# Response Letter

**Journal Name:** IEEE Transactions on Mobile Computing

**Reference Number:** TMC-2019-01-0039

**Title of Paper:** PROTECT: Efficient Password-based Threshold Single-sign-on Authentication for Mobile Users against Perpetual Leakage

**Authors:** Yuan Zhang, Chunxiang Xu, Hongwei Li, Kan Yang, Nan Cheng, and Xuemin (Sherman) Shen

We would like to thank the editor-in-chief, the associate editor, and anonymous reviewers for all the valuable comments. We have carefully revised our manuscript according to the comments. The revision and response can be found in the following sections.

---

## I. Response to Reviewer #1

**Comment 1.1:** *In the Sign-on phase, it is not clear when $\mathcal{U}$ is actually authenticated. The part of the signature on $H(psw_{\mathcal{U}})$ at registration ($sp_i$) is stored by $\mathcal{IS}_i$. In the authentication phase, an entirely new $sp_i$ is sent and (1) at no point in the protocol specification is there any attempt at matching this value to anything; (2) This value is derived from a new signature and would bear no relationship to the old $sp_i$.*

**Response 1.1:** PROTECT achieves secure password-based authentication, which is proved in the following.

First, in PROTECT, the authentication credential, which is stored on $\mathcal{IS}_i$ and used to authenticate the user $\mathcal{U}$, is $sp_i = \hbar(sp_{\mathcal{U}}||i)$ ($\hbar$ is a secure hash function). $sp_{\mathcal{U}}$ is computed on the $\mathcal{U}$'s password $psw_{\mathcal{U}}$ and the server-side key $k$. It is first generated in the **Register** algorithm. In **RetriSerPsw**, $\mathcal{U}$ interacts with identity servers to re-compute $sp_{\mathcal{U}}$. The signature algorithm used in PROTECT is the BLS signature [1], which is a deterministic signature algorithm, i.e., for the same message and the same secret key (i.e., signing key), it always yields the same signature, and generating the signature via the $(t, n)$-threshold way would not change the determinism of the signature [2]. Therefore, if $\mathcal{U}$ takes the correct password $psw_{\mathcal{U}}$ in **RetriSerPsw** as the input,

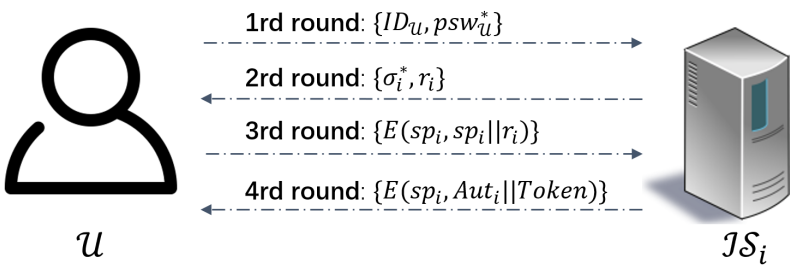


Fig. 1: Communications between $\mathcal{U}$ and $\mathcal{IS}_i$ in the *Sign-on* phase

she/he would obtain $sp_{\mathcal{U}}$ that is the same as the one computed in **Register**. With $sp_{\mathcal{U}}$, $\mathcal{U}$ is able to compute $sp_i$ and to pass $\mathcal{IS}_i$'s authentication.

Here, $sp_{\mathcal{U}}$ should be re-computed in **RetriSerPsw**, due to the following two reasons:

- PROTECT is a password-based single-sign-on authentication scheme, the only secret known to $\mathcal{U}$ for authentication is her/his human-memorable password $psw_{\mathcal{U}}$.
- Given $sp_{\mathcal{U}}$, an adversary cannot extract $psw_{\mathcal{U}}$ by performing dictionary guessing attacks.

Second, in the *Sign-on* phase (which consists of two algorithms: **RetriSerPsw** and **Authentication**), the communications between $\mathcal{U}$ and $\mathcal{IS}_i$ in PROTECT are shown in Fig. 1.

After the 3rd round of communication, $\mathcal{IS}_i$ receives $E(sp_i, sp_i||r_i)$ from $\mathcal{U}$. $E(sp_i, sp_i||r_i)$ is the ciphertext of $sp_i||r_i$ protected under $sp_i$ (this $sp_i$ is re-computed by $\mathcal{U}$ in **RetriSerPsw**) using a secure symmetric-key encryption algorithm (e.g., AES). Due to the security of the encryption algorithm, if and only if $\mathcal{U}$ utilizes the correct authentication credential $sp_i$ to encrypt $sp_i||r_i$, $\mathcal{IS}_i$ is able to decrypt the ciphertext successfully (by using $sp_i$ stored on $\mathcal{IS}_i$ and generated in **Register**) and retrieve the correct $sp_i||r_i$. Therefore, $\mathcal{IS}_i$ does not need to check whether the decrypted $sp_i$ (which is re-computed by $\mathcal{U}$ in the *Sign-on* phase) matches the one that is generated and stored locally in **Register**, it only needs to check whether the decryption succeeds and whether $r_i$ is the one sent in the 2rd round of communication.

Therefore, PROTECT achieves secure password-based authentication: if and only if $\mathcal{U}$ takes the correct password as the input, she/he can pass the identity servers' authentication and can obtain a valid authentication token.

In the revised manuscript, we have added a correctness proof of PROTECT in Section 6.1.

**Comment 1.2:** *In the Sign-on phase (pg 7, col 2, , line 49+), $ID_{\mathcal{U}}$ and $psw_{\mathcal{U}}^*$ are sent in the clear, and $\rho_{\mathcal{U}}$ is incremented without further checking. Any eavesdropper may replay this message to increase the value of $\rho_{\mathcal{U}}$ and the real $\mathcal{U}$ will be locked out soon enough. Indeed, since $ID_{\mathcal{U}}$ is*

*sent in the open (Fig 1), anyone can replay $ID_{\mathcal{U}}$ with any arbitrary $r'H(psw')$ and masquerade as $\mathcal{U}$.*

**Response 1.2:** In the revised manuscript, we propose a detection mechanism and integrate it into PROTECT to resist the eavesdropper who intercepts and replays users' messages.

Particularly, in addition to $\rho_{\mathcal{U}}$ (which records the number of authentication token requests made by $\mathcal{U}$), each identity server also records the number that $\mathcal{U}$ fails to pass identity servers' authentication. This number is denoted by $\phi_{\mathcal{U}}$, and the upper limit during an epoch is $\phi$ which is determined by the security parameter. In reality, $\phi$ can be strictly restricted and should be much smaller than $\rho$ (which is the upper limit of the number of authentication token requests made by $\mathcal{U}$ during an epoch). If $\phi_{\mathcal{U}}$ reaches the upper limit, identity servers can consider that an adversary is attempting to compromise the $\mathcal{U}$'s account (either guessing $\mathcal{U}$'s password or baring $\mathcal{U}$ from requesting authentication tokens). As such, identity servers would lock the $\mathcal{U}$'s account.

We also considered an alternative mechanism for alleviating the above attack while ensuring the availability of single-sign-on authentication for users. Instead of directly locking $\mathcal{U}$'s account if $\phi_{\mathcal{U}} > \phi$, identity servers utilize an exponential delay mechanism as follows: each time identity servers receive a query from $\mathcal{U}$, an artificial delay $t_D$ is introduced before key servers respond to $\mathcal{U}$'s query. Initially, $t_D$ could be small and doubles this quantity after each query. This delay would be removed in next epoch.

In password-based authentication schemes, the only secret known to the user $\mathcal{U}$ is her/his human-memorable password $psw_{\mathcal{U}}$. In PROTECT, each identity server can be compromised by an adversary and the user's identity $ID_{\mathcal{U}}$ can be leaked to the adversary after the Registration phase. Furthermore, in the Sign-on phase, to compute the servers-derived password $sp_{\mathcal{U}}$, $\mathcal{U}$ first sends $ID_{\mathcal{U}}$ and $psw^*_{\mathcal{U}}$ to $\mathcal{IS}_i$. Here, $psw^*_{\mathcal{U}} = rH(psw_{\mathcal{U}})$ and the adversary cannot extract any information about $psw_{\mathcal{U}}$ from $psw^*_{\mathcal{U}}$ due to the obliviousness of PROTECT (the formal proof of the obliviousness is provided in Section 7.2 of our manuscript). Therefore, the adversary would not benefit from eavesdropping on the communications between $\mathcal{U}$ and $\mathcal{IS}_i$.

In the revised manuscript, we have integrated the above detection mechanism into PROTECT to mitigate attacks from the eavesdropper in Section 6.1, and have discussed the alternative mechanism in Section 6.2.

**Comment 1.3:** *I believe the authors should consider carefully each entity and the communication*

| Entity \ Maintained values | Setup | | MasterKeyGen | | Register | | RetriSerPsw | | Authentication | | RenewShare | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Secret | Public | Secret | Public | Secret | Public | Secret | Public | Secret | Public | Secret | Public |
| $\mathcal{U}$ | $\perp$ | $PP$ | $\perp$ | $PMSK$ $\{PMSK_i\}_{i=1,\ldots,n}$ | $psw_{\mathcal{U}}$ | $ID_{\mathcal{U}}$ | $\sigma_{\mathcal{U}}$ | $\perp$ | $\perp$ | $AutToken_{\mathcal{U}}$ | $\perp$ | $\{PMSK_i'\}_{i=1,\ldots,n}$ |
| $\mathcal{IS}_i$ | $\perp$ | $PP$ | $msk_i$ | $PMSK$ $\{PMSK_i\}_{i=1,\ldots,n}$ | $sp_i, k_i$ | $ID_{\mathcal{U}}$ $\rho_{\mathcal{U}}$ $\phi_{\mathcal{U}}$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $msk_i'$ | $\{PMSK_i'\}_{i=1,\ldots,n}$ |

Fig. 2: Parameters stored on each entity after the execution of each algorithm

*channels between them, and what values are stored at each entity. It seems also that it is important that certain values are deleted after certain communications. The protocol should specify all these.*

**Response 1.3:** In PROTECT, the communications in the registration phase and key renewal phase are well protected. In these two phases, a user/identity server would establish a TLS connection with identity servers. However, the sign-on phase is completely under the control of adversaries, where the interaction messages between a mobile user and identity servers can be arbitrarily intercepted and modified by adversaries.

In PROTECT, the parameters that are stored on each entity after the execution of each algorithm are specified as shown in Fig. 2 and detailed as follows. In Fig. 2, "Secret" means that the parameters should be secretly stored; "Public" means that the parameters can be publicly verifiable.

In the Register phase, to register with identity servers, a user $\mathcal{U}$ needs to randomly choose $r \in Z_p^*$, compute $psw^*$, $\sigma_{\mathcal{U}}$, $sp_{\mathcal{U}}$, $sp_i$ and the identity server $\mathcal{IS}_i (i = 1, 2, \cdots, n)$ needs to compute $s_i$ and $\sigma_i^*$. After execution of **Register**, $\mathcal{U}$ deletes $\sigma_{\mathcal{U}}$, $r$, $sp_{\mathcal{U}}$, and $sp_i$, she/he only needs to memorize her/his password for subsequent authentications; $\mathcal{IS}_i$ securely maintains $ID_{\mathcal{U}}$, $sp_i$, $k_i$, $msk_i$ (which is generated in **MasterKeyGen**), $\rho_{\mathcal{U}}$, $\phi_{\mathcal{U}}$ for authenticating $\mathcal{U}$ and deletes $\sigma_i^*$.

In **RetriSerPsw**, $\mathcal{U}$ interacts with $\mathcal{IS}_i$ to compute $\sigma_{\mathcal{U}}$ (which is the same as the one generated in **Register**). With $\sigma_{\mathcal{U}}$, $\mathcal{U}$ is able to compute the server-derived password $sp_{\mathcal{U}}$ and further calculate the authentication credentials $sp_i$ for $i = 1, 2, \cdots, n$. If and only if $\mathcal{U}$ takes the correct $psw_{\mathcal{U}}$ as input, he can retrieve the correct $\sigma_{\mathcal{U}}$, and can further compute the correct $sp_i$ to pass the $\mathcal{IS}_i$'s authentication. If the authentication succeeds, $\mathcal{IS}_i$ generates an authentication token share $Aut_i||Token$ for $\mathcal{U}$. With a sufficient number authentication shares, $\mathcal{U}$ can compute a valid authentication token $(Token, Aut_{\mathcal{U}})$. All the above processes are involved in the Sign-on phase. After the Sign-on phase, $\mathcal{U}$ needs to maintain $(Token, Aut_{\mathcal{U}})$ for access to services from service

providers and memorize $psw_{\mathcal{U}}$ for subsequent authentications, and deletes other values and parameters generated in the Sign-on phase; $\mathcal{IS}_i$ does not need to maintain other new parameters (in reality, $\mathcal{IS}_i$ may be required to store $Aut_i||Token$ for post investigations, we would not discuss this for the sake of brevity).

After the key renewal phase, each identity server $\mathcal{IS}_i(i = 1, 2, \cdots, n)$ stores renewed master secret share $msk_i^{'}$ locally and securely deletes $msk_i$ and other values generated in **RenewShare**.

In the revised manuscript, we have further improved the specification of PROTECT, where the parameters that should be securely deleted are specified.

**Comment 1.4:** *The authors should also be clearer about "authentication", here there are two aspects of "authentication": one of user authentication to the identity servers, and one of the user authentication to service providers using tokens. It is not always clear in the text which aspect the authors are referring to.*

**Response 1.4:** To avoid the potential misunderstanding proposed by the reviewer, in the revised manuscript, "authentication" refers to the operation that identity servers authenticate a user. The operation that the service providers check the users' tokens is called "verification".

**Comment 1.5:** *It is also not clear what DGA is - an adversary performing the attack on passwords leaked from servers or adversaries using servers as oracles to guess passwords?*

**Response 1.5:** DGA (dictionary guessing attacks) refers to such an attack: an adversary collects all possible passwords to form a password dictionary and try the passwords in the dictionary one by one until one succeeds. The success of DGA is based on the observation that short strings are easier for humans to remember, and meaningful words are far more memorable than random character sequences [3].

Recently works [4], [5] have also shown the fact that users always choose popular passwords and utilize sister passwords (tweaked or reused passwords) in different systems. With the fact, the adversary is easier to collect the password dictionary if he can obtain a password database leaked from other servers.

The attack that adversary utilizes the identity server as an oracle to guess passwords is formalized as online passwords testing attacks (OPTA). OPTA can be considered as a variant of DGA: In a DGA, the adversary fixes an account (ID) and tries different passwords from the dictionary until one succeeds; In an OPTA, the adversary fixes a password from the dictionary, and try

different accounts until one succeeds (if the adversary fails to pass the authentication by using all accounts, he can fix another password from the dictionary and repeat this process until one succeeds).

In the revised manuscript, we have further elaborated on DGA and OPTA and provided a comparison between DGA and OPTA in Section 4.2.

**Comment 1.6:** *Section 2: pg 2 line 58-59 2nd column: What "other authentication paradigms"?*

**Response 1.6:** Other authentication paradigms refer to the authentication schemes that users authenticate themselves to the authenticator using other authentication factors, such as physiological features, secret keys shared between users and the authenticator, distances between users and the authenticator, and certificates issued by a Certificate Authority, rather than passwords.

Compared with these authentication paradigms, password-based authentication provides mobile users a more convenient, user-friendly, and efficient way to identify themselves.

In the revised manuscript, we have further elaborated on "other authentication paradigms" in Section 2.

**Comment 1.7:** *pg 3 line 2-3 1st column: "an identity server who is subject to the service provider(s)" - what does "subject to" mean here?*

**Response 1.7:** In single-sign-on authentication schemes, an identity server is introduced to authenticate users on behalf of the one or more service providers. Here, the identity server is jointly employed by the service providers.

In the revised manuscript, we have modified this sentence in Section 2 to describe the mechanism precisely.

**Comment 1.8:** *line 6: "With the authentication token, the user can request multiple services from other mobile service providers." A token can be used multiple times?*

**Response 1.8:** The construction and usage of authentication tokens depend on the system.

In PROTECT, the authentication token consists of two parts. The first part is the content of token, which contains the user's information/attributes, the expiration time of the token, a policy that would control the nature of access, and other auxiliary information that precisely defines the usage of the token. The second part is the signature of the token generated by identity servers.

For example, in Kerberos [6], the authentication token (i.e., the ticket) can be used multiple

times before the expiration time. The usage of authentication tokens in PROTECT can be defined as needs.

**Comment 1.9:** *line 38-52: Two categories of techniques preventing off-line DGA, one in 1990 and the other one from 2015 onwards. Is there nothing in between?*

**Response 1.9:** Password-based authentication with salted password hashing is a general solution to protect passwords from external adversaries. All the schemes utilizing salted password hashing essentially share the same construction, i.e., the authentication credential is the hash value of a password and a long-term random number (called salt). Therefore, we only cite the priori work on this type of password-based authentication schemes.

There is another type of password-based authentication schemes that can resist off-line DGA: An independent server is introduced to harden the users' passwords [8]–[10]. Specifically, the authentication credential is computed on the password and a secret that is possessed by the server. As long as the secret is inaccessible to adversaries, the security of the scheme against DGA is ensured. However, the security of these schemes is based on the reliability of the independent server. If the independent server is compromised, the schemes are still vulnerable to off-line DGA.

In the revised manuscript, we have improved the related work section.

**Comment 1.10:** *Section 3.2. Threshold cryptography - strictly speaking [42] is generally referred to as threshold secret sharing. Many threshold cryptographic primitives are based on [42] but not all.*

**Response 1.10:** In the revised manuscript, we have revised the description of threshold cryptography to make it precise.

**Comment 1.11:** *Section 5: the figures and the text are not consistent with each other. The inputs and outputs of the algorithms are not presented in the same way. It is very confusing. And when you write Register(...) it is not clear that this protocol is executed by different entities, and it is not clear what values are given to which entities, and it is not clear at the end of the protocol what values persist and what values must be deleted.*

**Response 1.11:** In the revised manuscript, we have improved the presentation of Section 5 to overview PROTECT clearly. Specifically, the algorithms (i.e., **MasterKeyGen**, **Register**,

**RetriSerPsw**, and **Authentication**) in PROTECT are interactive, and we prefix calls with $\mathcal{U}$ when they are made by the user $\mathcal{U}$ and with $\mathcal{IS}_i$ when they are made by the identity server $\mathcal{IS}_i$ for $i = 1, 2, \cdots, n$. We have added the inputs of each entity in different algorithms in Fig. 2 and Fig. 3 of the revised manuscript. We also have specified the parameters that should be well maintained and those that must be deleted in Section 6.

**Comment 1.12:** *Algorithm 1 seems redundant, while Algorithm 2 seems lacking a reference.*

**Response 1.12:** We have deleted Algorithm 1 and added the corresponding references.

## II. Response to Reviewer #2

**Comment 2.1:** *Overall, I liked the paper as a follow up to PASTA. I have only some comments that should be easy to addressed.*

**Response 2.1:** Thanks for your positive comments.

**Comment 2.2:** *Protection against an on-line attack. One of the main contributions of the paper is protecting identity servers from attackers that try to guess a password using on-line trials. The motivation behind this is that databases with similar (sister) passwords could be leaked, so the attacker can infer the user's password, not by means of offline cracking a hash, by means of just guessing it, perhaps by performing some tweaks over a leaked password. I think it's hard to get convinced with this argument, since the attempts required should be again *enough* to be detected by simple systems. As an example, most services will inject a CAPTCHA after only a few attempts, so on-line guessing should be considered a weak threat model. Of course, I appreciate the cryptographic solution presented in the paper, but, to be fair, there should be a discussion of alternative solutions for capturing an on-line guessing attempt.*

**Response 2.2:** In PROTECT, we consider the online passwords testing attack (OPTA), where the adversary, who has a set of password dictionary, can fix a password from the dictionary and try different identities to execute **RetriSerPsw** until one succeeds (if the adversary fails to pass the authentication by using all accounts, he can fix another password from the dictionary and repeat this process until one succeeds).

Resistance of existing single-sign-on authentication schemes (including PROTECT) against OPTA is based on a fact that the authenticator is able to obtain the authentication result. With the authentication results, if an adversary launches OPTA, the authenticator can detect

it. Traditionally, CAPTCHA is an effective way to resist such the attack. However, with the development of machine learning, the efficiency that the adversary breaks CAPTCHA has been improved significantly [7].

One of the primary motivations of our work is to resist OPTA. To this end, in PROTECT, if the adversary fails to sign on the system multiple times, the account utilized by the adversary is locked by the authenticator. Such the mechanism ensures that the number that the adversary tries different passwords to sign on the system can be strictly restricted. On the other hand, this mechanism could also be an overkill for negligent users, if a negligent user enters incorrect passwords multiple times, her/his account would be locked. As a consequence, the regular service would be impeded.

In the revised manuscript, we have proposed an alternative mechanism to mitigate OPTA while ensuring regular services for negligent users. Specifically, in addition to $\rho_{\mathcal{U}}$ (which records the number of authentication token requests made by $\mathcal{U}$), each identity server also records the number that $\mathcal{U}$ fails to pass identity servers' authentication. This number is denoted by $\phi_{\mathcal{U}}$, and the upper limit during an epoch is $\phi$ which is determined by the security parameter. In reality, $\phi$ can be strictly restricted and should be much smaller than $\rho$ (which is the upper limit of the number of authentication token requests made by $\mathcal{U}$ during an epoch). If $\phi_{\mathcal{U}}$ reaches the upper limit, instead of directly locking $\mathcal{U}$'s account, identity servers utilize an exponential delay mechanism as follows: each time identity servers receive a query from $\mathcal{U}$ (after $\phi_{\mathcal{U}} > \phi$), an artificial delay $t_D$ is introduced before key servers respond to $\mathcal{U}$'s query. Initially, $t_D$ could be small and doubles this quantity after each query. This delay would be removed in next epoch.

**Comment 2.3:** *How often the master key should be refreshed? One of the contributions of the paper is the resistance against strong attackers that they can compromise a large set (larger than the threshold, t) of identity servers. For defending such attackers, PROTECT periodically refreshes the cryptographic keys shared by the identity servers. However, here there is a race between the attacker compromising identity servers and PROTECT refreshing the keys. This should be further explored. In fact, in the paper the authors, when referencing Fig. 10, they claim that the operation of key renewal will not be triggered frequently. Since, this is one of the main contributions of the paper, it should be further discussed, since there is an obvious race the attacker is trying to win here.*

**Response 2.3:** The frequency to renew the secrets on identity servers corresponds to the length of epochs in PROTECT, since the renewal of secrets (i.e., the **RenewShare** algorithm) only needs to be executed once at the end of each epoch. The length of each epoch depends on the system. In PROTECT, renewing the secrets on identity servers would not require users' participation. **RenewShare** is an interactive algorithm (only) between identity servers. The length of epochs can be fixed or dynamically adjusted, and it mainly depends on the requirements of actual schemes protected by PROTECT.

In the revised manuscript, we have added a discussion on the setting of the length of epoch in Section 8.3 as follows.

"In reality, the length of each epoch (i.e., the frequency of performing key renewal) depends on the system requirements, it can be pre-fixed according to the security parameter or dynamically adjusted with the state of systems protected by PROTECT. Since identity servers could be well equipped and keep online in PROTECT, the communication costs on the identity server side can be accepted and would not become a bottleneck for applications."

**Comment 2.4:** *There are many recent papers [8]–[11] in password hardening. These papers solve a similar problem (how to protect the password/credential once the "service", or "identity server" in our case, is compromised. This is done with a different approach, and they also use advances in modern cryptography. I believe PROTECT should be discussed and compared (qualitatively) with them.*

**Response 2.4:** We have further investigated the papers [8]–[11] mentioned by the reviewer and discussed the differences and relationship between PROTECT and these schemes in the following.

In [8]–[10], the authors target at protecting passwords from off-line dictionary guessing attacks (DGA) in any authentication schemes that leverage passwords as an authentication factor. These schemes share essentially the same construction: an independent PRF server (i.e., the PRF service provider), who holds a system-level secret, is introduced to process the passwords before they are stored or verified. Barring compromise of the PRF server, it ensures that stolen password hashes cannot be cracked using the off-line DGA. The interaction between the PRF server and a user is oblivious, such that the PRF server cannot learn anything about the passwords.

Although utilizing the schemes in [8]–[10] can construct a single-sign-on authentication scheme

(the PRF server can be considered to be the identity server in the single-sign-on authentication), it is confronted with the single-point-of-failure problem, since the security of the scheme relies on the reliability of the identity server (i.e., PRF server). If the system-level secret is leaked, it is vulnerable to off-line DGA. Constructing a secure single-sign-on authentication scheme that addresses the single-point-of-failure problem is one of the primary motivations of PASTA [12] (the first password-based threshold authentication scheme). In PASTA, multiple identity servers are introduced to protect passwords and issue authentication tokens in a threshold way. As long as a threshold number of identity servers remains inaccessible to adversaries, the security of passwords against off-line DGA is ensured. PROTECT goes one step beyond PASTA and can further resist compromised key on identity servers. Therefore, compared with the single-sign-on authentication scheme which is constructed on [8]–[10], PROTECT provides a stronger security guarantee. Furthermore, since the target problem and design goals of the schemes in [8]–[10] are different from those of PROTECT, the single-sign-on authentication scheme based on these schemes incurs more costs in terms of communication and computation than PROTECT.

Another line of work to harden passwords is that of Diomedous et al. [11]. The authors propose a password-hardening service for web applications that are constructed over TLS. Generally, in a web application, two entities are involved: a user and a web server. The web server serves as an authenticator to authenticate the user based on her/his password. TLS enables the user to communicate with the web server in a way that prevents eavesdropping, tampering, or message forgery [13]. In such the application, the web server has a private key $kws$ and users' passwords are protected by $kws$. In particular, for a password $psw$, the corresponding authentication credential stored on the web server essentially has the form of $\mathsf{MAC}(kws, psw)$, where $\mathsf{MAC}$ is a secure message authentication code (MAC) algorithm and $kws$ serves as the MAC key. Without $kws$, an adversary cannot crack $psw$. However, the scheme in [11] targets at thwarting external adversaries and bears the assumption that the web server would not leak $kws$. By comparison, in PROTECT, we consider the identity servers (i.e., the authenticator) may be compromised multiple times over a long period of time.

In the revised manuscript, we have further enhanced the related work section. More related works are cited and analyzed, and comparisons between PROTECT and these schemes are provided.

## III. Response to Reviewer #3

**Comment 3.1:** *Authentication plays an important role in today's IT environment. Authors presented a password-based threshold single-sign-on authentication scheme that thwarts adversaries who can compromise identity server. To demonstrate their concept author had conducted a comprehensive performance evaluation, which shows good efficiency on the user side in terms of computation and communication and proves that it can be easily deployed on mobile devices. The current implementation is in C++ with library from https://libraries.docs.miracl.com, it would be more appropriate to have this implementation in android with further optimizing the memory used. However, this is only suggestion but does not effect the recommendation for acceptance.*

**Response 3.1:** Thanks for your positive comments. For the future work, we will test PROTECT in actual systems.

---

Thank you again for the valuable comments and suggestions, we hope our revision is satisfaction.

## References


[1] D. Boneh, B. Lynn, and H. Shacham. "Short Signatures from the Weil Pairing." In Proc. ASIACRYPT, pp. 514–532, 2001.
[2] A. Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-group Signature Scheme." In Proc. PKC, pp. 31–46, 2003.
[3] R. Morris and K. Thompson. "Password Security: A Case History." Communications of the ACM, vol. 22, no.11, pp. 594–597, 1979.
[4] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang. "Targeted Online Password Guessing: An Underestimated Threat." In Proc. CCS, pp. 1242–1254, 2016.
[5] M. L. Mazurek, S. Komanduri, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, P. G. Kelley, R. Shay, and B. Ur. "Measuring Password Guessability for An Entire University." In Proc. CCS, pp. 173–186, 2013.
[6] Kerberos: The Network Authentication Protocol. https://web.mit.edu/kerberos/.
[7] G. Ye, Z. Tang, D. Fang, Z. Zhu, Y. Feng, P. Xu, X. Chen, and Z. Wang. "Yet Another Text Captcha Solver: A Generative Adversarial Network Based Approach." In Proc. CCS, pp. 332–348, 2018.
[8] A. Everspaugh, R. Chaterjee, S. Scott, A. Juels, and T. Ristenpart. "The Pythia PRF Service." In Proc. USENIX Security, pp. 547–562, 2015.
[9] J. Schneider, N. Fleischhacker, D. Schroder, and M. Backes. "Efficient Cryptographic Password Hardening Services from Partially Oblivious Commitments." In Proc. CCS, pp. 1192–1203, 2016.
[10] R. W. F. Lai, C. Egger, D. Schroder, and S. S. M. Chow. "Phoenix: Rebirth of a Cryptographic Password-Hardening Service." In Proc. USENIX Security, pp. 899–916, 2017.
[11] C. Diomedous and E. Athanasopoulos. "Practical Password Hardening Based on TLS." In Proc. DIMVA, pp. 441–460, 2019.
[12] S. Agrawal, P. Miao, P. Mohassel, and P. Mukherjee. "PASTA: PASsword-based Threshold Authentication." In Proc. CCS, pp. 2042–2059, 2018.
[13] The Transport Layer Security (TLS) Protocol Version 1.2, https://tools.ietf.org/html/rfc5246