pad}(pw') \oplus r$ as well as $h(r)$. Since $\Delta pw = pw \oplus pw' \in \{0,1\}^t$, the adversary can search the space to test which Δpw^* let $h(r' \oplus \mathsf{pad}(\Delta pw^*)) = h(r)$. And then the question becomes extract pw from $pw \oplus pw'$ with the unknown pw', which success only in $1/2^t$.Hence the most efficient way for the adversary is to guess a password directly.

Proof: We prove the claim by enumerating all the possible strategies:

1) Note that p_1 is only random value, the adversary can decide to guess the password directly with the possibility of success $1/|\mathcal{D}|$.
2) If the adversary decides to capture password from a client retrieve request with correct password the adversary knows in fact $p_2 \oplus \mathsf{pad}(pw)$. However $p_2 \oplus \mathsf{pad}(pw)$ is exactly p_1, hence no further information is got by a right guess, and the adversary is exactly the one in case 1).
3) When the password guess from the client is wrong, then the adversary knows $\mathsf{pad}(pw) \oplus \mathsf{pad}(pw') \oplus r$. Instead of extracting r from $h(r)$, the adversary can search the space $\Delta pw = \mathsf{pad}(pw) \oplus \mathsf{pad}(pw') \in \{0,1\}^t$ to find out a suitable Δpw^* satisfying $h(r' \oplus \mathsf{pad}(\Delta pw^*)) = h(r)$. However it is clear that to extract pw from (Δpw^*) successfully only happens in probability $1/2^t \leq 1/|\mathcal{D}|$. Hence in this case the advantage is still less than $1/|\mathcal{D}|$.

By summing up the possibility of success in each case, we conclude the result. ∎

To extend our protocol more practically, we may consider allowing the user to backup $p_2||K_2$ (which is normally stored in user's local storage after Protocol Setup) to an out-of-band-authenticated (typically by SMS) cloud service in order to recover these data after reinstalling the operation system or transferring to some other new device. Due to the out-of-band assumption, the leaks of these data in this sort of cloud service can be considered equally as the leaks from the device hold by users, which, as long as $p_1||K_1$ is secure, does not give the adversary any more hint.

V. Experiment

We instantiate our scheme under a Macbook Air computer running Ubuntu 14.04, by using the Linux special file `/dev/random` [7] as the random number generator and SHA-2 hash function family as the candidate one-way functions.In detail, we choose $\lambda \in \{224, 256, 384, 512\}$, where the numbers are allowed output lengths of the SHA-2 family. The dictionary space are 8 bytes ASCII printable characters, of which the length is considered memorizable. Since K_2 stored in the server is exactly the one-time pad chipertext that is perfectly secure, we focus on password-related information.

As an example, we chose $\lambda = 224$, and a simple password `sess2015`, then after the protocol Setup, the master-password-related information stored in the client-side (in our experiment) was:

```
33 70 3B AF 2A F8 CE 69 8A 36 50 5F 09 65
29 3B B4 0B CE 21 0A 13 12 59 41 22 66 4A
```

and that stored in the server-side was:

```
40 15 48 DC 18 C8 FF 5C 8A 36 50 5F 09 65
29 3B B4 0B CE 21 0A 13 12 59 41 22 66 4A
```

Although the last 20 bytes of each side are the same, we stress that the adversary cannot see the two sides concurrently as we have assumed. Then no matter the data of which side are seen by the advervesary, it is, in the adversasry's view, just a randomized 28-byte-long string. Further more, the adversary will find she can "decode" the string to any password she want, so offline attacking is not just computationally impossible. It is in fact absolutely impossible. This is very different from the normal password file format as `hash(password||salt)||salt`, which the adversary can do offline attack by pre-calculated rainbow table, because the hash value will hint the adversary whether her guess is right.

The password-guess-related information $c||h(r)$ generated during the execution of protocol Retrieve were:

```
63 80 1C C0 A1 8C E4 87 2A D1 12 D5 C0 C2
F0 A7 CA 37 D6 0B 27 01 E3 AD E2 74 A2 58
B6 8B 31 23 B5 39 8E 92 40 88 10 12 53 87
B1 DF DB 22 E4 74 F0 12 9F 97 FB 45 C7 04
```

(a right guess, where $pw' =$ `sess2015`) and

```
1A 6B EA 4B 69 03 5D DE 35 D4 32 D9 0D BF
9B 0E E2 97 7B 74 99 C6 46 92 10 51 A5 B4
95 88 A0 EC 5F 2A 3B 41 36 43 7A 10 26 94
57 B6 5E A8 87 2B AB FF 9C 98 FD E5 B1 B4
```

(a wrong guess, where $pw' =$ `12345678`).

For the right guess, as we have shown in the proof of password-privacy, the adversary who even had cracked the server would obtain nothing except the fact the client did a right guess. For the wrong guess, note that it is possible to recovery $\Delta pw = pw \oplus pw'$ via an offline attack if *the adversary has cracked the server*, also as we have shown in the proof of password-privacy. However, there is no other way for the adversary to get what pw' was in that message, because if the adversary controls the server, she will not control the client in our model. Note that there are still 2^t pairs (pw, pw') satisfying $\Delta pw = pw \oplus pw'$, the adversary would not obtain any advantage from such an offline attack.

VI. Conclusion

We proposed a lightweight SESS protocol which holds soundness and password-privacy against malicious adversaries, which can especially resist offline dictionary attacks after either the server or the client is corrupted, even if low-entropy passwords are used. We believe this scheme is practical for enhancing the security of password managers nowadays. Further work will be to extend this scheme to a more rigorous model such as Universally Composable Security Framework [8].

Acknowledgment

This work is supported by National Key Basic Research Program of China (No. 2013CB338003).

References

[1] N. Perlroth and D. Gelles, "Russian hackers amass over a billion internet passwords," *New York Times*, no. August 5, 2014.

[2] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu, "Password-protected secret sharing," in *Proceedings of the 18th ACM conference on Computer and Communications Security*. ACM, 2011, pp. 433–444.

[3] J. Camenisch, A. Lysyanskaya, and G. Neven, "Practical yet universally composable two-server password-authenticated secret sharing," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 525–536.

[4] J. Camenisch, A. Lehmann, A. Lysyanskaya, and G. Neven, "Memento: How to reconstruct your secrets from a single password in a hostile environment," in *Advances in Cryptology–CRYPTO 2014*. Springer, 2014, pp. 256–275.

[5] S. Jarecki, A. Kiayias, and H. Krawczyk, "Round-optimal password-protected secret sharing and t-pake in the password-only model," in *Advances in Cryptology–ASIACRYPT 2014*. Springer, 2014, pp. 233–253.

[6] M. Bellare and P. Rogaway, "Introduction to modern cryptography," *UCSD CSE*, vol. 207, p. 33, 2005.

[7] Z. Gutterman, B. Pinkas, and T. Reinman, "Analysis of the linux random number generator," in *Security and Privacy, 2006 IEEE Symposium on*. IEEE, 2006, pp. 15–pp.

[8] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*. IEEE, 2001, pp. 136–145.