This work will appear in Proc. of the 41st International Symposium on Reliable Distributed Systems (SRDS 2022)

Reliable Password Hardening Service with Opt-Out

Chunfu Jia
Nankai University
cfjia@nankai.edu.cn

Shaoqiang Wu
Nankai University
wushaoqiang@mail.nankai.edu.cn

Ding Wang
Nankai University
wangding@nankai.edu.cn

***Abstract*—As the most dominant authentication mechanism, password-based authentication suffers catastrophic offline password guessing attacks once the authentication server is compromised and the password database is leaked. Password hardening (PH) service, an external/third-party crypto service, has been recently proposed to strengthen password storage and reduce the damage of authentication server compromise. However, all existing schemes are unreliable in that they overlook the important *restorable* property: PH service opt-out. In existing PH schemes, once the authentication server has subscribed to a PH service, it must adopt this service *forever*, even if it wants to stop the external/third-party PH service and restore its original password storage (or subscribe to another PH service).**

To fill the gap, we propose a new PH service called PW-Hero that equips its PH service with an option to terminate its use (i.e., opt-out). In PW-Hero, password authentication is strengthened against offline attacks by adding external secret spices to password records. With the opt-out property, authentication servers can proactively request to end the PH service after successful authentications. Then password records can be securely migrated to their traditional salted hash state, ready for subscription to other PH services. Besides, PW-Hero achieves all existing desirable properties, such as comprehensive verifiability, rate limits against online attacks, and user privacy. We define PW-Hero as a suite of protocols that meet desirable properties and build a simple, secure, and efficient instance. Moreover, we develop a prototype implementation and evaluate its performance, establishing the practicality of our PW-Hero service.

***Index Terms*—Password-based authentication, Password hardening service, Opt-out option, Offline password guessing attack**

I. Introduction

Password authentication plays an essential role in protecting our digital assets. However, securely implementing password authentication is difficult. Generally, the authentication server needs to store users' passwords in salted-hash form (see the upper part of Fig. 1) as recommended by major standards like NIST-800-63B [1] and the NCSC guideline [2]. Once this salted password file is leaked (see the 3 billion Yahoo leakage [3]), passwords can be recovered relatively easily by offline password guessed because users' passwords follow the Zipf's law [4]. For concrete real-world successful/damaging offline password guessing attacks, see the cracking campaigns of Ashley-Madison [5] and Mall.cz [6].

A promising solution to resist offline password guessing attacks is the password hardening (PH) service [7]–[10]. It introduces an external crypto server that provides cryptographic functions, such as pseudo-random functions (PRF), to help authentication servers generate password records hardened by external secrets. As shown in the low part of Fig. 1, an authentication server carries out the PH registration protocol with a PH server. After that, the authentication server generates an "encrypted" password record, which is a PRF value with a secret key known only to the PH server. Thus, without this key, even if the authentication server is compromised and the record is leaked, attackers cannot crack the password offline. Attackers can only turn to online attacks by requesting the PH verification protocol to confirm each password guess, while the PH server can resist the online attacks by rate-limiting verification requests. In a nutshell, a PH service liberates authentication servers from cryptographically protecting passwords by providing external crypto functions. Note that PH protocols are transparent to users[1], and users' password operations (i.e., register and login) remain as usual.

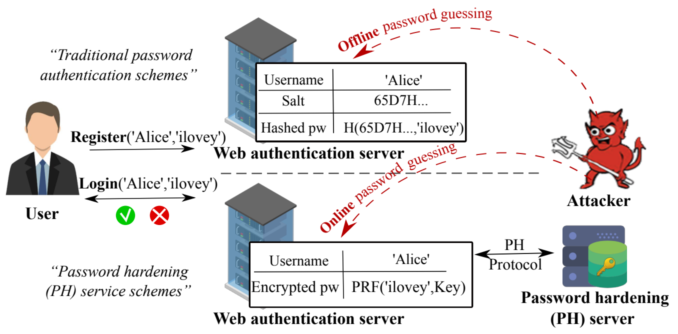

"Traditional password authentication schemes"
Username | 'Alice'
Salt | 65D7H...
Hashed pw | H(65D7H...,'ilovey')
Offline password guessing
Register('Alice','ilovey')
Web authentication server
Login('Alice','ilovey')
User
Attacker
Online password guessing
"Password hardening (PH) service schemes"
Username | 'Alice'
Encrypted pw | PRF('ilovey',Key)
PH Protocol
Web authentication server
Password hardening (PH) server

Fig. 1: Password authentication with a password hardening service will transform the canonical, catastrophic *offline* password guessing attack (upon an authentication server compromise) into a remediable *online* password guessing attack (e.g., resisted by rate-limiting and lockout).

A. Motivations

Due to the above desirable advantages of the PH service, it has drawn considerable attention [8]–[10]. A number of PH schemes have been successively proposed, such as Pythia [8], PO-COM [9], and Phoenix [10]. Among them, Pythia fares poorly on performance, with registration and verification protocols costing three times more time than the other two. Lai *et al.* [10] pointed out that PO-COM cannot withstand offline attacks after a single validation. Therefore, Phoenix is the best existing PH scheme regarding efficiency and security overall.

[1]Therefore, for clarity, we omit the user and refer to the PH server as the server for the PH protocols and the authentication server as the client.

However, we find that Phoenix [10] has some shortcomings when assuming that the PH server can be compromised.[2] (1) In the registration protocol of Phoenix, the values returned from the server are unverifiable. This unverifiability may cause the client to be tricked by malicious servers into storing an incorrect password record and then deleting the original password, resulting in permanent loss of password verification information. (2) The username and verification result are public to the server. This discloses user behavior privacy, such as what sites the user registered to and when the user logged in. Furthermore, (3) in the multi-tenant mode[3], all Phoenix clients share the same secret key. This key share leads to an unreasonable situation where all clients must perform a key rotation when any client requests one.

Besides the above defects in existing PH schemes, most crucially, we find that they all [8]–[10], [12], [13] are unreliable because of overlooking the important property: PH service opt-out. Once a client uses the existing PH services, it cannot get out of them securely and efficiently on the premise that the registered passwords remain valid to authenticate. However, there inevitably are situations where the client needs to withdraw from the PH service. For example, the PH service operator intends to shut down her PH service, and a client wants to stop renewing the current service and subscribe to a new-generation one. Therefore, an exit mechanism is indispensable for complete PH schemes. It is helpful in increasing the reliability of PH services and user confidence in PH services. To this end, we define the opt-out property, which means that a legitimate client can exit the PH service with the assistance of the server and generate new password records without adding external secrets. After that, the client can authenticate independently. Besides, this exit process should be secure, efficient, and verifiable.

In addition, a well-summarised property set can guide the design and comparative evaluation of PH schemes. However, there is no comprehensive and systematic summary of the properties of the PH service. Its absence leads to existing PH schemes asserting their advantages while overlooking disadvantages. Phoenix [10], for example, has better performance than Pythia but lacks the verifiability noted early in the Pythia study [8]. To solve this issue, we conclude all properties of the PH service in Section II.

B. Challenges

Our goal is to build a new PH service that meets the desirable properties of the prior schemes and covers their shortages. Designing such a PH scheme is rather challenging.

Opt-out property is challenging to obtain in the existing schemes, and we need to achieve it from scratch. In Pythia [8] and Phoenix [10], the client can only wait until each user logins with her correct password and then recalculates a new password record based on the password plaintext. In the simplified Phoenix [14], where the password record is represented as $H_{\mathcal{C}}(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}} \cdot H_{\mathcal{S}}(\mathsf{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}}$, the client can ask for $H_{\mathcal{S}}(\mathsf{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}}$ for each password by registration protocols and obtain a service-independent record $H_{\mathcal{C}}(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}}$. Here, $n_{\mathcal{C}}$ and $n_{\mathcal{S}}$ are random values, and $k_{\mathcal{C}}$ and $k_{\mathcal{S}}$ are the secret keys of the client and the PH server, respectively. However, the two exit methods are undesirable. First, the exit processes are lengthy, increasing security risk. Second, they are not batch executed for all registered passwords and are therefore cumbersome. Third, the communication traffic required for exiting the simplified Phoenix is related to the size of the password database, so it is inefficient and poorly scalable. In addition, the correctness of the newly generated password records should be verifiable. To support opt-out with verifiability, simply modifying the existing PH schemes based on our PH definition, which first formally introduces the exit phase in Section III, is inconvenient.

Some privacy and functional properties are conflict and challenging to balance. First, achieving both user login privacy and rate limits is tricky. If a PH server doesn't learn login verification results, it is only capable of rate-limiting the total number of legitimate users and attackers' verification requests, which is the tactic adopted by [8], [13]. Second, providing an opt-out would compromise the privacy of the external secret because it allows a way to get the external secret by exit protocols and recover efficient offline attacks. Therefore, appropriate trade-offs are needed for these properties that are not compatible with each other. In this work, we present out-of-protocol methods to alleviate possible shortcomings.

C. Our Contributions

Our main contribution is that, for the first time, we present the <u>**P**</u>ass<u>**W**</u>ord <u>**H**</u>ardening s<u>**er**</u>vice equipped with the crucial <u>**o**</u>pt-out property, named PW-Hero. We set up a decryption protocol to achieve opt-out, where the hardened password record stored by the client is stripped to its traditional salted hash form. In PW-Hero, the decryption meets three conditions: the decrypted password form is convenient for introducing other PH services, the correctness of the decrypted password is verifiable, and the decryption protocol can batch decrypt with a compact communication traffic. Due to passwords always participating in PH protocols in a hash form, transitioning to their salted hash will not affect subsequent access to other PH services. The client can quick exit: By asking for the decryption protocol, the client receives a constant-sized decryption token with which it can quickly batch transfer all password records to their salted hash. In addition, the client keeps the hash of a pivotal intermediate value to verify that the decryption protocol yields the correct password hash. Note that the PH server should strictly verify the client's identity before allowing it to exit. The client authentication can prevent attackers from abusing the exit function to evade password protection from external PH crypto services.

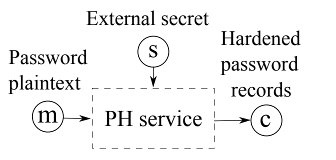
We define the PW-Hero as a suite of protocols in Section III. And the PW-Hero definition is general enough to

[2]In general, we assume that both the identity server and the PH server can be compromised for the PH service schemes, but not both.

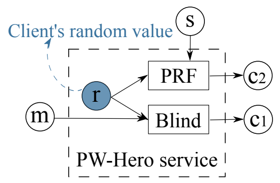
[3]A PH service can be built by the enterprise for internal use only, or it can be publicly deployed on the cloud to provide services for multiple authentication servers simultaneously, that is, the multi-tenant mode [11].

capture all properties specified for PH services by prior works. Besides the above opt-out, we enrich the PH property set with other crucial properties, including username privacy and verification-result privacy. In addition, we give the security definition of PW-Hero in the way of game-based challenges in Appendix B and formally prove them in Appendix C.

To prove the practicality of PW-Hero, we build a simple construction based on our definition of PW-Hero. As a brief schematic shown in Fig. 2b, we set a two-layer mechanism. Instead of the password m, a random value r is involved to interactively calculate a PRF value as c_2. And r also is used as a key to blind m as c_1. When finished, r is securely deleted. The two-layer mechanism introduces an external secret s into the verification record c of the password m without sending any value derived from the password to the server. The detailed PW-Hero protocols are described in Section IV. Moreover, we implement the PW-Hero scheme and test its performance compared with all existing PH schemes [8]–[10] in Section V. The experimental results show that our PW-Hero scheme is as efficient as Phoenix [10].



(a) PH adds external secret s into password m to generate hardened c.



(b) Two-layer hardening mechanism of our PW-Hero scheme.

Fig. 2: Schematic diagram of our PW-Hero scheme.

D. Related Work

Faced with the gap between the realistic demand for the password-based authentication mechanism and the password security problems caused by rampant offline guessing attacks, Facebook [7] built a cryptographically secure solution by introducing an external PRF-Service to strengthen password records. The authentication server must interact with the external PH server to verify that the candidate password matches its stored password verification record. Neither the authentication server nor the PH server can do this alone. Therefore, offline attack cracking passwords becomes impossible.

Everspaugh *et al.* [8] provide the first formal treatment of the PH service and propose Pythia, based on a new cryptographic primitive called the verifiable partially-oblivious PRF (PO-PRF). Pythia computes $\mathsf{pair}(H(t), H(\mathsf{pw}))^{ks}$ as the verification record of the password pw, where t is an identifier of the username un, termed tweak. At the client's request $(t, H(\mathsf{pw})^n)$, where n is a random value, Pythia hides the password pw by exponentially blinding but reveals t to the PH server. Based on the username tweak, the server can fine-grained rate-limit requests per user. Additionally, the verifiability of PO-PRF ensures that faulty returns from the server cannot earn clients' trust. However, pair is a bilinear pairing operation with poor performance. Since then, two more schemes have been proposed, PO-COM [9] and Phoenix [10]. Schneider *et al.* [9] rely on a weaker cryptographic primitive called partially oblivious commitments (PO-COM) and propose an efficient construction based on the multiplicative group operations. However, Lai *et al.* [10] soon find that the PO-COM scheme is vulnerable to offline attacks after a single verification request and revive it with a new construction Phoenix, three times as efficient as Pythia. Later, Lai *et al.* [12] propose password-hardened encryption (PHE) based on their Phoenix. PHE is an external crypto server that encrypts user data and recovers it when a user supplies the correct password. Brost *et al.* [13] deal with the single point of failure introduced by PHE. In addition, Diomedous *et al.* [15] implement a TLS-based password hardening scheme in which the cryptographic functions equipped in TLS-enabled web servers are modified to act as a password hardening service.

Moreover, password policy [16], strength meter [17], password manager [18], two/multi-factor authentication [19], [20], device-enhanced password authentication [21], [22], password crack alarm mechanism [23]–[25], and more [26] can improve password security.

II. Overview

In this work, we define the password hardening (PH) service with the opt-out, i.e., PW-Hero, based on the property set of the PH service we envision, and then construct an instantiation to verify PW-Hero's practicality. Overall, we follow the idea of perfecting the PH property set, defining the PH service with opt-out as PW-Hero, and then instantiating PW-Hero.

This section mainly suggests the PH property set. Generally speaking, a comprehensive and systematical set of the properties of the PH service is available for PH schemes to be faultlessly designed and objectively assessed.

A. Properties of the Password Hardening Service

Based on the existing studies [8]–[10], [13], we aggregate the properties of the PH services and classify them into five types $A \sim G$. In particular, we provide some properties not mentioned before, which enrich the PH property set to match our PW-Hero. Specifically, these new properties are opt-out, username anonymity, and verification-result privacy. Opt-out is an entirely new property. While username anonymity and verification-result privacy are not explicitly stated in prior work, Pythia [8] meets them, i.e., PH service cannot obtain usernames and verification results through PH protocols, but Phoenix [10] does not. We generalize them into the property set of PW-Hero. In addition, we emphasize the username binding and the password binding to ensure that attackers will not bypass the service's rate limit by forging the username or password in the verification request. And we also emphasize that all values returned from the service should be verifiable.

The whole properties of our envisioned PW-Hero service are summarized and classified as follows.

- **Classification A** - Disable offline password guessing attacks. It is the starting point of the research on PH service.

P1: Password privacy. The PH server should learn nothing about the password and its record. This eliminates the introduced external server from cracking the user password directly or through offline attacks.

P2: External secret privacy. External secrets are kept secret and should not be disclosed to clients via the PH protocols. This ensures that any offline attack is not feasible, even though the password record storage is spilled.

- **Classification B** - Limit online password guessing attacks. The PH server should limit the number of verified passwords by online requests on a per-account basis over a certain time.

P3: Rate-limit verification. The PH server should rate-limit password verification requests per user. This effectively reduces the number of passwords guessed by online attackers, reducing the likelihood of password cracking.

P4: Username binding. It should be impossible for the username involved in the verification protocol to be forged. This ensures that an attacker cannot construct a fake username to bypass the rate limit imposed by the PH server under the original username.

P5: Password binding. One verification should only be able to verify one candidate password. This ensures that limiting requests is equivalent to limiting verified passwords.

- **Classification C** - Prevent leakages of user privacy.

P6: Username anonymity. The PH server should only link verification requests involving the same username by virtue of the username identifier but not learn the username itself. This ensures that the PH server does not know which user is registering and logging in.

P7: Verification-result privacy. The PH server should not be able to learn password verification results. This avoids the external PH server analyzing user login behaviors.

- **Classification D** - Respond to the malicious/compromised PH server.

P8: Verifiability. The authentication server should be able to verify all values returned from the PH server to prevent itself from being spoofed by the malicious/compromised PH server. If the return value verification fails, indicating that the PH server is malicious/compromised, the authentication server should promptly alert the PH service operator to restore the security of her PH service. Moreover, during the registration phase, verifiability can troubleshoot wrong return values and prevent the authentication server from falling into the dilemma of missing password verification information. Inappropriately, Phoenix [10] overlooks verifiability in the registration phase.

P9: Opt-out. The legitimate authentication server can opt out its PH service, bringing password verification records back to a specific state from which other services can be introduced. This allows the authentication server to end use of the PH service, once it is found to be malicious. PW-Hero adopts the traditional salted hash as the backed state of password authentication records.

- **Classification E** - Rotate keys. The PH services should support key rotations as the current password records or service key becomes insecure or needs to be updated periodically.

P10: Individual key rotation. In multi-tenant mode, the PH server should provide each authentication server with a unique secret key that can be updated independently. This ensures that when one authentication server needs to update the key, it does not affect others.

P11: Master key rotation. The PH server should be able to rotate its master key to refresh the security of the entire PH service.

P12: Secure key rotation. It means that key rotation can render old password records completely incomprehensible and useless the old service key to verify passwords for new records. In addition, other security properties do not diminish due to key rotations.

- **Classification F** - Efficiency and scalability.

P13: Compact traffic. It means that the token communicated by protocols should be constant-size, independent of the size of the password database.

P14: Batch update. The authentication server should be able to update all its stored password records in batches, not one by one, through a protocol.

- **Classification G** - Improve user experience.

P15: User transparency. The PH service is transparent to users, who enter a username and password to register and log in to the authentication server as usual.

P16: Low latency. The PH protocols should be single-round and computationally efficient.

Tab. III , at the end of this paper, summarizes the properties possessed by each existing scheme.

In addition, since this paper focuses on the PH service with one single server, the issue of the single point of failure is not addressed; see the threshold schemes that specifically address the single point of failure of services [13], [27]–[29].

B. Property Discussion

We answer the following key questions in this section.

- *Q1:* Why PW-Hero adopts the master-slave key mode, where individual keys are derived from a master key?
- *Q2:* Will rate-limiting all requests indiscriminately instead of requests with wrong passwords affect normal user logins?
- *Q3:* How can we prevent attackers from impersonating the authentication server and requesting to exit the PH service?

The master-slave key model is suitable for multi-tenant scenarios. The master key held by the PH server is used to derive an individual key for each authentication server participating in the PH service. And the individual key is used as the external secret specific to the authentication server to strengthen its password records. In this way, one authentication server asking for a key rotation or exiting will only affect her individual key, and the other authentication servers will not incur any cost for these requests. In addition, the PH server only needs to store one master key instead of keeping all individual keys, which can be temporarily derived with the master key when used.

Incidentally, Phoenix [10] can be improved by introducing the master-slave key model to solve the problem that its keys cannot be updated independently for each client.

Rate limiting all verification requests could have little to no effect on normal user logins. Verification-result privacy causes the PH server indiscriminately rate-limiting all requests. This rate limit may lead to concern; for example, allowing three attempts before locking out the account for five minutes can cause significant delays for the legitimate user who wants to log in four times in a row. As a simple but effective solution, adjusting the time interval, such as 100 attempts per three hours, will make it equally hard for offline attackers to crack passwords while not inconveniencing legitimate users.

To avoid abusive opt-out by attackers, the PH server must authenticate the requester before responding to an exit request. Besides, the authentication should be robust, such as the secure email or telephone confirmation, to be still valid even if authentication servers are compromised. In this way, PH services can prevent attackers from impersonating the authentication server to exit the PH service and activate the offline attack.

III. Password Hardening Service with Opt-out

We define our PH service with opt-out and call it PW-Hero, which comes from the "**P**ass**W**ord **H**ardening s**er**vice with **o**pt-out". PW-Hero meets all properties we envisioned in Section II. Nevertheless, we assume that the PW-Hero server and the authentication server will not be compromised simultaneously because, in this case, any PH service is equivalent to a typical salted hash scheme. We assume that when the authentication server is compromised, its secret key is leaked to attackers along with password records. We also assume that all protocols are to transmit messages through a secure communication channel.

A. Definition of PW-Hero

We define PW-Hero to be a suite of protocols based on the related definitions of the PH services in [10] [9]. Due to user transparency, we call a PH-Hero protocol a two-party protocol, including an authentication server and a PH server. We call an authentication server a client $\mathcal{C}$ and call a PW-Hero server a server $\mathcal{S}$ from now on.

Definition 1. PW-Hero is a password hardening service with the opt-out, consisting of a tuple of cryptographic algorithms $\Phi = \{\text{Setup}, \text{CKeyGen}, \text{SKeyGen}, \langle\mathcal{C},\mathcal{S}\rangle_{\text{reg}}, \langle\mathcal{C},\mathcal{S}\rangle_{\text{ver}}, \langle\mathcal{C},\mathcal{S}\rangle_{\text{rot}}, \langle\mathcal{C},\mathcal{S}\rangle_{\text{dec}}\}$. Concretely, as following five phases:

Setup phase: On input the security parameter λ, $\mathsf{Setup}(1^\lambda)$ returns the public parameter pp, which is public to all entities, so that omitted by us. Then, the client key generation algorithm $\mathsf{CKeyGen}()$ returns a secret key $k_\mathcal{C}$ of the client. And on inputs $\mathsf{mk}_\mathcal{S}$ and a client state identifier[4] w, the server key generation algorithm $\mathsf{SKeyGen}(\mathsf{mk}_\mathcal{S}, w)$ generates a private key $\mathsf{sk}_\mathcal{S}$ for w and a public key $\mathsf{pk}_\mathcal{S}$ which is sent to the client. If the parameter $\mathsf{mk}_\mathcal{S}$ is none, $\mathsf{SKeyGen}$ first randomly generates a master key.

Registration phase: On inputs of the client state identifier w, a username un, a password pw, which are from the client, and the master key $\mathsf{mk}_\mathcal{S}$ of the server, the registration protocol $\Phi.\langle\mathcal{C}(w,\mathsf{un},\mathsf{pw},\mathsf{pk}_\mathcal{S},k_\mathcal{C}),\mathcal{S}(\mathsf{mk}_\mathcal{S})\rangle_{\text{reg}}$ returns the password verification record T to the client and a username identifier h_0 to the server. Finally, the server initializes a counter c for h_0, and the client stores the T. Note that not only when a new password is registered, but also when a registered user re-sets/changes her password, the registration phase will also be executed.

Verification phase: On inputs of the client state identifier w, a username un, and a candidate password pw', the verification protocol $\Phi.\langle\mathcal{C}(w,\mathsf{un},\mathsf{pw}',\mathsf{pk}_\mathcal{S},k_\mathcal{C}),\mathcal{S}(\mathsf{mk}_\mathcal{S})\rangle_{\text{ver}}$ returns an authentication result ("Accept"/"Reject"/abort[5]) to the client and returns a username identifier h_0 to the server. The server increments h_0's counter by 1.

Key rotation phase: There are two main key rotation protocols, the external secret key ($\mathsf{sk}_\mathcal{S}$) rotation protocol $\langle\mathcal{C}(w,w',\mathsf{pk}_\mathcal{S}),\mathcal{S}(\mathsf{mk}_\mathcal{S})\rangle_{\text{skrot}}$ and the master key ($\mathsf{mk}_\mathcal{S}$) rotation protocol $\langle\mathcal{C}(w,\mathsf{pk}_\mathcal{S}),\mathcal{S}(\mathsf{sk}_\mathcal{S})\rangle_{\text{mkrot}}$. For the former, on inputs of the old client state identifier w and a new w', a secret key rotation protocol returns new Ts for all password records and a new public key $\mathsf{pk}'_\mathcal{S}$ to the client. For the latter, a master key rotation protocol returns new Ts and a new public key $\mathsf{pk}'_\mathcal{S}$ to the client as well and returns a new master $\mathsf{mk}'_\mathcal{S}$ to the server.

Decryption phase: On input of the client state identifier w, a decryption protocol $\Phi.\langle\mathcal{C}(w,\mathsf{pk}_\mathcal{S},k_\mathcal{C}),\mathcal{S}(\mathsf{mk}_\mathcal{S})\rangle_{\text{dec}}$ returns a triple, ("identifier of un", "salt value", "salted hash of pw"), for each record T to the client and returns nothing to the server. After that, the server deletes all counters of the client.

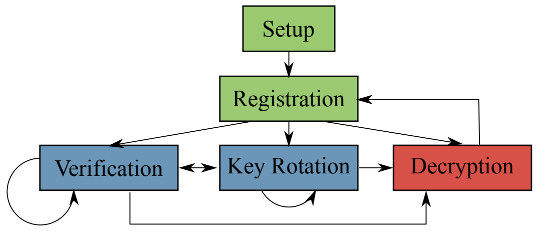


Fig. 3: Five phases of our PH service, PW-Hero.

Fig. 3 illustrates the five phases of our PH service, PW-Hero, and the sequential relationship between them.

B. Security of PW-Hero

The security of our PW-Hero includes obliviousness, hiding, forward security, binding, and privacy. The first three are inherited from previous work [9], [10], but we need to redefine them since the existing definitions do not embody our PW-Hero's newly introduced decryption protocol. In addition, we enrich the meaning of binding security. The privacy is a new

[4]The client state identifier is a random string generated by the client and uniquely identifies the client.

[5]The verification protocol aborts when the cumulative number of verification requests reaches the limit, or the verifiability-related check fails.

security goal based on the properties of Classification C in Section II-A, and we provide its first formal treatment. We formally define the security of the PW-Hero in Appendix B.

1. **Obliviousness**. Any protocols do not leak the password and its verification record to the server. In other words, the server cannot learn the password from the protocol's messages.
2. **Hiding**. Without the external key, an attacker cannot verify a guess of the underlying password of the leaked password verification record.
3. **Forward security**. A key rotation should render the old secret keys and old verification records useless to attackers.
4. **Binding**. In the previous definition, binding means that a password and its verification record are bound, so a malicious/compromised server cannot convince the client to pass wrong passwords and fail correct passwords. We believe that the meaning of binding is insufficient and add that the client cannot trick the server into verifying the request with a forged username or password.
5. **Privacy**. It is terrible if the server can learn what site and when the user logged in from the PH protocols. Therefore, PW-Hero keeps username and verification results private to the client. For the sake of username anonymity, we do not recommend direct transmission of username plaintext, and the random identifier is an appropriate choice, such as $H(\mathsf{un})^k$. Here, the privacy definition focus on the privacy of verification results. In a nutshell, the server should not be informed of the result of whether the candidate password matches the password record.

IV. A PW-HERO SCHEME

We propose a simple construction for PW-Hero based on a two-layer cryptographic mechanism. It is composed of a pseudorandom function (PRF) and a multiplicative blind function (MBF) on a finite cyclic multiplicative group. These two functions have the same parameter r. In MBF, r is used as the blinding factor, and it generates the blinded output c_1 of the input m, $c_1 = H(m)\cdot r$; In PRF, r is the input, and it generates the pseudorandom output c_2 with the key k, $c_2 = r^k$. c_1 and c_2 are taken together as the m's ciphertext hardened with k.

Concretely speaking, in the registration phase, the client $\mathcal{C}$ samples a random value r and sends it instead of the password pw to the server $\mathcal{S}$. Locally on the client side, $\mathcal{C}$ uses r as a random factor to blind the password pw to generate $t_1 = H(\mathsf{pw})\cdot r$[6] (*Layer 1*). Meanwhile, $\mathcal{S}$ computes and returns the PRF value $t_2 = r^{k_\mathcal{S}}$, where $k_\mathcal{S}$ is the secret key (*Layer 2*). Benefiting from the two-layer mechanism, the client does not need to send any password-derived value to the server but a random value r. Here, r is used as an intermediate value to completely isolate the password from the server to ensure password privacy. After that, $\mathcal{C}$ stores the blinded password ciphertext t_1 and the PRF value t_2 as the password records of pw and then securely deletes pw and r.

In the decryption/exit phase, $\mathcal{S}$ returns the external key $k_\mathcal{S}$ to $\mathcal{C}$. $\mathcal{C}$ first decrypts t_2 with the $k_\mathcal{S}$ to get $r' = t_2^{1/k_\mathcal{S}}$ (*Inverse of layer 2*) and then decrypts t_1 with r' to get $H(\mathsf{pw})' = t_1/r'$ (*Inverse of layer 1*). To ensure the correctness of the decryption, i.e., $H(\mathsf{pw})' = H(\mathsf{pw})$, the client needs to record $t_3 = H(r)^{k_\mathcal{S}}$ as well in the registration phase and verify $H(r')^{k_\mathcal{S}}$ during the decryption phase. The client can determine if r' is correct by checking if $H(r)' = t_3^{1/k_\mathcal{S}}$ is equal to the hash of r'. If r is correct, $H(\mathsf{pw})'$ decrypted with r must be correct, that is, $H(\mathsf{pw})' = H(\mathsf{pw})$.

In the verification phase, the client $\mathcal{C}$ uses the login password pw' to solve r', $r' \leftarrow t_1/H(\mathsf{pw}')$ (*Inverse of layer 2*). Then $\mathcal{C}$ samples a random value $n_\mathcal{C}$ to exponentially blind r' to get $x_1 = (r')^{n_\mathcal{C}}$, and send it to the server $\mathcal{S}$. After that, $\mathcal{S}$ returns $y = (r')^{n_\mathcal{C}\cdot k_\mathcal{S}}$. Finally, the client can get $t_2' = (r')^{k_\mathcal{S}}$ after unblinding the return value. By checking whether t_2' is equal to the stored t_2, the client can confirm whether the login password pw' is correct.

Furthermore, we provide the zero-knowledge proofs (ZKP) for returned values, t_2 and y, to ensure that they are calculated according to agreed protocols by $\mathcal{S}$. In particular, to avoid the client falsifying h_0 in the verification phase to bypass the h_0-based rate limit enforced by $\mathcal{S}$, we additionally let clients provide ZKP for the PRF value of h_0.

A. Protocol Description

In this section, we formally describe the protocols of our PW-Hero construction. Φ denotes the protocol set of PW-Hero, and Π denotes the ZKP protocol we adopted, described in Appendix A. In addition, $\mathbb{G}$ denotes a finite cyclic multiplicative group of order q. H_0 and H_1 denote two cryptographic hash functions: $\{0,1\}^* \rightarrow \mathbb{G}$. H_2 denotes a cryptographic hash function: $\{0,1\}^* \rightarrow \{0,1\}^\lambda$. H_3 denotes a cryptographic hash function: $\{0,1\}^* \rightarrow \mathbb{Z}_q$.

1) Setup: As shown in Fig. 4, in the setup phase, Φ.Setup algorithm randomly picks a generator g from $\mathbb{G}$ and sets up four cryptographic hash functions $H_{i|i\in\{0,1,2,3\}}$ and returns them as the public parameter pp. Next, Φ.CKeyGen algorithm samples an integer from $\mathbb{Z}_q$ and sets it as the client key $k_\mathcal{C}$. And Φ.SKeyGen randomly picks a master key $\mathsf{mk}_\mathcal{S}$ from $\mathbb{Z}_q$, generates a private key for $\mathcal{C}$ with the hash of $\mathsf{mk}_\mathcal{S}$ and w, and initializes the public key with $\mathsf{pk}_\mathcal{S} = g^{\mathsf{sk}_\mathcal{S}}$. Finally, the public key $\mathsf{pk}_\mathcal{S}$ is sent to $\mathcal{C}$.

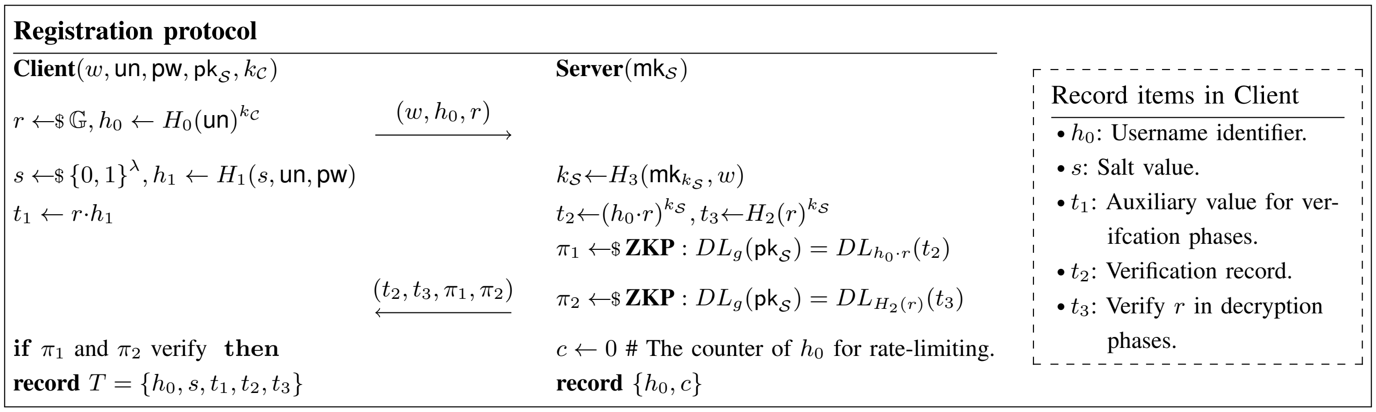
```
Setup($1^\lambda$)
    $g \leftarrow\$ \mathbb{G}$
    $H_{i|i\in\{0,1\}} \leftarrow\$ \mathcal{H}_1 = \{H : \{0,1\}^* \rightarrow \mathbb{G}\}$
    $H_{i|i\in\{1,2\}} \leftarrow\$ \mathcal{H}_2 = \{H : \{0,1\}^* \rightarrow \mathbb{Z}_q\}$
    return $\mathsf{p} = \{g, H_0, H_1, H_2, H_3\}$

CKeyGen()
    $k_\mathcal{C} \leftarrow\$ \mathbb{Z}_q$
    return $k_\mathcal{C}$

SKeyGen($\mathsf{mk}_\mathcal{S}, w$)
    if not $\mathsf{mk}_\mathcal{S}$
        then $\mathsf{mk}_\mathcal{S} \leftarrow\$ \mathbb{Z}_q$
    $k_\mathcal{S} \leftarrow\$ H_3(\mathsf{mk}_\mathcal{S}, w)$
    $\mathsf{sk}_\mathcal{S} \leftarrow\$ k_\mathcal{S}$
    $\mathsf{pk}_\mathcal{S} \leftarrow\$ g^{k_\mathcal{S}}$
    return $\mathsf{mk}_\mathcal{S}, \mathsf{sk}_\mathcal{S}, \mathsf{pk}_\mathcal{S}$
```

Fig. 4: The setup protocol of PW-Hero.

[6] $H(\cdot)$ could be the salted hash algorithm, here abbreviated as $H(\mathsf{pw})$.

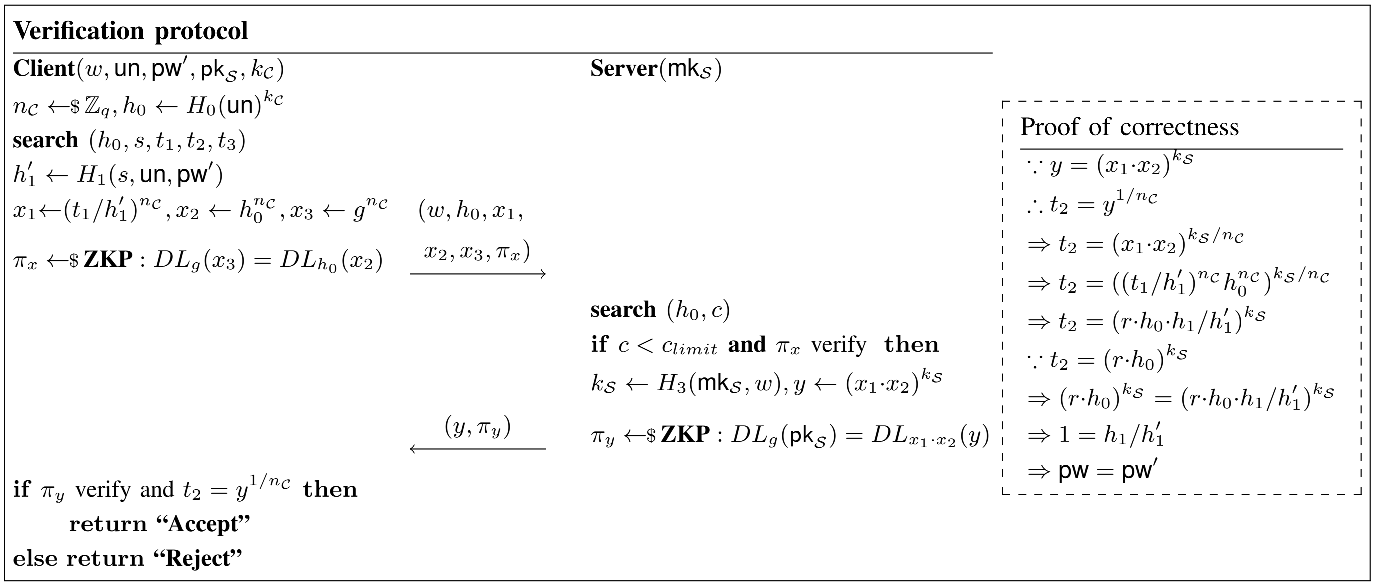


Registration protocol

Client$(w, \mathsf{un}, \mathsf{pw}, \mathsf{pk}_{\mathcal{S}}, k_{\mathcal{C}})$		**Server**$(\mathsf{mk}_{\mathcal{S}})$
$r \leftarrow\$ \mathbb{G}, h_0 \leftarrow H_0(\mathsf{un})^{k_{\mathcal{C}}}$	$\xrightarrow{(w, h_0, r)}$	
$s \leftarrow\$ \{0,1\}^{\lambda}, h_1 \leftarrow H_1(s, \mathsf{un}, \mathsf{pw})$		$k_{\mathcal{S}} \leftarrow H_3(\mathsf{mk}_{k_{\mathcal{S}}}, w)$
$t_1 \leftarrow r \cdot h_1$		$t_2 \leftarrow (h_0 \cdot r)^{k_{\mathcal{S}}}, t_3 \leftarrow H_2(r)^{k_{\mathcal{S}}}$
		$\pi_1 \leftarrow\$ \textbf{ZKP} : DL_g(\mathsf{pk}_{\mathcal{S}}) = DL_{h_0 \cdot r}(t_2)$
	$\xleftarrow{(t_2, t_3, \pi_1, \pi_2)}$	$\pi_2 \leftarrow\$ \textbf{ZKP} : DL_g(\mathsf{pk}_{\mathcal{S}}) = DL_{H_2(r)}(t_3)$
if π_1 and π_2 verify **then**		$c \leftarrow 0$ # The counter of h_0 for rate-limiting.
record $T = \{h_0, s, t_1, t_2, t_3\}$		**record** $\{h_0, c\}$

Record items in Client
- h_0: Username identifier.
- s: Salt value.
- t_1: Auxiliary value for verifcation phases.
- t_2: Verification record.
- t_3: Verify r in decryption phases.

Fig. 5: The registration protocol of PW-Hero.



Verification protocol

Client$(w, \mathsf{un}, \mathsf{pw}', \mathsf{pk}_{\mathcal{S}}, k_{\mathcal{C}})$		**Server**$(\mathsf{mk}_{\mathcal{S}})$
$n_{\mathcal{C}} \leftarrow\$ \mathbb{Z}_q, h_0 \leftarrow H_0(\mathsf{un})^{k_{\mathcal{C}}}$		
search (h_0, s, t_1, t_2, t_3)		
$h_1' \leftarrow H_1(s, \mathsf{un}, \mathsf{pw}')$		
$x_1 \leftarrow (t_1/h_1')^{n_{\mathcal{C}}}, x_2 \leftarrow h_0^{n_{\mathcal{C}}}, x_3 \leftarrow g^{n_{\mathcal{C}}}$	$(w, h_0, x_1,$	
$\pi_x \leftarrow\$ \textbf{ZKP} : DL_g(x_3) = DL_{h_0}(x_2)$	$\xrightarrow{x_2, x_3, \pi_x)}$	
		search (h_0, c)
		if $c < c_{limit}$ **and** π_x verify **then**
		$k_{\mathcal{S}} \leftarrow H_3(\mathsf{mk}_{\mathcal{S}}, w), y \leftarrow (x_1 \cdot x_2)^{k_{\mathcal{S}}}$
	$\xleftarrow{(y, \pi_y)}$	$\pi_y \leftarrow\$ \textbf{ZKP} : DL_g(\mathsf{pk}_{\mathcal{S}}) = DL_{x_1 \cdot x_2}(y)$
if π_y verify and $t_2 = y^{1/n_{\mathcal{C}}}$ **then**		
return "Accept"		
else return "Reject"		

Proof of correctness

$$\because y = (x_1 \cdot x_2)^{k_{\mathcal{S}}}$$
$$\therefore t_2 = y^{1/n_{\mathcal{C}}}$$
$$\Rightarrow t_2 = (x_1 \cdot x_2)^{k_{\mathcal{S}}/n_{\mathcal{C}}}$$
$$\Rightarrow t_2 = ((t_1/h_1')^{n_{\mathcal{C}}} h_0^{n_{\mathcal{C}}})^{k_{\mathcal{S}}/n_{\mathcal{C}}}$$
$$\Rightarrow t_2 = (r \cdot h_0 \cdot h_1/h_1')^{k_{\mathcal{S}}}$$
$$\because t_2 = (r \cdot h_0)^{k_{\mathcal{S}}}$$
$$\Rightarrow (r \cdot h_0)^{k_{\mathcal{S}}} = (r \cdot h_0 \cdot h_1/h_1')^{k_{\mathcal{S}}}$$
$$\Rightarrow 1 = h_1/h_1'$$
$$\Rightarrow \mathsf{pw} = \mathsf{pw}'$$

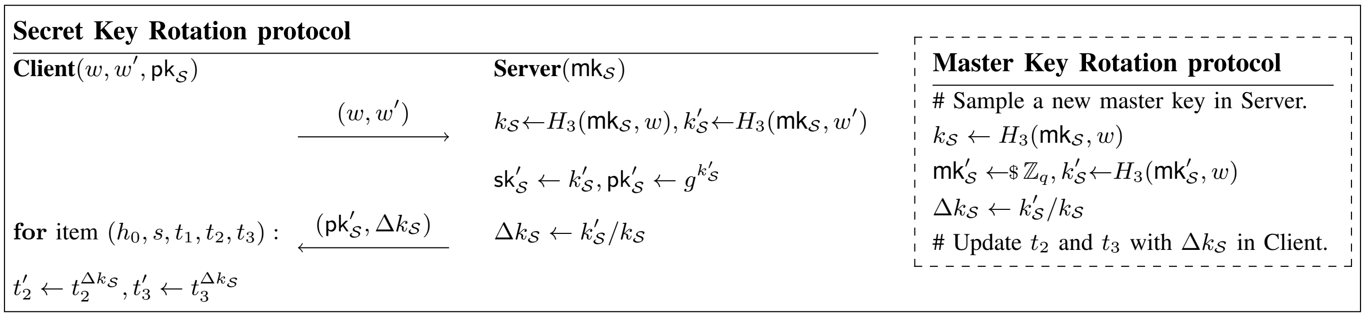
Fig. 6: The verification protocol of PW-Hero.

2) Registration: Fig. 5 shows the registration protocol $\Phi.\langle \mathcal{C}(w, \mathsf{un}, \mathsf{pw}, \mathsf{pk}_{\mathcal{S}}, k_{\mathcal{C}}), \mathcal{S}(\mathsf{mk}_{\mathcal{S}})\rangle_{\text{reg}}$. The client generates an identifier h_0[7] of username un and samples a ramdom vale r from $\mathbb{Z}_q$. Then, the client sends her state identifier w, h_0, and r to the server. Next, the client randomly picks a salt value s from $\mathbb{Z}_q$ and calculates a salted hash h_1 for the username-and-password pair $(\mathsf{un}, \mathsf{pw})$. Then the password salted hash is multiplicatively blinded with the random value r to generate the hardened password record t_1. In parallel, after receiving (w, h_0, r), the server generates a private key $k_{\mathcal{S}}$[8] for the client w with her master key $\mathsf{mk}_{\mathcal{S}}$ and calculates a PRF values $t_2 \leftarrow (r \cdot h_0)^{k_{\mathcal{S}}}$ and $t_3 \leftarrow H_2(r)^{k_{\mathcal{S}}}$, which are sent to the client together with their ZPKs. If the proofs are verified, the client securely deletes pw and r. Finally, the client records (h_0, s, t_1, t_2, t_3) as the verification records of $(\mathsf{un}, \mathsf{pw})$, and the server initializes a counter for h_0.

3) Verification: Fig. 6 shows the verification protocol $\Phi.\langle \mathcal{C}(w, \mathsf{un}, \mathsf{pw}', \mathsf{pk}_{\mathcal{S}}, k_{\mathcal{C}}), \mathcal{S}(\mathsf{mk}_{\mathcal{S}})\rangle_{\text{ver}}$. The client re-generates the username identifier h_0 of un and uses it as the index to search its password verification record (s, t_1, t_2, t_3). After that, the client use the searched salt s to calculate the salted hash h_1' of the login username pw' by $h_1' \leftarrow H_1(s, \mathsf{un}, \mathsf{pw}')$. With h_1', the client solve $r' \leftarrow t_1/h_1'$ from the password record t_1. Then, a nonce $n_{\mathcal{C}}$ is randomly selected from $\mathbb{Z}_q$, and the client use it to exponentially blind r' and h_0, by $x_1 \leftarrow r'^{n_{\mathcal{C}}}$ and $x_2 \leftarrow h_0{}^{n_{\mathcal{C}}}$. In addition, the client generates a ZKP π_x for x_2. When the proof π_x is verified, and the counter of h_0 does not exceed the preset limit c_{limit}, the server recovers the client-specific key $k_{\mathcal{S}}$, and then calculates the PRF value $y \leftarrow (x_1 \cdot x_2)^{k_{\mathcal{S}}}$ and the corresponding ZKP

[7]A client private $k_{\mathcal{C}}$ can be introduced the username hash if the username privacy is a concern, such as $h_0 = H_0(\mathsf{un})^{k_{\mathcal{C}}}$.

[8]We derive a private key for each client with the server's master key, and the private key helps its client to give rise to isolated PRF. This facilitates individual rotations of PRF keys in a multi-tenant scenario. Pythia [8] takes the same method, but unlike Pythia, our client state identifiers are stored on the client-side and are only handed to the server when the protocol is running. Whereas in Pythia the client state identifiers are called the ensemble selectors and are always stored at the server.



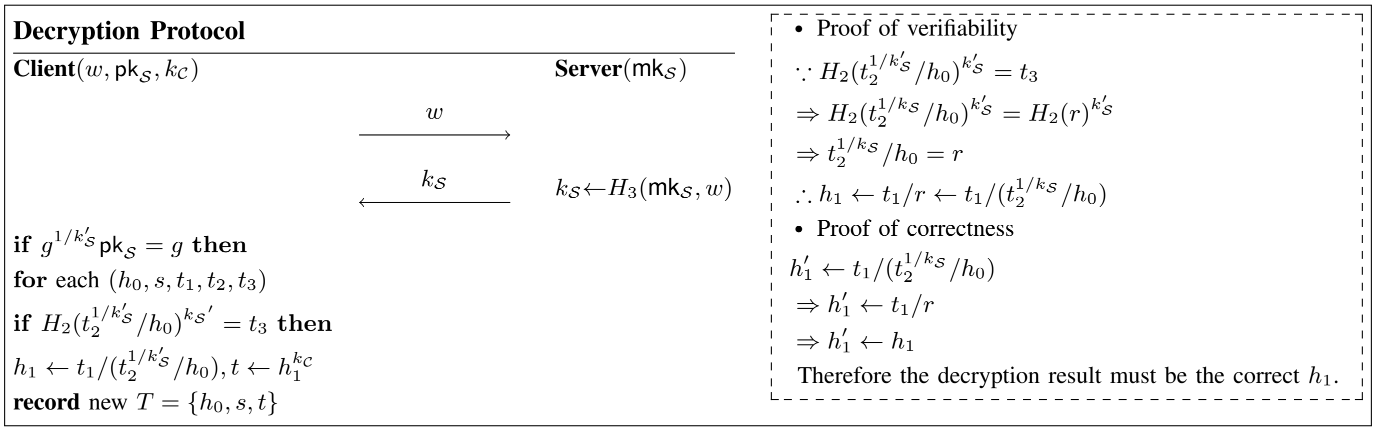
Secret Key Rotation protocol

Client$(w, w', \mathsf{pk}_{\mathcal{S}})$		**Server**$(\mathsf{mk}_{\mathcal{S}})$
	$\xrightarrow{(w, w')}$	$k_{\mathcal{S}} \leftarrow H_3(\mathsf{mk}_{\mathcal{S}}, w), k'_{\mathcal{S}} \leftarrow H_3(\mathsf{mk}_{\mathcal{S}}, w')$
		$\mathsf{sk}'_{\mathcal{S}} \leftarrow k'_{\mathcal{S}}, \mathsf{pk}'_{\mathcal{S}} \leftarrow g^{k'_{\mathcal{S}}}$
for item (h_0, s, t_1, t_2, t_3) :	$\xleftarrow{(\mathsf{pk}'_{\mathcal{S}}, \Delta k_{\mathcal{S}})}$	$\Delta k_{\mathcal{S}} \leftarrow k'_{\mathcal{S}}/k_{\mathcal{S}}$
$t'_2 \leftarrow t_2^{\Delta k_{\mathcal{S}}}, t'_3 \leftarrow t_3^{\Delta k_{\mathcal{S}}}$		

Master Key Rotation protocol

# Sample a new master key in Server.
$k_{\mathcal{S}} \leftarrow H_3(\mathsf{mk}_{\mathcal{S}}, w)$
$\mathsf{mk}'_{\mathcal{S}} \leftarrow\$ \mathbb{Z}_q, k'_{\mathcal{S}} \leftarrow H_3(\mathsf{mk}'_{\mathcal{S}}, w)$
$\Delta k_{\mathcal{S}} \leftarrow k'_{\mathcal{S}}/k_{\mathcal{S}}$
# Update t_2 and t_3 with $\Delta k_{\mathcal{S}}$ in Client.

Fig. 7: The key Rotation protocol of PW-Hero.



Decryption Protocol

Client$(w, \mathsf{pk}_{\mathcal{S}}, k_{\mathcal{C}})$		**Server**$(\mathsf{mk}_{\mathcal{S}})$
	$\xrightarrow{w}$	
	$\xleftarrow{k_{\mathcal{S}}}$	$k_{\mathcal{S}} \leftarrow H_3(\mathsf{mk}_{\mathcal{S}}, w)$
if $g^{1/k'_{\mathcal{S}}} \mathsf{pk}_{\mathcal{S}} = g$ **then**		
for each (h_0, s, t_1, t_2, t_3)		
if $H_2(t_2^{1/k'_{\mathcal{S}}}/h_0)^{k_{\mathcal{S}}'} = t_3$ **then**		
$h_1 \leftarrow t_1/(t_2^{1/k'_{\mathcal{S}}}/h_0), t \leftarrow h_1^{k_{\mathcal{C}}}$		
record new $T = \{h_0, s, t\}$		

- Proof of verifiability

$\because H_2(t_2^{1/k'_{\mathcal{S}}}/h_0)^{k'_{\mathcal{S}}} = t_3$
$\Rightarrow H_2(t_2^{1/k_{\mathcal{S}}}/h_0)^{k'_{\mathcal{S}}} = H_2(r)^{k'_{\mathcal{S}}}$
$\Rightarrow t_2^{1/k_{\mathcal{S}}}/h_0 = r$
$\therefore h_1 \leftarrow t_1/r \leftarrow t_1/(t_2^{1/k_{\mathcal{S}}}/h_0)$

- Proof of correctness

$h'_1 \leftarrow t_1/(t_2^{1/k_{\mathcal{S}}}/h_0)$
$\Rightarrow h'_1 \leftarrow t_1/r$
$\Rightarrow h'_1 \leftarrow h_1$
Therefore the decryption result must be the correct h_1.

Fig. 8: The decryption protocol of PW-Hero.

π_y. After receiving the returns y and π_y, the client verifies π_y and then unblinds the return y to obtain $t'_2 \leftarrow y^{1/nc}$. If $t_2 = t'_2$, the $(\mathsf{un}, \mathsf{pw}')$ passes verification, otherwise it fails.

CORRECTNESS. In the registration phase, the client stored the password verification record tuple $T \leftarrow (h_0, s, t_1, t_2, t_3)$ for the username-and-password pair $(\mathsf{un}, \mathsf{pw})$. During the verification phase, the client verifies the equation $t_2 = t'_2$ to ensure that the login password pw' is identical with the underlying password pw of T.

$$t_2 = (r \cdot h_0)^{k_{\mathcal{S}}}, \tag{1}$$

$$\begin{aligned} t'_2 &= y^{1/nc} = (x_1 \cdot x_2)^{k_{\mathcal{S}}/nc} = ((t_1/h'_1)^{nc} h_0^{nc})^{k_{\mathcal{S}}/nc} \\ &= ((t_1/h'_1)h_0)^{k_{\mathcal{S}}} = (r \cdot h_0 \cdot h_1/h'_1)^{k_{\mathcal{S}}}. \end{aligned} \tag{2}$$

If the equation $t_2 = t'_2$ holds, i.e., Eq. (1)=Eq. (2), it means that $(r \cdot h_0 \cdot h_1/h'_1)^{k_{\mathcal{S}}}$ is equal to $(r \cdot h_0)^{k_{\mathcal{S}}}$, that is, $h_1 = h'_1$. h_1 and h'_1 are the hashes of the registration password pw and the verification password pw' respectively, with the same salt value s and username un, so that pw=pw' can be obtained from $h_1 = h'_1$. In other words, the equation holds $t_2 = t'_2$ if and only if pw=pw'.

4) Key Rotation: Fig. 7 shows the secret key rotation protocol $\Phi.\langle \mathcal{C}(w, w', \mathsf{pk}_{\mathcal{S}}), \mathcal{S}(\mathsf{mk}_{\mathcal{S}}) \rangle_{\text{skrot}}$ and the master key rotation protocol $\Phi.\langle \mathcal{C}(\mathsf{pk}_{\mathcal{S}}), \mathcal{S}(\mathsf{mk}_{\mathcal{S}}) \rangle_{\text{mkrot}}$. In the former protocol, the client sends old and new state identifiers (w, w') to request a key rotation. The server calculates old secret $k_{\mathcal{S}}$ of w, generates new key $k'_{\mathcal{S}}$ with $\mathsf{SKeyGen}$, and computes the update token $\triangle k_{\mathcal{S}} \leftarrow k'_{\mathcal{S}}/k_{\mathcal{S}}$. Then, the new public key $\mathsf{pk}'_{\mathcal{S}}$ and the update token $\triangle k_{\mathcal{S}}$ are sent to the client. Then the client updates $t'_2 \leftarrow t_2^{\triangle k_{\mathcal{S}}}$ and $t'_3 \leftarrow t_3^{\triangle k_{\mathcal{S}}}$. As for the latter protocol, the server first samples a new master key and generates the update tokens for all clients. Then all clients update t_2 and t_3.

CORRECTNESS. In the key rotation protocols, the key $k_{\mathcal{S}}$ is rotated to $k'_{\mathcal{S}}$. And the client needs to update the password record t_2 to t'_2. If the key rotation protocols are correct, t'_2 should be the password record hardened with the new key.

$$t_2 = (r \cdot h_0)^{k_{\mathcal{S}}} \tag{3}$$

$$t'_2 = t_2^{\triangle k_{\mathcal{S}}} = ((r \cdot h_0)^{k_{\mathcal{S}}})^{k'_{\mathcal{S}}/k_{\mathcal{S}}} = (r \cdot h_0)^{k'_{\mathcal{S}}} \tag{4}$$

As shown in Eq. (3) and Eq. (4), only the key $k_{\mathcal{S}}$ used for hardening t_2 is updated to $k'_{\mathcal{S}}$, so the updated t'_2 is correct.

In addition, the client key $k_{\mathcal{C}}$ can also be rotated to the new one $k'_{\mathcal{C}}$. To be specific, the client computes $\triangle k_{\mathcal{C}} \leftarrow k'_{\mathcal{C}}/k_{\mathcal{C}}$ and send it to the server. Then, the client and server both update h_0 by $h'_0 \leftarrow h_0^{\triangle k_{\mathcal{S}}}$.

5) Decryption: Fig. 8 shows the decryption protocol $\Phi.\langle \mathcal{C}(\mathsf{pk}_{\mathcal{S}}, k_{\mathcal{C}}), \mathcal{S}(\mathsf{mk}_{\mathcal{S}}) \rangle_{\text{dec}}$. The server only needs to send $k_{\mathcal{S}}$ to the client. Then, if the return value passes the check $g^{1/k_{\mathcal{S}}} \mathsf{pk}_{\mathcal{S}} = g$, the client opens verification information by three steps: Firstly, the client recovers r' by $r' \leftarrow t_2^{1/k_{\mathcal{S}}}/h_0$; Secondly, it checks $H_2(r')^{k_{\mathcal{S}}} = t_3$ and ensures $r' = r$; Thirdly, it recovers h'_1 by $h'_1 \leftarrow t_1/r'$. Finally, the client can get a traditional verification information (h_0, s, h'_1).

CORRECTNESS. When an authenticated client wants to terminate the use of the PW-Hero service,

The client recovers the password salted hash h_1 by requesting the decryption protocol. In this process, the client first checks the correctness of the returned key $k_{\mathcal{S}}'$ through Eq. (5).

$$g^{1/k_{\mathcal{S}}'} \cdot \mathsf{pk}_{\mathcal{S}} = g. \tag{5}$$

If $k_{\mathcal{S}}' = k_{\mathcal{S}}$, the client goes on check whether recovered r' is correct by whether $H_2(r')^{k_{\mathcal{S}}'} = t_3$ holds, where $r' \leftarrow t_2^{1/k_{\mathcal{S}}}/h_0$ and $t_3 \leftarrow H_2(r)^{k_{\mathcal{S}}}$. If r' is correct (i.e., $r' = r$), the client can determine that h_1' derived by $h_1' \leftarrow t_1/r'$ is correct (i.e., $h_1' = h_1$). In other words, the decryption protocol correctly returns the salted hash value of pw.

B. Security Analysis

In this section, we analyze the security of the PW-Hero scheme and show that our PW-Hero scheme satisfies: obliviousness, hiding, forward security, binding, and privacy. The formal security definitions and proofs are in Appendix B and C. We follow previous works [9], [10] and provide security assumptions: The decisional Diffie-Hellman (DDH) problem and the discrete logarithm (DL) problem are hard in group $\mathbb{G}$, and $\{H_{i,i\in\{0,1,2,3\}}\}$ are modeled as random oracles. We assume that an adversary will not compromise both client and server simultaneously; otherwise any PH service is insecure. We also assume that the client of the decryption protocol is authenticated and cannot be malicious.

1) Obliviousness: It means that a malicious/compromised server cannot learn passwords from the protocol interactions with the client. We prove the obliviousness by ensuring that the server cannot distinguish whether the password participating in PW-Hero protocols is pw_0 and pw_1. Of the intermediate values available to servers, only x_1 depends on pw, $x_1 = (r \cdot H_1(s, \mathsf{un}, \mathsf{pw}_b)/H_1(s, \mathsf{un}, \mathsf{pw}))^{n_{\mathcal{C}}}$, where $n_{\mathcal{C}}$ is a nonce sampled by the client. The malicious server needs to distinguish between $(r \cdot H_1(s, \mathsf{un}, \mathsf{pw}_b)/H_1(s, \mathsf{un}, \mathsf{pw}_0))^{n_{\mathcal{C}}}$ and $(r \cdot H_1(s, \mathsf{un}, \mathsf{pw}_b)/H_1(s, \mathsf{un}, \mathsf{pw}_1))^{n_{\mathcal{C}}}$, which is equivalent to distinguishing between the tuple $(g, g^{\alpha}, g^{\beta}, g^{\gamma}, g^{\alpha\gamma})$ and the tuple $(g, g^{\alpha}, g^{\beta}, g^{\gamma}, g^{\beta\gamma})$, where $r = g^{\alpha}$, $n_{\mathcal{C}} = \gamma$, $(r)^{n_{\mathcal{C}}} = g^{\alpha\gamma}$, and $(r \cdot H_1(s, \mathsf{un}, \mathsf{pw}_b)/H_1(s, \mathsf{un}, \mathsf{pw}_{b-1}))^{n_{\mathcal{C}}} = g^{\beta\gamma}$. The latter is the DDH-problem. Since the DDH problem is hard in $\mathbb{G}$, we conclude that the server guessing whether $\mathsf{pw} = \mathsf{pw}_0$ or $\mathsf{pw} = \mathsf{pw}_1$ cannot perform better than tossing a coin.

2) Hiding: It means that a compromised client cannot learn passwords from the password verification record $T := \{h_0, s, t_1, t_2, t_3\}$ by offline attacking. First, cracking the external key $k_{\mathcal{S}}$ from $t_2 = (r \cdot h_0)^{k_{\mathcal{S}}}$ and $t_3 = H_2(r)^{k_{\mathcal{S}}}$ is equivalent to solving the DL-problem, where , which is hard in group $\mathbb{G}$. Second, only the $t_1 = r \cdot H_1(s, \mathsf{un}, \mathsf{pw})$ is dependent on the pw. But, r, a random value securely deleted by the honest registration-phase client, covers the pw so that later compromised client attackers cannot see it. Thus, malicious clients cannot learn pw from T. Third, the attacker cannot distinguish the records of pw_1 and pw_2, which can be simplified to distinguish between $H_1(s, \mathsf{un}, \mathsf{pw}_1)^{k_{\mathcal{S}}}$ and $H_1(s, \mathsf{un}, \mathsf{pw}_2)^{k_{\mathcal{S}}}$. Given a tuple $(g, g^{\alpha}, g^{\beta}, g^{k_{\mathcal{S}}}, g^{\alpha k_{\mathcal{S}}}, g^{\beta k_{\mathcal{S}}})$, let $g^{\alpha} = H_1(s, \mathsf{un}, \mathsf{pw}_1)$, $g^{\beta} = H_1(s, \mathsf{un}, \mathsf{pw}_2)$, $g^{\alpha k_{\mathcal{S}}} = H_1(s, \mathsf{un}, \mathsf{pw}_1)^{k_{\mathcal{S}}}$, and $g^{\beta k_{\mathcal{S}}} = H_1(s, \mathsf{un}, \mathsf{pw}_2)^{k_{\mathcal{S}}}$. The problem is equivalent to distinguishing between the tuple $(g, g^{\alpha}, g^{\beta}, g^{k_{\mathcal{S}}}, g^{\alpha k_{\mathcal{S}}})$ and the tuple $(g, g^{\alpha}, g^{\beta}, g^{k_{\mathcal{S}}}, g^{\beta k_{\mathcal{S}}})$, the DDH-problem, which is hard in $\mathbb{G}$.

3) Forward Security: It means that a key rotation can make the old secret key useless for the new password verification records and the new key useless for the old password verification records. We prove the forward security by ensuring that the verification record updated by the key rotation protocols is indistinguishable from the verification record generated by starting over with the registration protocol. Let old record $t_2 = (r \cdot h_0)^{k_{\mathcal{S}}}$. After a secret key rotation, let the new secret key be $k_{\mathcal{S}}'$ such that $t_2' = ((r \cdot h_0)^{k_{\mathcal{S}}})^{k_{\mathcal{S}}'/k_{\mathcal{S}}}$. Solving the DL problem $t_2' = (r \cdot h_0)^{k_{\mathcal{S}}'}$ can be reduced to solving $t_2' = (r' \cdot h_0')^{k_{\mathcal{S}}}$, where $r' = r^{k_{\mathcal{S}}'/k_{\mathcal{S}}}$ and $h_0' = h_0^{k_{\mathcal{S}}'/k_{\mathcal{S}}}$. Clearly, the tuple $(r, h_0, k_{\mathcal{S}}, (r \cdot h_0)^{k_{\mathcal{S}}})$ and the tuple $(r', h_0', k_{\mathcal{S}}, (r' \cdot h_0')^{k_{\mathcal{S}}})$ are indistinguishable.

4) Binding: On the one hand, it means that the malicious/compromised server cannot trick the client into generating wrong verification results. We reduce this problem to the fact that a client will not pass two passwords for one verification record. We assume that there are two different underlying passwords pw_0 and pw_1 for T. In that case, there are $(t_1/H_1(s, \mathsf{un}, \mathsf{pw}_0)) \cdot h_0)^{\tilde{k_{\mathcal{S}}}} = (r \cdot h_0)^{k_{\mathcal{S}}}$ and $(t_1/H_1(s, \mathsf{un}, \mathsf{pw}_1)) \cdot h_0)^{\tilde{k_{\mathcal{S}}}} = (r \cdot h_0)^{k_{\mathcal{S}}}$ that hold, where $\tilde{k_{\mathcal{S}}}$ is set by the malicious server in the verification protocol. Due to the values returned from the server are together with their ZKP, $\tilde{k_{\mathcal{S}}} = k_{\mathcal{S}}$ must holds. Other related parameters (r, s, and un) are isolated with the server. Therefore, $\mathsf{pw}_0 = \mathsf{pw}_1$ is the necessary condition for the assumption that the two equations hold, but this contradicts the premise of the assumption ($\mathsf{pw}_0 \neq \mathsf{pw}_1$), so the pre-assumption does not keep.

On the other hand, binding also means that the malicious/compromised client cannot tamper with the username identifier h_0 to bypass the rate limit on verification requests in the server. We reduce the problem to the fact that the malicious client cannot verify the username-and-password pair $(\mathsf{un}_1, \mathsf{pw})$ through the verification interaction for the pair $(\mathsf{un}_0, \mathsf{pw})$, where $\mathsf{un}_0 \neq \mathsf{un}_1$. Due to the ZKP of x_2, $x_2 = h_0^{n_{\mathcal{C}}}$ must hold, where $n_{\mathcal{C}}$ is a random value sampled by the client. For the verification request $h_0 = H_0(\mathsf{un}_0)^{k_{\mathcal{C}}}$, the client receives $y = (t_1/H_1(s, \mathsf{un}_1, \mathsf{pw}) \cdot h_0)^{n_{\mathcal{C}} k_{\mathcal{S}}}$. Then the malicious client needs to solve for $y' = (t_1/H_1(s, \mathsf{un}_1, \mathsf{pw}) \cdot H_0(\mathsf{un}_1))^{n_{\mathcal{C}} k_{\mathcal{S}}}$, which is equivalent to solving for the secret key $k_{\mathcal{S}}$. The latter is a DL problem, which is hard in group $\mathbb{G}$.

5) Privacy: It means that a malicious/compromised server can learn nothing about the password verification results ("Accept" or "Reject"). Recall that the verification formula for the client to get the verification result is $((x_1 \cdot x_2)^{k_{\mathcal{S}}})^{1/n_{\mathcal{C}}} = (r \cdot h_0)^{k_{\mathcal{S}}}$. The server can obtain intermediate values $(r \cdot h_0)^{k_{\mathcal{S}}}$ from the registration protocol and $(x_1 \cdot x_2)^{k_{\mathcal{S}}}$ from the verification protocol, but not $1/n_{\mathcal{C}}$, where $n_{\mathcal{C}}$ is a random value isolated from the server. We conclude that the malicious sever cannot learn verification results from the PW-Hero protocols.

Furthermore, even if the verification record is specified to be that of pw_0, it is difficult for a malicious server to distinguish whether the correct password pw_0 or the wrong password pw_1 is participating in the verification protocol. Given a tuple $(g, g^{\alpha}, g^{\beta}, g^{n_{\mathcal{C}}}, g^{\alpha n_{\mathcal{C}}}, g^{\beta n_{\mathcal{C}}})$, where $g^{\alpha} = H_1(s, \mathsf{un}, \mathsf{pw}_0)$ and $g^{\beta} = H_1(s, \mathsf{un}, \mathsf{pw}_1)$. The problem is equivalent to distinguishing between the tuple $(g, g^{\alpha}, g^{\beta}, g^{n_{\mathcal{C}}}, g^{\alpha n_{\mathcal{C}}})$ and the tuple $(g, g^{\alpha}, g^{\beta}, g^{n_{\mathcal{C}}}, g^{\beta n_{\mathcal{C}}})$, which is a DDH problem, hard in $\mathbb{G}$.

Additionally, privacy means that usernames are anonymous in the PW-Hero protocols. $h_0 = H_0(\mathsf{un})^{k_{\mathcal{C}}}$, where $k_{\mathcal{C}}$ is a client key. Even assuming that the username is low-entropy, solving this problem is at least equivalent to solving a DL problem, which is hard in group $\mathbb{G}$. Thus, the malicious server cannot learn the username un from h_0.

C. Opt-out and Accessing to New PH Services

PW-Hero permits its clients to return to a traditional storage form of salted hashes, i.e., $T := \{h_0, s, h_1\}$, after opting out. We use Phoenix [10] as an example of a new service to illustrate the process of a client opting out of the PW-Hero service and accessing a new service. The client provides identity proof, such as secure email and device token, to the PW-Hero service and then makes an exit request. The service verifies the client's identity and then executes a decryption protocol. After that, the client obtains the new password record $T := \{h_0, s, h_1\}$. When joining the new service Phoenix, s is used as $n_{\mathcal{C}}$, and h_1 is used as $H_{\mathcal{C}}(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}})$. The service switching process for Pythia [8] or PO-COM [9] as a new service is also similar.

There is an optional suggestion about the offline status for the client in service switching. Suppose the client exits the old server and joins a new service immediately. In this case, it is recommended that the client remains offline until all password records are migrated to the new service for protection, to avoid accidental corruption and disclosure of offline-crackable salted hash records during this time.

V. Evaluation

We implemented PW-Hero protocols' server and client cryptographic algorithms based on the Charm-Crypto [30] cryptographic framework and using NIST P-256 as the multiplicative group. We implemented a prototype application of the server based on the Falcon Python Web framework [31]. The communication was implemented with the Python httplib2 library. Data was passed to the server as GET request parameters and returned to the client as a Json string. And we set up NGINX and uWSGI to run our server and provision its service.

We re-implemented the existing PH service schemes, such as PO-COM [9] and Phoenix [10], in the same way. A prototype of Pythia is provided at [11], [32] using BN 254 as the base group. The execution time of group operations of NIST P 256 and BN 254 is given in Appendix D.

Our experiments run on a machine equipped with Intel(R) Core(TM) i7-8850H/2.60GHz ×2 and 3.8 GiB RAM, installed with 64-bit Ubuntu 20.04.3 LTS. Our experiments were conducted to test the client latency of the registration phase and verification phases. All experiment results presented are the averages of 10,000 independent executions.

TABLE I: Client latency (in ms).

Keep-alive HTTP			✓	✓	✓
Zero-knowledge proof		✓		✓	✓
Verification of incorrect password					✓
Phoenix [10]	Enroll	1.134	1.434		
	Validate	2.508	2.341	2.402	1.365
PO-COM [9]	Enroll	1.424	1.181	1.433	
	Validate	1.876		1.887	2.035
Pythia [8]	Eval	8.696	3.583	8.533	
PW-Hero	Reg	1.362	0.945	1.371	
	Ver	1.978	1.118	1.917	1.907

✓ denotes that the experimental group provides the corresponding optional test parameter. A cell with no data indicates that the protocol cannot provide this combination of settings. Enroll and Validate represent the registration and verification protocols in Phoenix and PO-COM, respectively. Pythia executes Eval during both the registration and validation phases. The standard deviations of the above test results are very small (< 0.15 ms).

TABLE II: Client latency (in ms) over HTTPS.

Scheme	Latency of Registration	Latency of Verification
Phoenix [10]	5.191	6.667
PO-COM [9]	6.093	7.205
PW-Hero ⋆	5.632	6.900

The standard deviations of the above test results are less than 1.5 ms.

1) Latency: Client-side latency is the length of time the client takes from the start of the protocol to the end of the protocol, including the time spent by the client and server performing calculations based on the protocol. The tested client-side latency does not include the portion of time spent retrieving the database and rate-limiting, nor the RTT. In other words, we test the latency of pure protocol calculations. We provided three optional test parameters: Whether HTTP-keepalive is turned on, whether zero-knowledge proofs are included, and whether the login passwords are correct, and tested all the protocols in four combinations of setting parameters.

Tab. I shows the latency of one registration and one verification for three existing schemes and our PW-Hero. Pythia [8] has the worst performance, whose latency is about five times that of the other three schemes (i.e., PW-Hero, Phoenix [10], and PO-COM [9]) with similar performance. Among them, PW-Hero slightly outperforms the other two (i.e., Phoenix and PO-COM). Phoenix has the shortest verification time for incorrect passwords. The reason is that the Phoenix server is informed of the password verification result in advance during the verification phase and omits part of the computation when the login password is incorrect. Another point to add is that the part of our registration protocol that can be parallelized is not reflected in our simple implementation of the PW-Hero prototype, so the registration latency of PW-Hero should be shorter in the actual case. On balance, the latency comparison of protocol calculations shows that PW-Hero is feasible and does not impose a significant computational and time burden on password registration and verification.

In addition, we further tested the client latency of the three well-performing schemes (i.e., PW-Hero, Phoenix [10], and PO-COM [9]) over HTTPS with an OpenSSL self-signed certificate. As shown in Tab. II, the tested client latency of these three schemes over HTTPS is almost the same. Our PW-Hero is inferior to Phoenix with a performance disadvantage of

TABLE III: Evaluation of our scheme ($\star$) with comparison among relevant password hardening schemes.

Scheme			Latency		Storage	Property															
			Registration	Verification		P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16
(t,m)-PHE	(2020)	[13]	$6H+19E$	$6H+96E$	$3L_\lambda+2L_g$	✓	✓	✓	✓	✓	✓	✓	✓	×	×	✓	✓	✓	✓	✓	×
PHE	(2018)	[12]	$6H+13E$	$6H+15E$	$2L_\lambda+2L_g$	✓	✓	✓	✓	✓	✓	×	✓	×	×	✓	✓	✓	✓	✓	✓
Phoenix	(2017)	[10]	$2H+5E$	$3H+16E$	$3L_\lambda+3L_g$	✓	✓	✓	✓	✓	×	×	(✓)	×	×	✓	✓	✓	✓	✓	✓
PO-COM	(2016)	[9]	$H+12E$	$H+19E$	$L_\lambda+3L_g$	✓	✓	✓	✓	✓	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓
Pythia	(2015)	[8]	$H_{G_2}+E_{G_2}+6E_{G_t}+\text{pair}$		$L_\lambda+L_{g_t}$	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	×
PW-Hero		$\star$	$3H+15E$	$2H+18E$	$2L_\lambda+3L_g$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

✓	: Achieving the corresponding goal.	L_λ	: Length of the element in $\{0,1\}^\lambda$.	P1: Password privacy.	P7: Verification-result privacy.
×	: Not achieving the corresponding goal.	L_g	: Length of the element in G.	P2: External secret privacy.	P8: Verifiability.
(✓)	: Partially achieving the corresponding goal.	L_{g_t}	: Length of the element in Gt.	P3: Rate-limit verification.	P9: Opt-out.
H	: Hashes inputs into G.	E	: Exponentiates the element in G.	P4: Username binding.	P10: Individual key rotation.
pair	: Pairs elements in G_1 and G_2 into G_T.	E_{G_2}	: Exponentiates the element in $G2$.	P5: Password binding.	P11: Master key rotation.
H_{G_2}	: Hashes inputs into G_2.	H_{G_t}	: Hashes inputs into G_t.	P6: Username anonymity.	P12: Secure key rotation.

P13: Compact traffic.
P14: Batch update.
P15: User transparency.
P16: Low latency.
Set $t = m = 1$.

no more than 0.5 ms, and is better than PO-COM with a performance advantage of no more than 0.5 ms.

2) Storage Analysis: Inevitably, the client of PH service needs to store some verification information for each registered username-and-password pair. In our PW-Hero scheme, a salt value, two verification-related values, and a decryption-related value need to be stored in the client. Our server stores a username-driven index for each account and a counter for rate-limiting as well as other existing schemes. Tab. III shows the comparison of the storage size of clients of four schemes, where PW-Hero performs in the middle.

Discussion. As shown in Tab. III, our PW-Hero is the best overall, considering efficiency, storage, and properties compared to the existing schemes [8]–[10]. PW-Hero achieves the most comprehensive properties, and its high efficiency is attributed to high-performance algorithms based on the multiplicative group. In addition, PO-COM [9] and Phoenix [10] perform as well. In contrast, the bilinear pairing-based Pythia [8] performs less well due to the time-consuming pairing operation. Moreover, the PHE scheme [12] is an encryption scheme based on Phoenix. Its threshold scheme (t, m)-PHE [13] is the only one that requires multiple rounds of communication; its protocols are complex, so its performance is relatively the worst. It should be noted that there are unexpected situations where the client wants to exit the PH service, but PW-Hero cannot support opt-out, such as the service operator going bankrupt and the server being malicious or hacked. In these cases, the server may be unwilling or unable to handle aftermath, i.e., the decryption protocol. Hopefully, this issue might be addressed by designing the PW-Hero threshold scheme [13], [27], [28] or the stand-alone decryption protocol [33].

VI. Conclusion

This paper proposes a new password hardening (PH) service with the opt-out property, named PW-Hero. PW-Hero assists its clients in strengthening stored password verification records to resist offline password guessing attacks. Additionally, PW-Hero increases the resistance to online password guessing attacks by rate-limiting verification requests. In contrast to previous works, the PW-Hero client can actively opt-out and return its password record to a state where it can easily join a new service, thus alleviating the concern of a failed or compromised PH service under one-shot deals of previous schemes. We demonstrate the practicality of our solution by building a PW-Hero instance and measuring the latency time of the client. We leave the question of building a more reliable and efficient PH service (e.g., restoring original password authentication when the PH service somehow does not work due to a hack or bankruptcy) as important future work.

Acknowledgment

The authors are grateful to the anonymous reviewers for their invaluable comments. Ding Wang is the corresponding author. This research was partly supported by the National Natural Science Foundation of China under Grant No.62172240 and by the Natural Science Foundation of Tianjin, China under Grant No.21JCZDJC00190. There are no competing interests.

References

[1] P. A. Grassi, J. L. Fenton, E. M. Newton, R. A. Perlner, A. R. Regenscheid, W. E. Burr, J. P. Richer *et al.*, "NIST 800-63B digital identity guidelines: Authentication and lifecycle management," National Institute of Standards and Technology, McLean, VA, Tech. Rep., 2017, doi:https://doi.org/10.6028/NIST.SP.800-63b.

[2] UK National Cyber Security Centre, *Password policy: Updating your approach*, Nov. 2018, https://www.ncsc.gov.uk/collection/passwords/updating-your-approach.

[3] D. Goodin, *Every Yahoo account that existed all 3 billion was compromised in 2013 hack*, Oct. 2017, https://arstechnica.com/information-technology/2017/10/yahoo-says-all-3-billion-accounts-were-compromised-in-2013-hack/.

[4] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian, "Zipf's law in passwords," *IEEE Trans. Inform. Foren. Secur.*, vol. 12, no. 11, pp. 2776–2791, 2017.

[5] D. Goodin, *Once seen as bulletproof, 11 million+ Ashley Madison passwords already cracked*, Sep. 2015, http://arstechnica.com/security/2015/09/once-seen-as-bulletproof-11-million-ashley-madison-passwords-already-cracked/.

[6] M. Spacek, *Cracking passwords from the Mall.cz dump*, Jan. 2018, https://www.michalspacek.com/cracking-passwords-from-the-mall.cz-dump.

[7] A. Muffett, *Facebook password hashing and authentication.*, https://www.youtube.com/watch?v=7dPRFoKteIU.

[8] A. Everspaugh, R. Chaterjee, S. Scott, A. Juels, and T. Ristenpart, "The Pythia PRF service," in *Proc. USENIX SEC 2015*, pp. 547–562.

[9] J. Schneider, N. Fleischhacker, D. Schröder, and M. Backes, "Efficient cryptographic password hardening services from partially oblivious commitments," in *Proc. CCS 2016*, pp. 1192–1203.

[10] R. W. Lai, C. Egger, D. Schröder, and S. S. Chow, "Phoenix: Rebirth of a cryptographic password-hardening service," in *Proc. USENIX SEC 2017*, pp. 899–916.

[11] Ace0, "Pythia server (prototype) implementation," 2015, https://github.com/ace0/pythia.
[12] R. W. Lai, C. Egger, M. Reinert, S. S. Chow, M. Maffei, and D. Schröder, "Simple password-hardened encryption services," in *Proc. USENIX SEC 2018*, pp. 1405–1421.
[13] J. Brost, C. Egger, R. W. Lai, F. Schmid, D. Schröder, and M. Zoppelt, "Threshold password-hardened encryption services," in *Proc. CCS 2020*, pp. 409–424.
[14] R. W. Lai, C. Egger, M. Reinert, S. S. Chow, M. Maffei, and D. Schröder, "Slides of simple password-hardened encryption services." https://www.usenix.org/sites/default/files/conference/protected-files/security18_slides_lai.pdf.
[15] C. Diomedous and E. Athanasopoulos, "Practical password hardening based on tls," in *Proc. DIMVA 2019*, pp. 441–460.
[16] B. Hitaj, P. Gasti, G. Ateniese, and F. Perez-Cruz, "Passgan: A deep learning approach for password guessing," in *Proc. ACNS 2019*, pp. 217–237.
[17] D. Wang, D. He, H. Cheng, and P. Wang, "fuzzypsm: A new password strength meter using fuzzy probabilistic context-free grammars," in *Proc. IEEE/IFIP DSN 2016*, pp. 595–606.
[18] AgileBits Inc, "1password password manager: Generate and keep passwords safe," 2022, https://1password.com.
[19] S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena, "Two-factor authentication with end-to-end password security," in *Proc. PKC 2018*, pp. 431–461.
[20] D. Wang and P. Wang, "Two birds with one stone: Two-factor authentication with security beyond conventional bound," *IEEE Trans. Depend. Secur. Comput.*, vol. 15, no. 4, pp. 708–722, 2018.
[21] S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena, "Device-enhanced password protocols with optimal online-offline protection," in *Proc. ASIACCS 2016*, pp. 177–188.
[22] M. Shirvanian, S. Jareckiy, H. Krawczykz, and N. Saxena, "Sphinx: A password store that perfectly hides passwords from itself," in *Proc. ICDCS 2017*, pp. 1094–1104.
[23] D. Wang, Y. Zou, Q. Dong, Y. Song, and X. Huang, "How to attack and generate honeywords," in *Proc. IEEE S&P 2022*, pp. 489–506.
[24] M. H. Almeshekah, C. N. Gutierrez, M. J. Atallah, and E. H. Spafford, "Ersatzpasswords: Ending password cracking and detecting password leakage," in *Proc. ACSAC 2015*, pp. 311–320.
[25] C. N. Gutierrez, M. H. Almeshekah, S. Bagchi, and E. H. Spafford, "A hypergame analysis for ersatzpasswords," in *Proc. IFIP SEC 2018*, pp. 47–61.
[26] J. Kelsey, D. Dachman-Soled, S. Mishra, and M. S. Turan, "Tmps: ticket-mediated password strengthening," in *Proc. CT-RSA 2020*, pp. 225–253.
[27] Y. Zhang, C. Xu, H. Li, K. Yang, N. Cheng, and X. Shen, "Protect: efficient password-based threshold single-sign-on authentication for mobile users against perpetual leakage," *IEEE Trans. Mobile Comput.*, vol. 20, no. 6, pp. 2297–2312, 2020.
[28] S. Agrawal, P. Miao, P. Mohassel, and P. Mukherjee, "Pasta: password-based threshold authentication," in *Proc. CCS 2018*, pp. 2042–2059.
[29] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu, "Password-protected secret sharing," in *Proc. CCS 2011*, pp. 433–444.
[30] Johns Hopkins University ISI, "Charm-crypto docs," https://jhuisi.github.io/charm/index.html.
[31] K. Griffiths, "The falcon web framework," https://falcon.readthedocs.io/en/stable/.
[32] Ace0, "Safeid protects passwords using the pythia prf protocol," 2015, https://github.com/ace0/safeid.
[33] X. Huang, Y. Xiang, E. Bertino, J. Zhou, and L. Xu, "Robust multi-factor authentication for fragile communications," *IEEE Trans. Depend. Secur. Comput.*, vol. 11, no. 6, pp. 568–581, 2014.
[34] Ace0, "Pyrelic: a python module that wraps the relic cryptography library," 2016, https://github.com/ace0/pyrelic.

Appendix A
Zero-Knowledge Proof

Fig. 9 shows the zero-knowledge proof (ZKP) protocol adopted by our PW-Hero instantiation, which is the same as that of Pythia [8]. The ZKP protocol is defined as $\Pi := \{\mathsf{Gen}, \mathsf{PoK}, \mathsf{Vf}\}$, which proves $y = x^k$ without delivering k.

```
Gen(1^λ)
    $H \leftarrow\!\$\ \mathcal{H} = \{\{0,1\}^* \to \mathbb{Z}_q\}$
    return $H$

Vf($g, g^k, x, y, \pi$)
    $(h, u) \leftarrow \pi$
    $a_1' \leftarrow g^u \cdot (g^k)^h,\ a_2' \leftarrow x^u \cdot y^h$
    $h' \leftarrow H(g, g^k, x, v, a_1', a_2')$
    return $h = h'$

PoK($g, g^k, x, y, k$)
    $v \leftarrow\!\$\ \mathbb{Z}_q$
    $a_1 \leftarrow g^v,\ a_2 \leftarrow x^v$
    $h \leftarrow H(g, g^k, x, v, a_1, a_2)$
    $u \leftarrow v - h \cdot k$
    return $\pi \leftarrow (h, u)$
```

Fig. 9: The ZKP protocol adopted by the PW-Hero scheme, $\Pi := \{\mathsf{Gen}, \mathsf{PoK}, \mathsf{Vf}\}$ that proves $y = x^k$ without revealing k.

Correctness. When $h = h'$, the ZKP protocol claims that $y = x^k$ does hold. Assuming that H is a random oracle, $h = h'$ is equivalent to $a_1 = a_1'$ and $a_2 = a_2'$.

$$a_1 = g^v, \quad a_1' = g^u (g^k)^h = g^{v-hk}(g^k)^h = g^v,$$
$$a_2 = x^v, \quad a_2' = x^u y^h = x^{v-hk} y^h = x^v (x^{-k} y)^h.$$

$a_1 = a_1'$ means that the public g^k is the value of k-th exponential operation of g, and $a_2 = a_2'$ means $(x^{-k}y)^h = 1$, i.e., $y = x^k$. Thus, $y = x^k$ can be derived from $h = h'$.

Appendix B
Formal Security Definition

This section formally defines the securities of PW-Hero, including obliviousness, hiding, forward security, binding, and privacy, by considering their corresponding security games.

A. *Obliviousness*

We define the PH-OBL game, which is played between the challenger $\mathcal{R}$ acting as the client and the adversary $\mathcal{A}$ acting as the malicious server, as follows:

- Setup 1: On input of security parameter λ, $\Pi.Setup(1^\lambda)$ generates the public parameter pp. Set up the oracle accessed by $\mathcal{A}$: $\mathbb{O} \leftarrow \{\Pi.\langle \mathcal{A}(\cdot), \mathcal{A}(\cdot)\rangle_X, \Pi.\langle \mathcal{R}(\cdot), \mathcal{A}(\cdot)\rangle_X, X \in \{\mathsf{reg}, \mathsf{ver}, \mathsf{rot}, \mathsf{dec}\}\}$.
- Setup 2: Adversary $\mathcal{A}$ has free access to $\mathbb{O}$ and generates a public key $\mathsf{pk}_{\mathcal{S}}$, a username un and two passwords, pw_0 and pw_1, which are sent to the challenger $\mathcal{R}$.
- Challenge: Challenger $\mathcal{R}$ samples $b \leftarrow \{0,1\}$. With b, $\mathcal{R}$ performs the registration protocol $\langle \mathcal{R}(\mathsf{un}, \mathsf{pw}_b), \mathcal{A}(\cdot)\rangle_{\mathsf{reg}}$. $\mathcal{A}$ has free access to $\mathbb{O}$ again, including $\Pi.\langle \mathcal{R}(\mathsf{pw}_0), \mathcal{A}(\cdot)\rangle_{\mathsf{ver}}$ and $\Pi.\langle \mathcal{R}(\mathsf{pw}_1), \mathcal{A}(\cdot)\rangle_{\mathsf{ver}}$. [9]
- Output: Adversary $\mathcal{A}$ outputs his guess b' on b and wins the game if $b' = b$.

We refer to such adversary $\mathcal{A}$ as the PH-OBL adversary and define adversary $\mathcal{A}$'s advantage as

$$Adv_{\text{PH-OBL}}^{\mathcal{A},\mathbb{O}}(\lambda) = |Pr[b = b'] - \frac{1}{2}|.$$

[9] In [10], the Challenge returns $\perp$ for $\langle \mathcal{R}(\mathsf{un}, \mathsf{pw}_0), \mathcal{A}(\cdot)\rangle_{\mathsf{val}}$ and $\langle \mathcal{R}(\mathsf{un}, \mathsf{pw}_1), \mathcal{A}(\cdot)\rangle_{\mathsf{val}}$ to avoid the adversary $\mathcal{A}$ from directly learning pw from these two verification protocols. Its reason is that verification results are leaked to the server, i.e., $\mathcal{A}$, in Phoenix [10]. However, it results in the obliviousness definition not covering its verification protocol.

DEFINITION 2. (**Obliviousness**) We say that a PW-Hero service is PH-OBL secure if for any protocols in $\mathbb{O}$ and any probabilistic polynomial-time (PPT) PH-OBL adversary $\mathcal{A}$, the advantage $Adv^{\mathcal{A},\mathbb{O}}_{\text{PH-OBL}}(\lambda)$ is negligible.

B. Hiding

We define the PH-HID game, which is played between the two challengers, $\mathcal{R}_C$ acting as the first-phase honest client and $\mathcal{R}_S$ acting as the server, and the adversary $\mathcal{A}$ acting as the second-phase compromised client as follows:

- Setup 1: On input of the security parameter, $\Pi.Setup(1^\lambda)$ generates the public parameter pp. And $\Pi.SKeyGen()$ generates a master key $\mathsf{mk}_\mathcal{S}$, a secret key $\mathsf{sk}_\mathcal{S}$, and a public key $\mathsf{pk}_\mathcal{S}$. Then $\mathsf{mk}_\mathcal{S}$ is sent to the server challenger $\mathcal{R}_S$. $\mathsf{pk}_\mathcal{S}$ is sent to the adversary $\mathcal{A}$ and the client challenger $\mathcal{R}_C$. $\mathbb{O} \leftarrow \{\Pi.\langle\mathcal{A}(\cdot),\mathcal{A}(\cdot)\rangle_X, \Pi.\langle\mathcal{A}(\cdot),\mathcal{R}_S(\cdot)\rangle_X,$ $X \in \{\mathsf{reg},\mathsf{ver},\mathsf{rot},\mathsf{dec}\}\}$.
- Setup 2: Adversary $\mathcal{A}$ has free access to $\mathbb{O}$ and generates a username un and two passwords, pw_0 and pw_1, which are sent to the challenger $\mathcal{R}_C$.
- Challenge: Challenger $\mathcal{R}_C$ picks $b \leftarrow \{0,1\}$ at random. With b, $\mathcal{R}_C$ executes the protocol $\langle\mathcal{R}_C(\mathsf{un},\mathsf{pw}_b),\mathcal{R}_S(\cdot)\rangle_{\mathsf{reg}}$ and generates the verification record T^*, which is sent to the adversary $\mathcal{A}$. And $\mathcal{A}$ has free access to $\mathbb{O}$ again.
- Output: Adversary$\mathcal{A}$ outputs his guess b' on b and wins the game if $b' = b$.

We refer to such adversary $\mathcal{A}$ as a PH-HID adversary and define adversary $\mathcal{A}$'s advantage as

$$Adv^{\mathcal{A},\mathbb{O}}_{\text{PH-HID}}(\lambda) = |Pr[b = b'] - \frac{1}{2}|.$$

DEFINITION 3. (**Hiding**) We say that a PW-Hero service is PH-HID secure if for any protocols in $\mathbb{O}$ and any PPT PH-HID adversary $\mathcal{A}$, the advantage $Adv^{\mathcal{A},\mathbb{O}}_{\text{PH-HID}}(\lambda)$ is negligible.

C. Forward security

We define the PH-FOR-1 game, which is played between the two challengers, $\mathcal{R}_C$ acting as the first-phase honest client and $\mathcal{R}_S$ acting as the server, and the adversary $\mathcal{A}$ acting as the second-phase compromised client as follows:

- Setup 1: Same as the $\mathsf{Setup\ 1}$ in the PH-HID game.
- Setup 2: Adversary $\mathcal{A}$ has free access to $\mathbb{O}$ and generates two username-and-password pairs, $(\mathsf{un}_0,\mathsf{pw}_0)$ and $(\mathsf{un}_1,\mathsf{pw}_1)$, which are sent to the client challenger $\mathcal{R}_C$.
- Challenge: Challenger $\mathcal{R}_C$ picks randomly $b \leftarrow \{0,1\}$. With b, $\mathcal{R}_C$ performs registration protocol $\langle\mathcal{R}_C(\mathsf{un}_b,\mathsf{pw}_b),\mathcal{R}_S(\cdot)$ $\rangle_{\mathsf{reg}}$ and generates the verification record T^*, which is sent to adversary $\mathcal{A}$. Immediately, $\mathcal{R}_C$ performs secret key rotation protocol $\langle\mathcal{R}_C(\cdot),\mathcal{R}_S(\cdot)\rangle_{\mathsf{skrot}}$. $\mathcal{A}$ has free access to $\mathbb{O}$ again.
- Output: Adversary $\mathcal{A}$ outputs his guess b' on b and wins the game if $b' = b$.

We refer to such adversary $\mathcal{A}$ as the PH-FOR-1 adversary and define adversary $\mathcal{A}$'s advantage as

$$Adv^{\mathcal{A},\mathbb{O}}_{\text{PH-FOR-1}}(\lambda) = |Pr[b = b'] - \frac{1}{2}|.$$

Additionally, the PH-FOR-2 game is played between the two challengers, $\mathcal{R}_S$ acting as the second-phase honest server and $\mathcal{R}_C$ acting as the client, and the adversary $\mathcal{A}$ acting as the first-phase compromised server:

- Setup 1: On input of security parameter λ, $\Pi.Setup(1^\lambda)$ generates the public parameter pp. $\mathbb{O} \leftarrow \{\Pi.\langle\mathcal{A}(\cdot),\mathcal{A}(\cdot)\rangle_X,$ $\Pi.\langle\mathcal{R}_C(\cdot),\mathcal{A}(\cdot)\rangle_X, X \in \{\mathsf{reg},\mathsf{ver},\mathsf{rot},\mathsf{dec}\}\}$
- Setup 2: Adversary $\mathcal{A}$ has free access to $\mathbb{O}$ and generates a master key $\mathsf{mk}_\mathcal{S}$, a public key $\mathsf{pk}_\mathcal{S}$, and a username un and two passwords, pw_0 and pw_1, all but the first of which are sent to the client challenger $\mathcal{R}_C$. And $\mathsf{mk}_\mathcal{S}$ is sent to the server challenger $\mathcal{R}_S$.
- Challenge: $\mathcal{R}_C$ picks $b \leftarrow \{0,1\}$ at random. With b, $\mathcal{R}_C$ performs the registration protocol $\langle\mathcal{R}_C(\mathsf{un},\mathsf{pw}_b),\mathcal{A}(\cdot)\rangle_{\mathsf{reg}}$ and generates the password verification record T^*. Immediately, $\mathcal{R}_S$ performs master key rotation protocol $\langle\mathcal{R}_C(\cdot),\mathcal{R}_S(\cdot)\rangle_{\mathsf{mkrot}}$. Then, the old verification record T^* is sent to $\mathcal{A}$, who has free access to $\mathbb{O}$ again.
- Output: Adversary $\mathcal{A}$ outputs his guess b' on b and wins the game if $b' = b$.

We refer to such adversary $\mathcal{A}$ as the PH-FOR-2 adversary and define adversary $\mathcal{A}$'s advantage as

$$Adv^{\mathcal{A},\mathbb{O}}_{\text{PH-FOR-2}}(\lambda) = |Pr[b = b'] - \frac{1}{2}|.$$

DEFINITION 4. (**Forward security**) We say that a PW-Hero service is PH-FOR secure if for any protocols in $\mathbb{O}$ and any PPT PH-FOR-1 adversary $\mathcal{A}_1$ and PH-FOR-2 adversary $\mathcal{A}_2$, $\mathcal{A}_1$'s advantage $Adv^{\mathcal{A}_1,\mathbb{O}}_{\text{PH-FOR-1}}(\lambda)$ and $\mathcal{A}_2$'s advantage $Adv^{\mathcal{A}_2,\mathbb{O}}_{\text{PH-FOR-2}}(\lambda)$ both are negligible.

D. Binding

We define the PH-BIN-1 game, which is played between the challenger $\mathcal{R}$ acting as the client and the adversary $\mathcal{A}$ acting as the malicious server:

- Setup 1: Same as the $\mathsf{Setup\ 1}$ in the PH-OBL game.
- Setup 2: Adversary $\mathcal{A}$ has free access to $\mathbb{O}$ and generates a public key $\mathsf{pk}_\mathcal{S}$ and two different username-and-password pairs, $(\mathsf{un}_0,\mathsf{pw}_0)$ and $(\mathsf{un}_1,\mathsf{pw}_1)$, which are sent to $\mathcal{R}$.
- Challenge: Challenger $\mathcal{R}$ picks $b \leftarrow \{0,1\}$ at random. With b, $\mathcal{R}$ performs the registration protocol $\langle\mathcal{R}(\mathsf{un}_b,\mathsf{pw}_b,\mathsf{pk}_\mathcal{S}),$ $\mathcal{A}(\cdot)\rangle_{\mathsf{reg}}$ and generates the verification record T^*.
- Output: $\mathcal{R}$ performs two verification protocols with the $\mathcal{A}$, $\langle\mathcal{R}(\mathsf{un}_0,\mathsf{pw}_0,T^*,\mathrm{pk}_\mathcal{S}),\mathcal{A}(\cdot)\rangle_{\mathsf{ver}}$ and $\langle\mathcal{R}(\mathsf{un}_1,\mathsf{pw}_1,T^*,\mathrm{pk}_\mathcal{S}),$ $\mathcal{A}(\cdot)\rangle_{\mathsf{ver}}$, which return two verification results, r_0 and r_1. $b_0 := (r_0 = \text{“Accept”})$, $b_1 := (r_1 = \text{“Accept”})$. Adversary $\mathcal{A}$ wins the game if $b_0 \wedge b_1$.

We refer to such adversary $\mathcal{A}$ as a PH-BIN-1 adversary and define adversary $\mathcal{A}$'s advantage as

$$Adv^{\mathcal{A},\mathbb{O}}_{\text{PH-BIN-1}}(\lambda) = |Pr[b_0 \wedge b_1] - \frac{1}{2}|.$$

And the PH-BIN-2 game is played between the two challengers, $\mathcal{R}_S$ acting as the server and $\mathcal{R}_C$ acting as the first-phase honest client and the adversary $\mathcal{A}$ acting as the second-phase compromised client:

- Setup 1: Same as the $\mathsf{Setup\ 1}$ in the PH-HID game.
- Setup 2: Adversary $\mathcal{A}$ has free access to $\mathbb{O}$ and generates two different pairs of username and password, $(\mathsf{un}_0, \mathsf{pw}_0)$ and $(\mathsf{un}_1, \mathsf{pw}_1)$, which are sent to the client challenger $\mathcal{R}_C$.
- Challenge: $\mathcal{R}_C$ samples $b \leftarrow \{0,1\}$. With b, $\mathcal{R}_C$ performs the registration protocol $\langle \mathcal{R}_C(\mathsf{un}_b, \mathsf{pw}_b, \mathsf{pk}_\mathcal{S}), \mathcal{R}_S(\cdot)\rangle_{\mathsf{reg}}$ and generates the verification record T^*, which is sent to the adversary $\mathcal{A}$. $\mathcal{A}$ performs the verification protocols $\langle \mathcal{A}(\mathsf{un}_0, \mathsf{pw}_1, T^*, \mathsf{pk}_\mathcal{S}), \mathcal{R}_S(\cdot)\rangle_{\mathsf{ver}}$ and $\langle \mathcal{A}(\mathsf{un}_1, \mathsf{pw}_0, T^*, \mathsf{pk}_\mathcal{S}), \mathcal{R}_S(\cdot)\rangle_{\mathsf{ver}}$ and has free access to $\mathbb{O}$ again. However, if $\mathcal{A}$ requests to verify $(\mathsf{un}_0, \mathsf{pw}_0)$ or $(\mathsf{un}_1, \mathsf{pw}_1)$, $\mathcal{R}_S$ returns $\perp$ to avoid $\mathcal{A}$ getting the result directly.
- Output: Adversary $\mathcal{A}$ outputs his guess b' on b and wins the game if $b' = b$.

We refer to such adversary $\mathcal{A}$ as the PH-BIN-2 adversary and define adversary $\mathcal{A}$'s advantage as

$$Adv^{\mathcal{A},\mathbb{O}}_{\text{PH-BIN-2}}(\lambda) = |Pr[b = b'] - \frac{1}{2}|.$$

Definition 5. (**Binding**) We say that a PW-Hero service is PH-BIN secure if for any protocols in $\mathbb{O}$ and any PPT PH-BIN-1 adversary $\mathcal{A}_1$ and PH-BIN-2 adversary $\mathcal{A}_2$, advantages $Adv^{\mathcal{A}_1,\mathbb{O}}_{\text{PH-BIN-1}}(\lambda)$ and $Adv^{\mathcal{A}_2,\mathbb{O}}_{\text{PH-BIN-2}}(\lambda)$ both are negligible.

E. Privacy

We define the PH-PRI game, which is played between the challenger $\mathcal{R}$ acting as the client and the adversary $\mathcal{A}$ acting as the malicious server, as follows:

- Setup 1: Same as the $\mathsf{Setup\ 1}$ in the PH-OBL game.
- Setup 2: Adversary $\mathcal{A}$ has free access to $\mathbb{O}$ and generates a public key $\mathsf{pk}_\mathcal{S}$, a username un and two passwords, pw_0 and pw_1, which are sent to $\mathcal{R}$. And $\mathcal{R}$ performs the registration protocol $\langle \mathcal{R}(\mathsf{un}, \mathsf{pw}_0, \mathsf{pk}_\mathcal{S}), \mathcal{A}(\cdot)\rangle_{\mathsf{reg}}$ and outputs the verification record T^*.
- Challenge: Challenger $\mathcal{R}$ samples $b \leftarrow \{0,1\}$. With b, $\mathcal{R}$ performs the verification protocol $\langle \mathcal{R}(\mathsf{un}, \mathsf{pw}_b, T^*, \mathsf{pk}_\mathcal{S}), \mathcal{A}(\cdot)\rangle_{\mathsf{ver}}$. The adversary $\mathcal{A}$ has free access to $\mathbb{O}$ again.
- Output: Adversary $\mathcal{A}$ outputs his guess b' on b and wins the game if $b' = b$.

We refer to such adversary $\mathcal{A}$ as the PH-PRI adversary and define adversary $\mathcal{A}$'s advantage as

$$Adv^{\mathcal{A},\mathbb{O}}_{\text{PH-PRI}}(\lambda) = |Pr[b = b'] - \frac{1}{2}|.$$

Definition 6. (**Privacy**) We say that a PW-Hero service is PH-PRI secure if for any protocols in $\mathbb{O}$ and any PPT PH-PRI adversary $\mathcal{A}$, the advantage $Adv^{\mathcal{A},\mathbb{O}}_{\text{PH-PRI-1}}(\lambda)$ is negligible.

Appendix C
Formal Security Proof

This section proves the security of the PW-Hero scheme and show that it has obliviousness, hiding, forward security, binding, and privacy. To start with, we provide several assumptions: Let hash functions, H_0, H_1, H_2, H_3, be modeled as random oracles; Discrete logarithm (DL)[10] and decisional Diffie-Hellman (DDH)[11] assumptions both hold in group $\mathbb{G}$.

A. Obliviousness

Theorem 1. *Let hash function $H_1(\cdot)$ be a random oracle. Suppose DL and DDH assumptions hold in group $\mathbb{G}$. If there exists an adversary $\mathcal{D}$ who can break DDH assumption with $Adv^{\mathcal{D}}_{DDH}(\lambda)$, for any adversary $\mathcal{A}$, his advantage in the PH-OBL game is:*

$$Adv^{\mathcal{A},\mathbb{O}}_{PH\text{-}OBL} \leqslant Adv^{\mathcal{D}}_{DDH}(\lambda),$$

where $Adv^{\mathcal{A},\mathbb{O}}_{\text{PH-OBL}}$ is defined in Section B-A.

Proof. We prove the Theorem 1 by introducing four games in which $H_1(s, \mathsf{un}, \mathsf{pw})^{n_\mathcal{C}}$ gradually replaced by a random oracle, and proving that the transition of hidden bit b from 0 to 1 is indistinguishable.

- **Game 0:** Equivalent to the obliviousness game, PH-OBL.
- **Game 1:** Challenger simulates the random oracle H_1. On query $H_1(\cdot)$, the Challenger picks h from $\mathbb{Z}_q$ and returns g^h. Game 1 is functionally identical to Game 0.
- **Game 2:** Given a tuple (g, g^α, g^β), Challenger returns g^α and g^β on queries on $H_1(s, \mathsf{un}, \mathsf{pw}_b)$ and $H_1(s, \mathsf{un}, \mathsf{pw}_{1-b})$, respectively, and returns on other queries the same as Game 1. Game 2 is functionally identical to Game 1.
- **Game 3:** Given a tuple $(g, g^\alpha, g^\beta, g^\gamma, g^{\alpha\gamma}, g^{\beta\gamma})$, where $\alpha, \beta, \gamma \in \mathbb{Z}_q$. Let $r = g^\alpha$ and $n_\mathcal{C} = \gamma$. The Game 3 is identical to Game 2 except that when executing $\Phi.\langle \mathcal{C}(\mathsf{un}, \mathsf{pw}), \mathcal{S}()\rangle_{\mathsf{ver}}$, the Challenger returns $g^{\alpha\gamma}$ for $(r \cdot H_1(s, \mathsf{un}, \mathsf{pw}_b)/H_1(s, \mathsf{un}, \mathsf{pw}_b))^{n_\mathcal{C}}$ and returns $g^{\beta\gamma}$ for $(r \cdot H_1(s, \mathsf{un}, \mathsf{pw}_b)/H_1(s, \mathsf{un}, \mathsf{pw}_{1-b}))^{n_\mathcal{C}}$. Game 3 is functionally identical to Game 2. We build an adversary $\mathcal{D}$ who can break DDH assumption with the negligible advantage of $Adv^{\mathcal{D}}_{\text{DDH}}(\lambda)$. Thus, Pr[Distinguishing $(g, g^\alpha, g^\beta, g^\gamma, g^{\alpha\gamma})$ from $(g, g^\alpha, g^\beta, g^\gamma, g^{\beta\gamma})$ in Game 3] $\leqslant Adv^{\mathcal{D}}_{\text{DDH}}(\lambda)$.

Since the transition of b from 0 to 1 is indistinguishable in Game 3, the Theorem 1 is proved. $\square$

B. Hiding

Theorem 2. *Let hash function $H_1(\cdot)$ be a random oracle. Suppose DL and DDH assumptions hold in group $\mathbb{G}$. If there exists an adversary $\mathcal{B}$ who can break DL assumption with $Adv^{\mathcal{B}}_{DL}(\lambda)$ and an adversary $\mathcal{D}$ who can break DDH assumption with $Adv^{\mathcal{D}}_{DDH}(\lambda)$, for any adversary $\mathcal{A}$, his advantage in the PH-HID game is:*

$$Adv^{\mathcal{A},\mathbb{O}}_{PH\text{-}HID} \leqslant Adv^{\mathcal{B}}_{DL}(\lambda) + \frac{2q(\lambda)}{Z(\lambda)}(Adv^{\mathcal{B}}_{DL}(\lambda) + Adv^{\mathcal{D}}_{DDH}(\lambda)),$$

where $q(\lambda)$ is the number of queries to the Challenger by the adversary $\mathcal{A}$, and $Z(\lambda)$ is the size of $\mathbb{Z}_q$. In addition, the PH-HID game and $Adv^{\mathcal{A},\mathbb{O}}_{\text{PH-HID}}$ are defined in Section B-B.

[10]Discrete logarithm assumption says that given $(\mathbb{G}, g, y)$, for any PPT adversary $\mathcal{A}$, the probability of finding a $k \in \mathbb{Z}_q$, $y = g^k$ is negligible.

[11]Decisional Diffie-Hellman assumption says that for any PPT $\mathcal{A}$, the probability of distinguishing the tuple (g, g^a, g^b, g^{ab}) from (g, g^a, g^b, g^c) is negligible, where $a, b, c \in \mathbb{Z}_q$.

Proof. First, adversary $\mathcal{A}$ cracking the service key $k_{\mathcal{S}}$ from the password record $(\cdot)^{k_{\mathcal{S}}}$ is equivalent to solving the DL problem. Second, we introduce five games where $H_1(s, \mathsf{un}, \mathsf{pw})^{k_{\mathcal{S}}}$ gradually replaced by random value, and prove that the transition of hidden bit b from 0 to 1 is indistinguishable.

- **Game 0:** Equivalent to the hiding game, PH-HID.
- **Game 1:** Challenger simulates the random oracle H_1. On query $H_1(\cdot)$, the Challenger picks h from $\mathbb{Z}_q$ and returns g^h. Game 1 is functionally identical to Game 0.
- **Game 2:** Given a tuple (g, g^α, g^β), the Challenger returns g^α on queries on $H_1(s, \mathsf{un}, \mathsf{pw}_b)$, returns g^β on queries on $H_1(s, \mathsf{un}, \mathsf{pw}_{1-b})$, and returns on other queries the same as Game 1. In addition, Challenger records query history to $H_1(s, \mathsf{un}, \mathsf{pw}_0)$ and $H_1(s, \mathsf{un}, \mathsf{pw}_1)$. If a random oracle query is a history query, the Challenger aborts. Since the parameter s of H_1 is sampled from $\mathbb{Z}_q$ at random in the registration protocol, we assume $|\mathbb{Z}_q| = Z(\lambda)$. And then Pr[Challenger aborting in Game 2] $< \frac{2q(\lambda)}{Z(\lambda)}$ for most $q(\lambda)$ queries.
- **Game 3:** If adversary makes a history query, Challenger returns $h_1^{k_{\mathcal{S}}}$, where h_1 is came from the random oracle H_1 in Game 2 and $k_{\mathcal{S}}$ is sampled from $\mathbb{Z}_q$. In addition, we assume that adversary $\mathcal{B}$ breaks the DL assumption with the negligible advantage of $Adv_{\mathrm{DL}}^{\mathcal{B}}(\lambda)$. Thus, Pr[Solving $k_{\mathcal{S}}$ in Game 3] $\leqslant Adv_{\mathrm{DL}}^{\mathcal{B}}(\lambda)$.
- **Game 4:** Given a tuple $(g, g^\alpha, g^\beta, g^{k_{\mathcal{S}}}, g^{\alpha k_{\mathcal{S}}}, g^{\beta k_{\mathcal{S}}})$, when executing $\Phi.\langle \mathcal{C}(\mathsf{un}, \mathsf{pw}_b), \mathcal{S}()\rangle_{\mathsf{reg,ver}}$ protocols, Challenger returns $g^{\alpha k_{\mathcal{S}}}$ for $H_1(s, \mathsf{un}, \mathsf{pw}_b)^{k_{\mathcal{S}}}$ and returns $g^{\beta k_{\mathcal{S}}}$ for $H_1(s, \mathsf{un}, \mathsf{pw}_{1-b})^{k_{\mathcal{S}}}$. Game 4 is functionally identical to Game 3. We assume that adversary $\mathcal{D}$ can break the DDH assumption with the negligible advantage of $Adv_{\mathrm{DDH}}^{\mathcal{D}}(\lambda)$. Thus, Pr[Distinguishing $(g, g^\alpha, g^\beta, g^{k_{\mathcal{S}}}, g^{\alpha k_{\mathcal{S}}})$ from $(g, g^\alpha, g^\beta, g^{k_{\mathcal{S}}}, g^{\beta k_{\mathcal{S}}})$ in Game 4] $\leqslant Adv_{\mathrm{DDH}}^{\mathcal{D}}(\lambda)$.

Note that the transition of the hidden bit b from 0 to 1 is indistinguishable in Game 4, even if adversary $\mathcal{A}$ hits history query in most $q(\lambda)$ queries. □

C. Forward Security

Theorem 3. *Let hash function $H_1(\cdot)$ be a random oracle. Suppose DL and DDH assumptions hold in group $\mathbb{G}$. If there exists an adversary $\mathcal{A}'$ who can break hiding security of PW-hero with $Adv_{PH\text{-}HID}^{\mathcal{A}',\mathbb{O}}$, an adversary $\mathcal{B}$ who can break DL assumption with $Adv_{DL}^{\mathcal{B}}(\lambda)$ and an adversary $\mathcal{D}$ who can break DDH assumption with $Adv_{DDH}^{\mathcal{D}}(\lambda)$, for any adversary $\mathcal{A}$, his advantage in the PH-FOR game is:*

$$\begin{aligned} Adv_{PH\text{-}FOR}^{\mathcal{A},\mathbb{O}} &\leqslant Adv_{PH\text{-}HID}^{\mathcal{A}',\mathbb{O}} \\ &\leqslant \frac{2q(\lambda)}{Z(\lambda)}(Adv_{DL}^{\mathcal{B}}(\lambda) + Adv_{DDH}^{\mathcal{D}}(\lambda)), \end{aligned} \tag{6}$$

where $Adv_{\mathrm{PH\text{-}FOR}}^{\mathcal{A},\mathbb{O}}$ is defined in Section B-C.

Proof. For game PH-FOR-1, let $t_2^* = (r \cdot h_0)^{k_{\mathcal{S}}}$. After a secret key rotation, let the new secret key be $k_{\mathcal{S}}'$ such that $t_2^{*\prime} = ((r \cdot h_0)^{k_{\mathcal{S}}/k'_{\mathcal{S}}})^{k'_{\mathcal{S}}}$. Solving the DL problem $t_2^{*\prime} = ((r \cdot h_0)^{k_{\mathcal{S}}/k'_{\mathcal{S}}})^{k'_{\mathcal{S}}}$ can be reduced to solving $t_2^{*\prime} = (r' \cdot h'_0)^{k'_{\mathcal{S}}}$, where $r' = r^{k_{\mathcal{S}}/k'_{\mathcal{S}}}$ and $h'_0 = h_0^{k_{\mathcal{S}}/k'_{\mathcal{S}}}$. Due to the tuple $(r, h_0, k_{\mathcal{S}}, (r \cdot h_0)^{k_{\mathcal{S}}})$ and the tuple $(r', h'_0, k'_{\mathcal{S}}, (r' \cdot h'_0)^{k'_{\mathcal{S}}})$ are indistinguishable, adversary $\mathcal{A}$ in game PH-FOR-1 is equivalent to $\mathcal{A}'$ in game PH-HID.

Similarly, adversary $\mathcal{A}$ in game PH-FOR-2 can be equivalent to $\mathcal{A}'$ in game PH-HID. □

D. Binding

Theorem 4. *Let hash function $H_{0,1}(\cdot)$ be random oracles. Suppose DL assumption holds in group $\mathbb{G}$. If there exists an adversary $\mathcal{B}$ who can break DL assumption with $Adv_{DL}^{\mathcal{B}}(\lambda)$, for any adversary $\mathcal{A}$, his advantage in the PH-BIN game is:*

$$Adv_{PH\text{-}BIN}^{\mathcal{A},\mathbb{O}} \leqslant Adv_{DL}^{\mathcal{B}}(\lambda),$$

where $Adv_{\mathrm{PH\text{-}BIN}}^{\mathcal{A},\mathbb{O}}$ is defined in Section B-D.

Proof. We let $\mathcal{B}$ be a simulator which receives a discrete logarithm problem instance (g_0, g_1) and solves $\log_{g_0}(g_1)$. $\mathcal{B}$ records query history to H_0 and H_1 and maps (un) and $(s, \mathsf{un}, \mathsf{pw})$ respectively to random exponents of g_0 and g_1. When adversary $\mathcal{A}$ queries the random oracle H_1 on $(s, \mathsf{un}, \mathsf{pw})$, $\mathcal{B}$ checks whether $(s, \mathsf{un}, \mathsf{pw})$ is programmed. If so, it retrieves and returns its history value. Otherwise, it samples a exponent $a \in \mathbb{Z}_q$ and programs $H_1(s, \mathsf{un}, \mathsf{pw}) = g_1^a$. $\mathcal{B}$ simulates H_0 similarly, and we omit the details of it.

Recall that if adversary $\mathcal{A}$ wins the PH-BIN game, $\mathcal{A}$ needs to generate $\{(\mathsf{un}_0, \mathsf{pw}_0), (\mathsf{un}_1, \mathsf{pw}_1), T^*\}$ such that $\langle \mathcal{R}(\mathsf{un}_0, \mathsf{pw}_0, T^*), \mathcal{A}(\cdot)\rangle_{\mathsf{ver}}$ and $\langle \mathcal{R}(\mathsf{un}_1, \mathsf{pw}_1, T^*), \mathcal{A}(\cdot)\rangle_{\mathsf{ver}}$ both output "Accept". $T^* = \{h_0, s, t_1 = r \cdot h_1, t_2 = (r \cdot h_0)^{k_{\mathcal{S},\mathsf{reg}}}, t_3 = H_2(r)^{k_{\mathcal{S},\mathsf{reg}}}\}$.

If $\mathcal{A}$ can provide a ZKP proof $\pi_{0,\mathsf{ver}}$ for the first ver protocol, $((r \cdot h_1)/g_1^{b_0} g_0^{a_0})^{k_{\mathcal{S},0,\mathsf{ver}}} = (r \cdot h_0)^{k_{\mathcal{S},\mathsf{reg}}}$. And if $\mathcal{A}$ also can provide a proof $\pi_{1,ver}$ for the second ver protocol, $((r \cdot h_1)/g_1^{b_1} g_0^{a})^{k_{\mathcal{S},1,\mathsf{ver}}} = (r \cdot h_0)^{k_{\mathcal{S},\mathsf{reg}}}$.

Due to the correctness of $\pi_{0,\mathsf{ver}}$ and $\pi_{1,\mathsf{ver}}$, if Challenger accepts both returns from $\mathcal{A}$, $k_{\mathcal{S},0,\mathsf{ver}} = k_{\mathcal{S},1,\mathsf{ver}}$. We can easily obtain $g_0^{a_0} g_1^{b_0} = g_0^{a_1} g_1^{b_1}$, i.e., $\log_{g_0}(g_1) = (a_1 - a_0)/(b_0 - b_1)$, where $(\mathsf{un}_0, \mathsf{pw}_0) \neq (\mathsf{un}_1, \mathsf{pw}_1)$, (a_0, b_0) and (a_1, b_1) are sampled independently. This is equivalent to solving the DL problem $\log_{g_0}(g_1)$. Thus, if $\mathcal{A}$ wins with a non-negligible probability in the PH-BIN-1 game, then simulator $\mathcal{B}$ can solve the DL problem with non-negligible probability. □

E. Privacy

Theorem 5. *Let hash function $H_1(\cdot)$ be a random oracle. Suppose DDH assumption holds in group $\mathbb{G}$. If there exists an adversary $\mathcal{D}'$ who can break DDH assumption with $Adv_{DDH}^{\mathcal{D}'}(\lambda)$, for any adversary $\mathcal{A}$, his advantage in the PH-PRI game is:*

$$Adv_{PH\text{-}PRI}^{\mathcal{A},\mathbb{O}} \leqslant Adv_{DDH}^{\mathcal{D}'}(\lambda),$$

where $Adv_{\mathrm{PH\text{-}PRI}}^{\mathcal{A},\mathbb{O}}$ is defined in Section B-E.

Proof. We prove the Theorem 5 by introducing three games with the hidden bit b transition from 0 to 1 and proving that each transition is indistinguishable. Recall that the password record of $(\mathsf{un}, \mathsf{pw})$ is $T^* = \{H_0(\mathsf{un}), s, r \cdot H_1(s, \mathsf{un}, \mathsf{pw}), (r \cdot H_0(\mathsf{un}))^{k_{\mathcal{S}}}, H_2(r)^{k_{\mathcal{S}}}\}$.

- **Game 0:** The hidden bit $b = 0$.
- **Game 1:** Challenger simulates the random oracle H_1 and return g^a for the query on $H_1(s, \mathsf{un}, \mathsf{pw})$, where $a \leftarrow_{\$} \mathbb{Z}_q$. When executing $\Phi.\langle \mathcal{C}(\mathsf{un}, \mathsf{pw}_0), \mathcal{S}() \rangle_{\mathsf{ver}}$, Challenger samples $a_0, n_{0,\mathcal{C}} \leftarrow \mathbb{Z}_q$ and returns $g^{a_0 n_{0,\mathcal{C}}}$. Game 1 is functionally identical to Game 0.
- **Game 2:** $b = 1$. When executing $\Phi.\langle \mathcal{C}(\mathsf{un}, \mathsf{pw}_1), \mathcal{S}() \rangle_{\mathsf{ver}}$, Challenger samples $a_1, n_{1,\mathcal{C}} \leftarrow \mathbb{Z}_q$ and returns $g^{a_1 n_{1,\mathcal{C}}}$. Suppose an adversary $\mathcal{D}$ who can differentiate Game 2 from Game 1, which means that he can differentiate $\{g, g^{a_0}, g^{n_{0,\mathcal{C}}}, g^{a_1}, g^{n_{1,\mathcal{C}}}, g^{a_0 n_{0,\mathcal{C}}}\}$ from $\{g, g^{a_0}, g^{n_{0,\mathcal{C}}}, g^{a_1}, g^{n_{1,\mathcal{C}}}, g^{a_1 n_{1,\mathcal{C}}}\}$. So we can build an adversary $\mathcal{D}'$ who can break the DDH assamption with the negligible advantage of $Adv_{\mathsf{DDH}}^{\mathcal{D}'}(\lambda)$. Thus, $Pr[$Distinguishing Game 2 from Game 1$] \leqslant Adv_{\mathsf{DDH}}^{\mathcal{D}'}(\lambda)$.

□

APPENDIX D
EXPERIMENT SUPPLEMENT

We evaluate the time of all operations contained in the two groups, $\mathbb{G}$ and $\mathbb{S}$ $(= \{\mathbb{G}_1, \mathbb{G}_1, \mathbb{G}_t\})$, on which the schemes involved in the comparison were based. And the average runtime across 10,000 iteration on our machine is given in Tab. IV.

TABLE IV: Operation time (μs).

Charm [30]: NIST P 256		**Pyrelic [34]: BN 254**			
Random ZR	2.04	Random ZR	2.54	Add G2	3.02
Random G	37.47	Random ZR	69.04	Mul ZR	1.20
Hash ZR	1.43	Random G2	568.13	Mul GT	5.67
Hash G(H)	24.26	Random GT	408.92	Exp G1	156.91
Add ZR	0.27	Hash G1	36.23	Exp G2(E_{G_2})	427.52
Mul ZR	0.40	Hash G2(H_{G_2})	160.36	Exp GT(H_{G_t})	889.32
Mul G	0.82	Add ZR	1.49	Pair(pair)	747.87
Exp(E)	39.84	Add G1	2.55		
Random str: $\{0,1\}^* \to \{0,1\}^\lambda$			4.60		

Obviously, the pairing group operation used by Pythia [8] is more time-consuming compared to the multiplicative group operations of other schemes [9], [10]. In particular, the pair operation and the exponential operation of $\mathbb{G}_t$ elements take about 20 times longer than the exponential operations in the multiplicative group $\mathbb{G}$.