Universally Composable and Reliable Password Hardening Services

Shaoqiang Wu[1] and Ding Wang[1]

Nankai University {wushaoqiang@email.,wangding@}nankai.edu.cn

Abstract. The password-hardening service (PH) is a crypto service that armors canonical password authentication with an external key against offline password guessing in case the password file is somehow compromised/leaked. The game-based formal treatment of PH was brought by Everspaugh *et al.* at USENIX Security'15. Their work is followed by efficiency-enhancing PO-COM (CCS'16), security-patching Phoenix (USENIX Security'17), and functionality-refining PW-Hero (SRDS'22). However, the issue of single points of failure (SPF) inherently impairs the availability of these PH schemes. More specifically, the failure of a single PH server responsible for crypto computation services will suspend password authentication for *all* users.

We propose the notion of *reliable* PH, which improves the availability of PH by eliminating SPF. We present a modular PH construction, TF-PH, essentially a generic compiler that can transform any PH protocol into a reliable one without SPF via introducing threshold failover. Particularly, we propose a concrete *reliable* PH protocol, called TF-RePhoenix, a simple and efficient construction with RePhoenix (which improves over Phoenix at USENIX Security'17) as the PH module. Security is proven within the universally composable (UC) security framework and the random oracle model (ROM), where we, for the first time, formalize the ideal UC functionalities of PH and *reliable* PH. We comparatively evaluate the efficiency of our TF-PH with the canonical threshold method (taken as an example, the threshold solution introduced by Brost *et al.* at CCS'20 in a PH-derived domain – password-hardened encryption). Results show that our threshold failover-based solution to SPF provides optimal performance and achieves failover in a millisecond.

Keywords: Authentication, Password hardening, Single points of failure

1 Introduction

In recent years, unending password file breaches have made credential stuffing attacks the most significant threat against identity security [31]. For instance, the 2023 DBIR report [10] points out that 86% of the 1,287 web-based data breaches are due to stolen credentials. Generally, passwords are stored on the authentication server in salted-hash as recommended by NIST-800-63B [16] and the NCSC guideline [37]. However, this practice has an inherent weakness: It

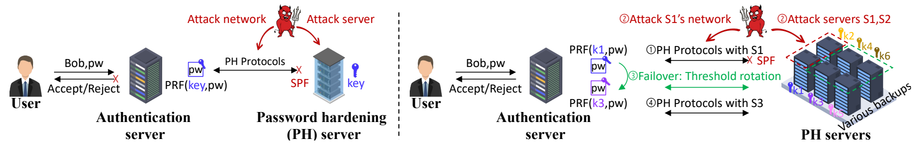


Fig. 1: An overview of password hardening service (PH), *with* and *without* single points of failure (SPF). Password authentication using PH hardens passwords with a remote PH key against offline password guessing (upon the authentication server compromise) and online password guessing (e.g., by rate-limiting and lockout; hence, the PH server is also called rate-limiter [27,4]). As shown on the left, all existing PH services are threatened by the *SPF* of the PH server (e.g., upon PH server corruption and network failure). This paper transforms the damaging threat into a remediable one on the right. In detail, it sets six PH servers with various keys (i.e.,k_1, k_2, k_3, $\cdots$, k_6) and failovers (i.e., ③ via a secure channel) when failure happens by using the (6,4)-threshold server rotation and password file update. A secure channel helps avoid malicious threshold failover.

only increases the cost of offline password guessing attacks but does not eliminate them. This weakness is fatal in reality, where the attackers constantly increase their offline guessing success rates by using powerful hardware (see [31,10]) and advanced guessing algorithms (e.g., [38,30]). As a result, an overwhelming fraction of passwords can be guessed/recovered (see [14,7]).

A promising solution to eliminate offline dictionary attacks (and credential stuffing attacks) is the Password Hardening service (PH) [29,11,9]. As shown in Fig. 1, the authentication server carries out an oblivious PH protocol with the PH server to evaluate the user-selected password as a pseudo-random value that is strengthened by the PH key and stores it as password verification information (known as a password file record). When a password needs to be verified, the PH server assists the authentication server in performing an oblivious PH protocol to generate the verification result (accept or reject). Throughout these processes, the PH server can learn nothing about user passwords. In particular, without the PH key, even the authentication server *cannot* crack the password file offline by independently verifying password guesses, let alone attackers who steal the password file. Attackers can only resort to online attacks by requesting a run of the PH verification protocol for every guess until they hit the target. The PH server can thwart such online attacks by rate-limiting and lockout [12].

1.1 Motivations

While existing PH schemes [11,32,28,24] significantly improve password storage security by developing secure and efficient PH protocols, they introduce a single point of failure that makes PH services less reliable [4]. Suppose the PH server is unreachable due to a network failure or malicious attack; that is a genuine possibility [26]. In that case, without the assistance of the PH server, the authentication server cannot independently verify its users' login passwords since its stored password records are encrypted with an external PH key. In short, the PH failure will disrupt all user logins for every authentication server subscribing to the PH service. To make matters worse, if the PH key is lost, all registered passwords are invalidated [4], leaving users with only a tedious password reset process to restart their accounts. Removing SPF for existing PHs can increase the trust of users and encourage authentication servers to subscribe to their services. Hence, the SPF issue should be resolved on an urgent basis.

Password-hardened encryption (PHE) [27] is a one-package solution that combines a PH instance (i.e., Phoenix [28]) and data encryption. PHE has the same SPF issue as PH. To eliminate its SPF, Brost *et al.* establish a threshold version [4] (namely T-PHE here). T-PHE [4] divides the PHE server's responsibility (the key and protocols) among n servers and guarantees that t servers can fully function. This way, as long as any t-of-n servers are correctly working on the encryption/decryption protocols, T-PHE [4] will not be interrupted by the failure of one or some servers. In many other works [4,1,40], the threshold method is often used to overcome the SPF problem. Similarly, we can deduce that the resolution could also work in existing PH schemes [11,32,28,24], although no such revision has been proposed.

However, the threshold method complicates protocols and increases communication rounds. Taking T-PHE [4] as an example, expanding the encryption protocol from one round to three rounds and the decryption protocol from two rounds to six rounds, makes latency two to three times that of PHE [27] even at the same setting of $n = t = 1$ [4]. Naturally, efficiency defects are also challenging to avoid in PH. This trade-offed performance ripples through the two fundamental protocols (i.e., registration/evaluation and login/authentication protocols), where users may perceive their increasing delay [2].

To our knowledge, there is no universally composable (UC) threshold PH scheme. For a non-UC threshold scheme [4], the security of an original PH scheme (with SPF) is not preserved in its threshold-improved scheme. The security needs to be re-proved across the board even if the original scheme has been proven secure. Second, a designed threshold solution cannot be directly migrated to other PH schemes with SPF. When threshold splitting a PH key, for example, protocol designers need first to gain insight into the original protocol and the computational process in which the key participates, and then translate the related computation into a splittable one, such as the complex matrix operation in T-PHE [4]. Although each PHS [11,32,28,24] may have a suitable threshold version, its protocol designers must individually rack their brains to design it and re-prove its security.

To sum up, the canonical threshold method is foreseeably feasible to eliminate the SPF of PH by carefully designing a sound version individually, just like PHE's [27] threshold version T-PHE [4]. However, its disadvantages are prominent: it comes at the cost of complexity and efficiency; the threshold method and its security are not universally composable.

1.2 Our work

We aim to build a reliable and efficient solution that eliminates the single points of failure of the PH server in *all* PH schemes. We arm them with a series of desirable properties such as a modular design with universally composable security and high efficiency (i.e., without substantial efficiency downgrade of main protocols for registration and login). In all, we make the following work.

We, for the first time, formalize the PH definition, $\mathcal{F}_{\mathrm{PH}}$, in the universally composable (UC) security framework [5]. According to the existing game-based definitions [11,28,24], PH protocols can be divided into two types. The first type [11] uses an oblivious pseudo-random function to evaluate the user password and generates a deterministic password record. It is executed regardless of registration and login. The second type [28,24] generates random password records that are non-deterministic and consistently different, even for the same password, and verifies login passwords via a separate PH authentication protocol. Since the second-type PH instantiations [28,24] are generally more efficient, we focus on and formalize them in the UC framework. As an independent interest, we define the first-type PH as the verifiable and partially oblivious pseudo-random function (VPOPRF) functionality in Appendix E. We prove that Pythia [11] (at USENIX Security'15) is a UC-secure VPOPRF protocol, which answers the question left in [11] on proving pseudo-randomness.

We demonstrate the practicality of our UC PH functionality ($\mathcal{F}_{\mathrm{PH}}$) by proposing the first UC PH scheme (named RePhoenix) that realizes $\mathcal{F}_{\mathrm{PH}}$ in the random oracle model (ROM). RePhoenix solves the SPF problem of Phoenix [28] without impairing efficiency. Moreover, RePhoenix eliminates the other defects of Phoenix [28], especially regarding the verifiabilities of the evaluation protocol and the "reject" branch of the authentication protocol, making fake password records and unreliable verification results detectable.

We formalize reliable PH as $\mathcal{F}_{\mathrm{rPH}}$ that strengthens the PH security notion by eliminating the single points of failure in the UC framework based on $\mathcal{F}_{\mathrm{PH}}$. In the reliable PH functionality, we set n servers holding their own independently generated random keys and label them with the state of online or offline. We designate an online server as the PH server defined in $\mathcal{F}_{\mathrm{PH}}$ at a time and call it the on-duty server. It interacts with the authentication server for password evaluation and authentication like in $\mathcal{F}_{\mathrm{PH}}$. Upon the failure of the on-duty server, we allow a (n,t)-threshold server rotation to activate an alternative online server to take over the failed on-duty server, as shown in Fig. 1. Failover is frequently used to ensure high availability in the field of cloud [39] and networks [35,3] from system, architecture and management. To our knowledge, we, for the first time, provide a cryptographical failover for a cryptographical protocol.

We present TF-PH, a specialized compiler for solving the SPF issue, which can transform any UC-secure PH protocol into a UC-secure reliable PH protocol. We take advantage of the modular security of the UC framework to make the security proof clear and simple. To this end, we define the above SPF solution in $\mathcal{F}_{\mathrm{rPH}}$ as the ideal threshold failover functionality $\mathcal{F}_{\mathrm{TF}}$ in the UC framework. By splitting the update token using Shamir's secret sharing algorithm [33] in the initialization phase and reconstructing the update token upon failure happens, a simple TF protocol UC-realizes $\mathcal{F}_{\mathrm{TF}}$. Therefore, TF-PH can UC-realize $\mathcal{F}_{\mathrm{rPH}}$ just by calling $\mathcal{F}_{\mathrm{PH}}$ and $\mathcal{F}_{\mathrm{TF}}$. For practical efficiency considerations, failover requires that the token be transmission efficient with compact and constant size and that the password file update be computation efficient (e.g., batch updates locally). For security, failover should satisfy forward security. For a user, failover should be a transparent process that requires no additional input from the user.

Importantly, our TF-PH enjoys an essential property for use in practice, i.e., *modularity*. It allows for all existing and future PH schemes that UC-realize $\mathcal{F}_{\text{PH}}$ to replace the $\mathcal{F}_{\text{PH}}$ sub-routine of TF-PH. In other words, TF-PH solves the single points of failure problem once and for all. This goes beyond the threshold method that works only on a case-by-case basis, as mentioned earlier.

Equipping our TF-PH with *UC security* is natural and meaningful. We reflect from the existing threshold work [4] that re-providing security is repetitive and tedious after improving a proven secure original PH scheme. We offer the UC security proof that shows that our TF-PH UC-realizes the reliable PH functionality. Therefore, we can benefit from the UC theorem [5]. The UC security of an improved reliable PH with TF-PH can be directly obtained from the UC security of the original PH scheme.

As for efficiency, low latency and user transparency are fundamental properties of PH [24]. In this term, TF-PH maintains these properties if inherent in the original PH scheme. The SPF solution should not suffer from the performance of the protocols that users participate in (e.g., registration and login). Accordingly, our TF-PH provides *optimal performance*, which means that the improved reliable PH with TF-PH has the same efficiency as the original PH. That is, the SPF problem is solved in TF-PH without sacrificing efficiency. Achieving this demanding goal is challenging. For this, we find an acceptable balance between performance and security. Our TF-PH does not enhance the protection of the PH protocols to the threshold security, whereas the threshold method does.

Finally, we instantiate a concrete reliable PH TF-PH$^{\mathcal{F}_{\text{PH}}\to\text{RePhoenix},\mathcal{F}_{\text{TF}}\to\text{TF}}$, called TF-RePhoenix. We analyze the performance of the main protocols and find that TF-RePhoenix has no performance loss as expected compared with the original RePhoenix. As a comparison, T-PHE [4] has a loss of 15%∼80%. Furthermore, we implement TF-RePhoenix and measure the time cost of failover. The experiment results show that it efficiently addresses single points of failure, with service failover in milliseconds. As $(n, t) = (15, 4)$ an example, the failover latency is 0.75 milliseconds, where 0.39 milliseconds for reconstructing the update token and 0.36 milliseconds for updating a password record.

Contributions. In summary, we make the contributions:

- We, for the first time in the UC framework, define the password hardening service (PH) functionality $\mathcal{F}_{\text{PH}}$ in Section 2.1 and the reliable PH functionality $\mathcal{F}_{\text{rPH}}$ without single points of failure (SPF) in Section 3.
- We present a reliable PH protocol named TF-PH in the $(\mathcal{F}_{\text{PH}}, \mathcal{F}_{\text{TF}})$-hybrid world in Section 4, where $\mathcal{F}_{\text{TF}}$ is the threshold failover functionality. TF-PH is a compiler that can transform any UC-secure PH protocol into a UC-secure reliable PH protocol without SPF using threshold failover.
- We provide an efficient and UC-secure PH protocol in Section 2.2, improved over the state-of-the-art Phoenix [28] mainly by refining verifiability under

the DDH assumption in ROM. Based on the UC theorem, we provide a concrete reliable PH protocol named TF-RePhoenix in Section 5.

– Experiments in Section D demonstrate that the failover in TF-RePhoenix is low-latency, taking only a millisecond to complete.

Notations. Let κ denote the security parameter. $\mathbb{Z}_n$ denotes the set of integers $\{1, 2, \dots, n\}$. For $a, b \in \mathbb{Z}$ and $a \le b$, $[a, b]$ denotes the set $\{a, a+1, \dots, b-1, b\}$. Simplified $[b]$ denotes the set $[1, b]$. If S is a set, then $|S|$ denotes its cardinality, which can be as subscript (i.e., $S_{|S|}$); that is, $\{s_i\}_t, i \in [n]$ denotes a set of t s_i's. We use $[\![s]\!]$ as a shorthand for $(s_1, \cdots, s_n)$ which are a set of (n, t)-shares of s. A concrete threshold scheme will specify how the shares will be generated with t and n as threshold parameters, where n denotes the number of shares s is split into and t denotes the number of shares required to reconstruct s.

2 The Password Hardening Service Functionality $\mathcal{F}_{\mathbf{PH}}$

Password hardening services (PH) provide secure password storage in an authentication server, where passwords are protected by an external key held by a PH server. Any insider and outsider attackers of the authentication server, who obtain the password file, cannot offline brute force crack passwords without knowing the PH key. A password hardening service is described as a set of two-party protocols executed between the authentication server (AS) and the PH server (RL). It does not interact directly with users, so a password hardening service cannot solve problems with the passwords themselves (e.g., weak passwords, reused passwords). Generally, a username and password pair $(\mathsf{un}, \mathsf{pw})$ is assumed to be transmitted from a user to the authentication server via a secure channel (i.e., "password over TLS") [11,28,27]. The authentication server takes the $(\mathsf{un}, \mathsf{pw})$ as inputs, and the PH server takes a PH key as input; specifically in the multi-client scenario, the PH server holds a master key and resorts to a key derivation function for individual PH keys for every authentication server. The PH server can learn nothing about user passwords. Since the authentication server introduces external cryptographic protection for password-derived values, an online authentication protocol with the PH server is necessary for verifying each password candidate. In this way, the PH server can rate-limit online password guessing attacks. Therefore, the PH server is also called a rate limiter [27,4].

There are two main PH protocols: evaluation and authentication. In the evaluation protocol, AS takes a username and a registration password as inputs and generates a (pseudo) random value as the password record. After that, AS discards the password and stores the password record in its password file. In the authentication protocol, AS inputs a login password and the username-indexed password record, with the output being the password verification result. This protocol verifies whether the login password matches the registered one underlying the password record. Obliviousness ensures that RL learns nothing about the AS inputted registration/login password.

Existing research on PH [32,28,27] defines the security properties through security games that cover partial obliviousness, hiding, soundness, and forward security [32,28,27]. Partial obliviousness ensures RL learns nothing about AS input user password from the PH protocol interactions. However, the username input may not be oblivious, such as in Phoenix [28]. It is recommended [11,32] to opt for a username tweak (i.e., a deterministic username-derived value) instead of sending the username in plaintext. Hiding captures that passwords remain hidden even when the password file records are available. Proving this property directly is difficult. Generally, one can prove it by proving the indistinguishability of the password record value and a random value [28], thereby proving randomness. Soundness ensures that AS will accept the correct password as valid for one password record and reject any wrong password. This requires all protocols to be verifiable. RL cannot trick AS into accepting incorrect outputs, including ineffective password evaluation values or wrong password authentication results. Forward security means the old PH key is ineffective in recovering password information from updated password records.

Further, we define randomness to ensure the non-deterministic outputs of the evaluation protocol, even with the same password input. This prevents attackers from verifying passwords through the evaluation protocol. Moreover, the authentication protocol requires unpredictability to ensure that AS cannot generate effective protocol outputs independently. This prevents online attackers from using a limited number of online requests to verify a large number of password guesses (a counterexample is in [32], where a single verification query is sufficient to brute-force the password afterward).

2.1 Defination of Password Hardening Functionality $\mathcal{F}_{\mathbf{PH}}$

The Universal Composability (UC) framework for expressing security is stronger than game-based security models. This is attributed to its ability to accommodate arbitrary interactions between protocol instances [5] and to encapsulate security in scenarios involving arbitrary password correlations, password typos, and arbitrary password information leakage [6]. Many (and increasingly more) important works [6,22,18] are presented under UC.

We, *for the first time*, propose the UC password hardening definition, $\mathcal{F}_{\mathrm{PH}}$ shown in Fig. 2-Fig. 3. Exploring PH protocols under the ideal functionality of $\mathcal{F}_{\mathrm{PH}}$ in the UC security framework allows us to benefit from the UC theorem[1] [5]. As in Section 4, we can modularize the solution to the single points of failure (SPF) and present an $\mathcal{F}_{\mathrm{PH}}$-hybrid compiler for eliminating SPF for all PH protocols which UC-realize $\mathcal{F}_{\mathrm{PH}}$. In this way, later researchers can temporarily ignore the SPF problem, focus on developing PHS itself, and then simply introduce our modular SPF solution to obtain an enhanced version without SPF. Furthermore, in a multi-client scenario where the PH server serves multiple authentication servers in parallel, UC security remains.

[1] Briefly put, protocols that run multiple UC-secure protocols simultaneously or in combination are also UC-secure.

Public Parameter and Conventions

- Output-length ℓ and token-length τ, polynomial in security parameter κ.
- The lmt is the limit number of online authentications allowed within a window.
- If no $\mathsf{prf}(\cdot)$ is defined until referenced, set $\mathsf{prf}(\cdot) \leftarrow\!\!\$\ \{0,1\}^{2\kappa}$.

Initialization

- On input $(\textsc{Init}, sid, \Delta k)$ from RL, send $(\textsc{Init}, sid, \mathsf{RL})$ to $\mathcal{A}^*$, mark RL as fresh, set $\mathsf{tx}(\mathsf{RL}) := 0$, and record $\langle \mathsf{token}, \mathsf{RL}, \Delta k\rangle$.

Compromising Server (with permission from $\mathcal{E}$)

- On $(\textsc{Compromise}, sid, \mathsf{RL})$ from $\mathcal{A}^*$, if RL is fresh, mark RL as COMPROMISED.

Online Evaluation

- On input $(\textsc{Eval}, sid, esid, \mathsf{RL}, \mathsf{un}, \mathsf{pw})$ from AS, send $(\textsc{Eval}, sid, esid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid})$ for $\mathsf{uid} \leftarrow \mathsf{prf}(\mathsf{un})$ to $\mathcal{A}^*$, and record $\langle esid, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$.
- On $(\textsc{EvalResp}, sid, esid, \mathsf{RL}, \mathsf{uid}^*)$ from $\mathcal{A}^*$ for honest RL, output $(\textsc{EvalResp}, sid, esid, \mathsf{uid}^*)$ to RL, record $\langle \mathsf{count}, \mathsf{uid}^*, c := 0\rangle$ for rate-limiting online authentication.
- On $(\textsc{StorePwdFile}, sid, esid, \mathsf{AS}, \mathsf{flag})$ from $\mathcal{A}^*$, ignore this message if there is no session record $\langle esid, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$.
  - If $\mathsf{flag} = \top$, record $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$, mark the file record as FRESH, and output $(\textsc{Eval}, sid, esid, \rho)$ for $\rho \leftarrow\!\!\$\{0,1\}^{\ell}$ to AS.
  - If $\mathsf{flag} = \bot$, output $(\textsc{Eval}, sid, esid, \bot)$ to AS.

Stealing Password Data and Offline Password Test

- On $(\textsc{StealPwdFile}, sid, \mathsf{AS}, \mathsf{uid})$ from $\mathcal{A}^*$:
  - If there is a file record $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$ s.t. $\mathsf{uid} = \mathsf{prf}(\mathsf{un})$, re-mark the file record COMPROMISED if it is FRESH, and send "password file stolen" to $\mathcal{A}^*$. For each $\langle \mathsf{file}, (\mathsf{AS}, \mathsf{RL}), \mathsf{RL}', \mathsf{un}, \mathsf{pw}\rangle$ s.t. $\mathsf{uid} = \mathsf{prf}(\mathsf{un})$, re-mark the file record COMPROMISED if it is FRESH.
    - If there is a record $\langle \mathsf{offlinetest}, \mathsf{pw}\rangle$, send pw to $\mathcal{A}^*$.
  - Otherwise, send "no password file" to $\mathcal{A}^*$.
- On $(\textsc{OfflineTestPwd}, sid, \mathsf{uid}, \mathsf{pw}^*)$ from $\mathcal{A}^*$, ignore this message if there is no file record $\langle \mathsf{file}, \cdot, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$ s.t. $\mathsf{uid} = \mathsf{prf}(\mathsf{un})$ or if RL is not COMPROMISED.
  - If the file record is FRESH, record $\langle \mathsf{offlinetest}, \mathsf{pw}\rangle$.
  - Otherwise, check that $\mathsf{pw}^* = \mathsf{pw}$.
    - If $\mathsf{pw}^* = \mathsf{pw}$, return "correct guess" to $\mathcal{A}^*$.
    - Else return "wrong guess" to $\mathcal{A}^*$.

Online Authentication

- On input $(\textsc{Auth}, sid, asid, \mathsf{RL}, \mathsf{un}, \mathsf{pw}')$ from AS, ignore this message if there is no file record $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$. Otherwise, send $(\textsc{Auth}, sid, asid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid})$ for $\mathsf{uid} \leftarrow \mathsf{prf}(\mathsf{un})$ to $\mathcal{A}^*$, and record $\langle asid, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}'\rangle$.
- On $(\textsc{AuthResp}, sid, asid, \mathsf{RL}, \mathsf{uid}^*)$ from $\mathcal{A}^*$ for honest RL, output $(\textsc{AuthResp}, sid, asid, \mathsf{uid}^*)$ to RL, retrieve $\langle \mathsf{count}, \mathsf{uid}^*, c\rangle$, and if $c < \mathsf{lmt}$, do: c ++, $\mathsf{tx}(\mathsf{RL})$ ++ and return SUCC to $\mathcal{A}^*$.
- On $(\textsc{AuthResult}, sid, asid, \mathsf{AS}, \mathsf{flag})$ from $\mathcal{A}^*$, ignore this message if there is no record $\langle asid, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}'\rangle$ or if $\mathsf{tx}(\mathsf{RL}) = 0$. Otherwise, set $\mathsf{tx}(\mathsf{RL})$ $--$.
  - If $\mathsf{flag} = \top$, retrieve the file record $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$.
    - If $\mathsf{pw}' = \mathsf{pw}$, output $(\textsc{Auth}, sid, asid, \text{"}\mathsf{accept}\text{"})$ to AS.
    - Else output $(\textsc{Auth}, sid, asid, \text{"}\mathsf{reject}\text{"})$ to AS.
  - If $\mathsf{flag} = \bot$, send $(\textsc{Auth}, sid, asid, \bot)$ to AS.

Online Password Test

- On $(\textsc{TestPwd}, sid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid}, \mathsf{pw}^*)$ from $\mathcal{A}^*$ for honest RL, ignore this message if there is no $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$ marked COMPROMISED s.t. $\mathsf{uid} = \mathsf{prf}(\mathsf{un})$, or if there is no $\langle \mathsf{count}, \mathsf{uid}, c\rangle$, or if $c = \mathsf{lmt}$. Otherwise, set c ++ and check that $\mathsf{pw}^* = \mathsf{pw}$.
  - If $\mathsf{pw}^* = \mathsf{pw}$, return "correct guess" to $\mathcal{A}^*$.
  - Else return "wrong guess" to $\mathcal{A}^*$.

Fig. 2: Password Hardening service (PH) functionality $\mathcal{F}_{\text{PH}}$.

File Record Update

- On input $(\textsc{Update}, sid, \mathsf{un}, (\mathsf{RL}, \Delta k), (\mathsf{RL}', \Delta k'))$ from AS, ignore this message if there are no token records $\langle \mathsf{token}, \mathsf{RL}, \Delta k \rangle$ and $\langle \mathsf{token}, \mathsf{RL}', \Delta k' \rangle$. Otherwise, retrieve $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw} \rangle$, modify it into $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}', \mathsf{un}, \mathsf{pw} \rangle$, and re-mark it FRESH if it is COMPROMISED.
- On input $(\textsc{Update}, sid, \mathsf{uid}, (\mathsf{RL}, \Delta k), (\mathsf{RL}', \Delta k'))$ from $\mathcal{A}^*$, ignore this message if there are no token records $\langle \mathsf{token}, \mathsf{RL}, \Delta k \rangle$ and $\langle \mathsf{token}, \mathsf{RL}', \Delta k' \rangle$. Otherwise, retrieve record $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw} \rangle$ s.t. $\mathsf{uid} = \mathsf{prf}(\mathsf{un})$, add record $\langle \mathsf{file}, (\mathsf{AS}, \mathsf{RL}), \mathsf{RL}', \mathsf{un}, \mathsf{pw} \rangle$ with the same mark COMPROMISED/FRESH.

Fig. 3: Password file record update functionality of $\mathcal{F}_{\text{PH}}$.

Our PH functionality $\mathcal{F}_{\text{PH}}$ includes four protocol phases: initialization, evaluation, authentication, and update, with two parties denoted by AS (i.e., the authentication server) and RL (i.e., the PH server), and the ideal adversary $\mathcal{A}^*$. $\mathcal{F}_{\text{PH}}$ grants $\mathcal{A}^*$ adaptive capabilities, including compromising RL and stealing password data from AS. Without loss of generality, initialization is via secure channels [11,28,27,22]. However, $\mathcal{A}^*$ can eavesdrop on and tamper with messages transmitted between AS and RL during the evaluation and authentication phases. $\mathcal{F}_{\text{PH}}$ defines the allowed information leakage by specifying the message sent to $\mathcal{A}^*$.

Initialization. The protocol session is initialized via an $\textsc{Init}$ message and is identified by a session identifier sid. Honest RL initializes by registering its update token Δk, which specifies the secret key held by RL. A simple way to understand it is: sid is bound to a fixed key base (virtual), and the update token Δk can rotate from the key base to the key of RL. We use RL to refer specifically to the initialized PH server in the following.

Evaluation. $\mathcal{F}_{\text{PH}}$ stipulates that online evaluation is a two-round interaction protocol over a public channel. Considering that there may be multiple evaluation sessions, we define $\mathcal{F}_{\text{PH}}$ as a multiple-session one [13].

- On an $\textsc{Eval}$ message (including username un and password pw) from AS, $\mathcal{F}_{\text{PH}}$ starts an evaluation session, identified by the sub-session identifier $esid$. $\mathcal{F}_{\text{PH}}$ defines a leakage of no more than the username tweak $\mathsf{uid} = \mathsf{prf}(\mathsf{un})$ (prf denotes a pseudo-random function) by sending $(\textsc{Eval}, sid, esid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid})$ to $\mathcal{A}^*$, which means that AS sends an evaluation request including uid over the public channel eavesdropped on by $\mathcal{A}^*$.
- When receiving a $\textsc{EvalResp}$ message with the same identifier $esid$, $\mathcal{F}_{\text{PH}}$ interprets it as RL responding to the AS's evaluation request. The message parameter uid^* is specified by $\mathcal{A}^*$, due to $\mathcal{A}^*$ can tamper with uid over the public channel from AS to RL. A syntactically meaningful action, $\mathcal{F}_{\text{PH}}$ outputs $(\textsc{EvalResp}, sid, esid, \mathsf{uid}^*)$ to RL. This means RL learns the username weak from the evaluation interaction. Additionally, a counter record $\langle \mathsf{count}, \mathsf{RL}, \mathsf{uid}^*, c \rangle$ is initiated for rate-limiting the RL's responses to authentication requests corresponding to uid^*.
- When receiving $\textsc{StorePwdFile}$ message, $\mathcal{F}_{\text{PH}}$ outputs $\rho \leftarrow\!\!\$ \{0,1\}^\ell$ to AS if $\mathsf{flag} = \top$, where flag defines verifiability. Following [19], the flag value of $\top$

or $\perp$ indicates whether the AS received value from RL is correctly calculated following the protocol specification. Additionally, the $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw} \rangle$ is recorded as the password file record and marked as FRESH, which signifies it has been kept away from $\mathcal{A}^*$ until it is re-marked as COMPROMISE. In particular, the evaluated output ρ is a non-deterministic random value, different from the pseudo-random output in $\mathcal{F}_{\mathrm{OPRF}}$ [19,22]. This satisfies randomness. Therefore, we don't restrain $\mathcal{A}^*$ from predicting one more evaluation output. Accordingly, $\mathcal{F}_{\mathrm{PH}}$ does not define the quantitative consistency between two messages of EvalResp and StorePwdFile.

Authentication. $\mathcal{F}_{\mathrm{PH}}$ stipulates that online authentication is also a two-round interaction protocol over a public channel:

– AS sends an Auth message with $(\mathsf{un}, \mathsf{pw}')$ to $\mathcal{F}_{\mathrm{PH}}$ to initiate an authentication session identified by a sub-session identifier *asid*. It is ignored in two cases where there is no evaluated password file record for the un.
– AuthResp denotes the RL's response to the authentication request. Due to RL rate-limiting authentication requests per user account, $\mathcal{F}_{\mathrm{PH}}$ ignores this message if the uid-indexed counter c (recorded in count) reaches the limit number lmt, else increase the c. Following [19], we introduce a ticketing mechanism to define authentication unpredictability. Each RL is associated with a ticket counter $\mathsf{tx}(\mathsf{RL})$ that counts the unused server responses to authentication requests. Therefore, when the rate limit is not triggered, $\mathcal{F}_{\mathrm{PH}}$ increments the $\mathsf{tx}(\mathsf{RL})$ by one to indicate RL newly responded a request.
– Upon receiving AuthResult message, $\mathcal{F}_{\mathrm{PH}}$ normally outputs accept or reject, denoting the inputted password pw' by AS in this Auth session is correct or wrong. $\mathcal{F}_{\mathrm{PH}}$ saves the registered password pw in the file record, so it can easily determine the password correctness by checking $\mathsf{pw}' = \mathsf{pw}$. Note that $\mathcal{F}_{\mathrm{PH}}$ checks whether the response tickets is exhausted (i.e., $\mathsf{tx}(\mathsf{RL}) = 0$), and only if not $\mathcal{F}_{\mathrm{PH}}$ outputs the authentication result as usual and decrease the ticket counter. Therefore, verifying more password guesses than the number of online authentications is impossible. Same as StorePwdFile, the verifiability of the authentication protocol is defined by adding a flag as a parameter. If $\mathsf{flag} = \perp$ (i.e., the AS received response is not calculated following the protocol specification), $\mathcal{F}_{\mathrm{PH}}$ outputs $\perp$.

File record update. The Update message is used to update the password file record evaluated by RL into ones by RL'. The message parameter includes the RL and RL'-registered update token: $\Delta k, \Delta k'$. (This is necessary for our modular token-based solution to single points of failure, which we will explain in Section 4.) Upon receiving this message from AS, $\mathcal{F}_{\mathrm{PH}}$ modifies all $(\mathsf{RL}, \Delta k)$ in file records into the specified $(\mathsf{RL}', \Delta k')$. The update has forward security. It makes the old server (key) ineffective for authentication of the new password file and vice versa by $\mathcal{F}_{\mathrm{PH}}$.

Adversary capabilities. The ideal adversary $\mathcal{A}^*$ controls the public channel between AS and RL. $\mathcal{F}_{\mathrm{PH}}$ defines the $\mathcal{A}^*$'s capability to eavesdrop on the public channel by some information leakage, e.g., sending uid to $\mathcal{A}^*$ in Eval and Auth. $\mathcal{F}_{\mathrm{PH}}$ defines the $\mathcal{A}^*$'s capability to tamper with messages transmitted on the public channel by receiving some messages forwarded by $\mathcal{A}^*$ and allowing $\mathcal{A}^*$ specifying the messages' parameter, e.g., EvalResp, StorePwdFile, AuthResp and AuthResult. Moreover, $\mathcal{F}_{\rm PH}$ addresses the following queries from $\mathcal{A}^*$ to clarify other ideal adversary capabilities.

- With the permission of the environment, $\mathcal{A}^*$ can compromise a specified RL by sending the message of $(\textsc{Compromise}, sid, \mathsf{RL})$, after RL is initialized. Then, the RL is inserted into $\mathsf{compromised}$, collecting all compromised servers.
- With the permission of the environment, $\mathcal{A}^*$ can compromise a specified AS and steal the password file record of a specified uid by sending the message of $(\textsc{StealPwdFile}, sid, \mathsf{AS}, \mathsf{uid})$. After that, the corresponding file record $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw} \rangle$ is re-marked as COMPROMISED. In particular, if the RL has already compromised (i.e., $\mathsf{RL} \in \mathsf{compromised}$) and the correct password pw has been pre-computed offline via a OfflineTestPwd message (i.e., $\mathsf{pw} \in \mathsf{offlinetest}$), herein returns pw to $\mathcal{A}^*$.
- After compromising RL and before stealing the password file record, $\mathcal{A}^*$ can mount pre-computation attacks. $\mathcal{A}^*$ can use OfflineTestPwd to build its rainbow table. All pre-computed password guesses are stored in $\mathsf{offlinetest}$. As described in StealPwdFile, while stealing the password record, $\mathcal{A}^*$ can determine which pre-computed password guess is correct.
- After compromising RL and stealing the password record, $\mathcal{A}^*$ can test password guesses offline by sending the message of $(\textsc{OfflineTestPwd}, sid, \mathsf{uid}, \mathsf{pw}^*)$, where uid is the targeted username tweak and pw^* is the guessed password by $\mathcal{A}^*$. $\mathcal{F}_{\rm PH}$ answers $\mathcal{A}^*$ with "correct guess" or "wrong guess".
- After stealing the password record, $\mathcal{A}^*$ can verify password guesses online via TestPwd. This capability is subject to RL rate-limiting online authentication. Therefore, only if the corresponding c does not reach the limit number, $\mathcal{F}_{\rm PH}$ answers $\mathcal{A}^*$ with "correct guess" or "wrong guess".
- If $\mathcal{A}^*$ learns the registered tokens of RL and RL', it has the ability to update the file record evaluated by RL to the corresponding file record evaluated by RL'. $\mathcal{F}_{\rm PH}$ defines this ability by recording $\langle \mathsf{file}, (\mathsf{AS}, \mathsf{RL}), \mathsf{RL}', \mathsf{un}, \mathsf{pw} \rangle$. The file record has the same mark with $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw} \rangle$.

Logically speaking, $\mathcal{A}^*$ can evaluate a new password record offline after compromising RL via a message such as OfflineEval. As in the Eval, each OfflineEval query will produce an uncorrelated random output $\rho \leftarrow\!\!\$ \{0,1\}^{\ell}$. Considering that this provides no benefit to $\mathcal{A}^*$, we do not define it in $\mathcal{F}_{\rm PH}$.

Security properties in the existing game-based definitions [32,28,27]. Existing works [32,28,27] define the security properties of a PH protocol in the game-based security model. Specifically, they define a security game for each security property and prove that the adversary wins with negligible probability. We describe the same security games in Fig. 4 as [32,28,27]. By proving the following game-based security theorem, we show that our ideal PH functionality $\mathcal{F}_{\rm PH}$ covers all security properties proposed by existing work, including password obliviousness, hiding, soundness, and forward security.

```
$\mathsf{Obl}^b_{\mathcal{F}_{\mathrm{PH}},\mathcal{A}}(1^\kappa)$
    $\mathsf{st}{\leftarrow}\mathcal{A}_1.\textsc{Init}(sid,\cdot)$
    $(\mathsf{pw}_0,\mathsf{pw}_1,\mathsf{st}){\leftarrow}{\mathcal{A}_2}^{\textsc{Eval},\textsc{Auth}}(\mathsf{st})$
    $\mathsf{st}{\leftarrow}\mathsf{AS}.\textsc{Eval}(sid,\cdot,\mathcal{A}_3(\mathsf{st}),\cdot,\mathsf{pw}_b)$
    $b{\leftarrow}{\mathcal{A}_4}^{\textsc{Eval},\textsc{Auth}}(\mathsf{st})$
    return $b$
```

```
$\mathsf{Sou}_{\mathcal{F}_{\mathrm{PH}},\mathcal{A}}(1^\kappa)$
    $\mathsf{st}{\leftarrow}\mathcal{A}_1.\textsc{Init}(sid,\cdot)$
    $(\mathsf{un},\mathsf{pw}_0,\mathsf{pw}_1,\mathsf{st}){\leftarrow}{\mathcal{A}_1}^{\textsc{Eval},\textsc{Auth}}(\mathsf{st})$
    $\mathsf{st}{\leftarrow}\mathsf{AS}.\textsc{Eval}(sid,\cdot,\mathcal{A}_2(\mathsf{st}),\mathsf{un},\mathsf{pw}_0)$
    $b_0{\leftarrow}\mathsf{AS}.\textsc{Auth}(sid,\cdot,\mathcal{A}_3(\mathsf{st}),\mathsf{un},\mathsf{pw}_0)$
    $b_1{\leftarrow}\mathsf{AS}.\textsc{Auth}(sid,\cdot,\mathcal{A}_3(\mathsf{st}),\mathsf{un},\mathsf{pw}_1)$
    $b{\leftarrow}b_0{\wedge}(\mathsf{pw}_0{\neq}\mathsf{pw}_1){\wedge}b_1$
    return $b$
```

```
$\mathsf{Hid}^b_{\mathcal{F}_{\mathrm{PH}},\mathcal{A}}(1^\kappa)$
    $\mathsf{RL}.\textsc{Init}(sid,\cdot)$
    $(\mathsf{pw}_0,\mathsf{pw}_1,\mathsf{st}){\leftarrow}{\mathcal{A}_1}^{\textsc{Init},\textsc{Eval},\textsc{Auth}}(\mathsf{st})$
    $\mathsf{st}{\leftarrow}\mathcal{A}_2(\mathsf{st}).\textsc{Eval}(sid,\cdot,\mathsf{RL},\cdot,\mathsf{pw}_b)$
    $b{\leftarrow}{\mathcal{A}_3}^{\textsc{Eval},\textsc{Auth}}(\mathsf{st})$ /* $\mathcal{A}_3.\textsc{Auth}$
     with $\mathsf{pw}_0$ or $\mathsf{pw}_1$ as a parameter
     will be ignored */
    return $b$
```

```
$\mathsf{For}_{\mathcal{F}_{\mathrm{PH}},\mathcal{A}}(1^\kappa)$
    $\mathsf{RL}.\textsc{Init}(sid,\Delta k)$
    $\mathbb{O}{\leftarrow}(\textsc{Eval},\textsc{Auth},\textsc{Update})$
    $(\Delta k',\mathsf{un},\mathsf{pw},\mathsf{st}){\leftarrow}{\mathcal{A}_1}^{\mathbb{O}}(sid)$
    $\mathsf{st}{\leftarrow}\mathcal{A}_2.\textsc{Init}(sid,\Delta k')$
    $\mathsf{st}{\leftarrow}\mathsf{AS}.\textsc{Eval}(sid,\cdot,\mathcal{A}_3(\mathsf{st}),\mathsf{un},\mathsf{pw})$
    $\mathsf{AS}.\textsc{Update}(sid,\mathcal{A},\Delta k',\mathsf{RL},\Delta k)$
    $b{\leftarrow}\mathsf{AS}.\textsc{Auth}(sid,\cdot,\mathcal{A}_4(\mathsf{st}),\mathsf{un},\mathsf{pw})$
    return $b$
```

Fig. 4: Security games of obliviousness, hiding, soundness, and forward security.

Theorem 1. *$\mathcal{F}_{\mathrm{PH}}$ defines password obliviousness. That is, for any adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\kappa)$ such that*

$$Pr[\mathit{Obl}^0_{\mathcal{F}_{\mathrm{PH}},\mathcal{A}}(1^\kappa)=1]-Pr[\mathit{Obl}^1_{\mathcal{F}_{\mathrm{PH}},\mathcal{A}}(1^\kappa)=1]{\leq}\mathit{negl}(\kappa),$$

where each probability is taken over the random coins of the experiment.

Proof. On $\textsc{Eval}(\mathsf{un},\mathsf{pw})$ and $\textsc{Auth}(\mathsf{un},\mathsf{pw})$ messages, $\mathcal{F}_{\mathrm{PH}}$ leaks information no more than $\mathsf{uid}{\leftarrow}\mathsf{prf}(\mathsf{un})$ to the adversary. The parameter pw is kept completely secret from the adversary. In the obliviousness game, the adversary can only randomly guess b. Thus, $Pr[\mathsf{Obl}^0_{\mathcal{F}_{\mathrm{PH}},\mathcal{A}}(1^\kappa)=1]-Pr[\mathsf{Obl}^1_{\mathcal{F}_{\mathrm{PH}},\mathcal{A}}(1^\kappa)=1]=0$.

Theorem 2. *$\mathcal{F}_{\mathrm{PH}}$ achieves hiding. That is, for any adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\kappa)$ such that*

$$Pr[\mathit{Hid}^0_{\mathcal{F}_{\mathrm{PH}},\mathcal{A}}(1^\kappa)=1]-Pr[\mathit{Hid}^1_{\mathcal{F}_{\mathrm{PH}},\mathcal{A}}(1^\kappa)=1]{\leq}\mathit{negl}(\kappa),$$

where each probability is taken over the random coins of the experiment.

Proof. $\mathcal{F}_{\mathrm{PH}}$ outputs random $\rho{\leftarrow}\$\{0,1\}^\ell$ for password evaluation. Even with knowledge of the evaluation value ρ of pw_b, the adversary can only randomly guess b. Direct authentication queries are not allowed. Therefore, $Pr[\mathsf{Hid}^0_{\mathcal{F}_{\mathrm{PH}},\mathcal{A}}(1^\kappa)=1]-Pr[\mathsf{Hid}^1_{\mathcal{F}_{\mathrm{PH}},\mathcal{A}}(1^\kappa)=1]=0$.

Theorem 3. *$\mathcal{F}_{\text{PH}}$ achieves soundness. That is, for any adversary $\mathcal{A}$, there exists a negligible function $\textsf{negl}(\kappa)$ such that*

$$Pr[\textsf{Sou}_{\mathcal{F}_{\text{PH}},\mathcal{A}}(1^\kappa) = 1] \leq \textsf{negl}(\kappa).$$

Proof. $\mathcal{F}_{\text{PH}}$ records $\langle \textsf{file}, \textsf{AS}, \textsf{RL}, \textsf{un}, \textsf{pw}_0 \rangle$ for the evaluation of $\textsf{pw}_0$. Only for the authentication of $\textsf{pw}_0$, $\mathcal{F}_{\text{PH}}$ outputs "accept". In the soundness game, if $\textsf{pw}_0 \neq \textsf{pw}_1$, b_1 must be 0. This means b must be 0 Thus, $Pr[\textsf{Sou}_{\mathcal{F}_{\text{PH}},\mathcal{A}}(1^\kappa) = 1] = 0$.

Theorem 4. *$\mathcal{F}_{\text{PH}}$ achieves forward security. That is, for any adversary $\mathcal{A}$, there exists a negligible function $\textsf{negl}(\kappa)$ such that*

$$Pr[\textsf{For}_{\mathcal{F}_{\text{PH}},\mathcal{A}}(1^\kappa) = 1] \leq \textsf{negl}(\kappa).$$

Proof. On the update message of $\text{UPDATE}(sid, \mathcal{A}, \Delta k', \textsf{RL}, \Delta k)$, $\mathcal{F}_{\text{PH}}$ updates the $\textsf{file}$ record $\langle \textsf{file}, \textsf{AS}, \mathcal{A}, \textsf{un}, \textsf{pw} \rangle$ to $\langle \textsf{file}, \textsf{AS}, \textsf{RL}, \textsf{un}, \textsf{pw} \rangle$. After that, $\mathcal{F}_{\text{PH}}$ will ignore the authentication queries containing $(\mathcal{A}, \textsf{un})$ in the parameter. In the forward security game, b must be 0. Therefore, $Pr[\textsf{For}_{\mathcal{F}_{\text{PH}},\mathcal{A}}(1^\kappa) = 1] = 0$.

– On input $(\text{SEND}, sid, \textsf{P}', m)$ from $\textsf{P}$, send $(\text{SEND}, sid, \textsf{P}, \textsf{P}', m)$ to $\mathcal{A}^*$, save $\langle \textsf{message}, sid, \textsf{P}, \textsf{P}', m \rangle$ and mark it PENDING.
– On $(\text{SENT}, sid, \textsf{P}, \textsf{P}')$ from $\mathcal{A}^*$, ignore this message if there is no record $\langle \textsf{message}, sid, \textsf{P}, \textsf{P}', m \rangle$ marked PENDING. Otherwise, remark the $\textsf{message}$ record COMPLETED, and output $(\text{SENT}, sid, \textsf{P}, m)$ to $\textsf{P}'$.

Fig. 5: Authenticated channel functionality $\mathcal{F}_{\text{AC}}$.

– On input $(\text{SEND}, sid, \textsf{P}', m)$ from $\textsf{P}$, send $(\text{SEND}, sid, \textsf{P}, \textsf{P}', |m|)$ to $\mathcal{A}^*$, save $\langle \textsf{message}, sid, \textsf{P}, \textsf{P}', m \rangle$ and mark it PENDING.
– On $(\text{SENT}, sid, \textsf{P}, \textsf{P}')$ from $\mathcal{A}^*$, ignore this message if there is no record $\langle \textsf{message}, sid, \textsf{P}, \textsf{P}', m \rangle$ marked PENDING. Otherwise, remark the $\textsf{message}$ record COMPLETED, and output $(\text{SENT}, sid, \textsf{P}, m)$ to $\textsf{P}'$.

Fig. 6: Authenticated and secure channel functionality $\mathcal{F}_{\text{SC}}$.

2.2 RePhoenix: A PH Realization from Phoenix [28] with a Few Straightforward yet Crucial Adjustments.

Fig. 7 shows an adaptive PH protocol named RePhoenix, an improved version of Phoenix [28] with a few straightforward yet crucial adjustments. In detail, we mainly make the following revisions, except for syntactical changes made to acclimatize to the UC definition of PH in Fig. 2 and Fig. 3:

Verifiability of password evaluation. In the evaluation phase, Phoenix [28] lacks the necessary verifiability, allowing AS to indiscriminately accept a response

Public Parameters and Components

- Group $\mathbb{G}$ of prime order q and generator g, on which the DDH problem is difficult.
- Hash functions H, H_C, H_S with range $\{0,1\}^\kappa, \mathbb{G}, \mathbb{G}$.
- NIZK schemes $(\mathsf{PoK}_1, \mathsf{Vf}_1), (\mathsf{PoK}_2, \mathsf{Vf}_2)$ depicted in Fig. 19.
- Server group $\mathcal{RL}$ s.t. $|\mathcal{RL}| = n$. $\mathsf{RL}(i)$ denotes the i-th RL server in $\mathcal{RL}$.
- An authenticated channel $\mathcal{F}_{\text{AC}}$ depicted in Fig. 5 and a secure channel $\mathcal{F}_{\text{SC}}$ depicted in Fig. 6.

Initialization (via $\mathcal{F}_{\text{AC}}$ and $\mathcal{F}_{\text{SC}}$)

1. AS picks a salt $s \leftarrow\!\!\$ \{0,1\}^\kappa$.
2. $\mathsf{RL}(1)$ picks $\hat{u}, \hat{k_S} \leftarrow\!\!\$ \mathbb{Z}_q$, sends $(\hat{u}, \hat{k_S})$ to all other $\mathsf{RL}(i)$ $(i \in [2,n])$ via $\mathcal{F}_{\text{SC}}$, and publishes $\hat{\mathsf{pk}} \leftarrow (g^{\hat{u}}, g^{\hat{k_S}})$ via $\mathcal{F}_{\text{AC}}$.
3. On input $(\textsc{Init}, sid, \Delta k)$, RL generates its secret keys $u \leftarrow \alpha \cdot \hat{u} + \beta, k_S \leftarrow \alpha \cdot \hat{k_S} + \gamma$, where $(\alpha, \beta, \gamma) \leftarrow \Delta k$, deletes $\hat{u}, \hat{k_S}$, and publishes $\mathsf{pk} \leftarrow \{g^u, g^{k_S}\}$ via $\mathcal{F}_{\text{AC}}$.

Online Evaluation

1. On input $(\textsc{Eval}, sid, esid, \mathsf{RL}, \mathsf{un}, \mathsf{pw})$, AS sends $\mathsf{uid} \leftarrow H(\mathsf{un}, s)$ to RL and saves $\langle esid, \mathsf{un}, \mathsf{pw}, \mathsf{uid} \rangle$.
2. On message uid' from AS, RL samples $n_S \leftarrow\!\!\$ \{0,1\}^\kappa$, computes $h_S \leftarrow H_S(\mathsf{uid}', n_S)$, $y \leftarrow {h_S}^{k_S}$, $\zeta \leftarrow \mathsf{PoK}_1(k_S : h_S, y)$, sends (y, ζ, n_S) to AS, records $(\mathsf{count}, \mathsf{uid}', c)$ for $c := 0$, and outputs $(\textsc{EvalResp}, sid, esid, \mathsf{uid}')$.
3. On message (y', ζ', n'_S) from RL, AS recovers $\langle esid, \mathsf{un}, \mathsf{pw}, \mathsf{uid} \rangle$, computes $h_S \leftarrow H_S(\mathsf{uid}, n'_S)$, and verifies this values via $\mathsf{Vf}_1(\mathsf{pk}_2, \zeta' : h_S, y')$.
   - If ζ' is valid, AS picks $n_C \leftarrow\!\!\$ \{0,1\}^\kappa$ and $k_C, v \leftarrow\!\!\$ \mathbb{Z}_q$, and computes $t \leftarrow g^v$ and $\rho \leftarrow {\mathsf{pk}_1}^v \cdot {h_C}^{k_C} \cdot y'$ for $h_C \leftarrow H_C(\mathsf{un}, \mathsf{pw}, n_C)$. Then AS stores $\langle \mathsf{file}, \mathsf{uid}, \mathsf{RL}, k_C, n'_S, n_C, t, \rho \rangle$ in a password file, which is indexed by uid. Finally, AS outputs $(\textsc{Eval}, sid, esid, \rho)$.
   - Else AS outputs $(\textsc{Eval}, sid, esid, \perp)$.

Online Authentication

1. On input $(\textsc{Auth}, sid, asid, \mathsf{RL}, \mathsf{un}, \mathsf{pw}')$, AS uses $\mathsf{uid} \leftarrow H(\mathsf{un}, s)$ to retrieve the corresponding file record $\langle \mathsf{file}, \mathsf{uid}, \mathsf{RL}, k_C, n_S, n_C, t, \rho \rangle$ (ignoring this message if no such record), computes $h_S \leftarrow H_S(\mathsf{uid}, n_S)$, $x_1 \leftarrow t \cdot g^z$ for $z \leftarrow\!\!\$ \mathbb{Z}_q$, $x_2 \leftarrow \rho \cdot {\mathsf{pk}_1}^z / {h_C}^{k_C}$ for $h_C \leftarrow H_C(\mathsf{un}, \mathsf{pw}', n_C)$, saves $\langle asid, \mathsf{uid}, h_S, x_1, x_2 \rangle$, sends (uid, n_S, x_2) to RL.
2. On message $(\mathsf{uid}', n'_S, x'_2)$ from AS, RL retrieves $(\mathsf{count}, \mathsf{uid}', c)$, if $c < \mathsf{limit}$, then computes $h_S \leftarrow H_S(\mathsf{uid}', n'_S)$, $y_1 \leftarrow {h_S}^{-u+k_S}$, $y_2 \leftarrow (x'_2/y_1)^{1/u}$, $\zeta_1 \leftarrow \mathsf{PoK}_2(u, k_S : {h_S}^{-1}, h_S, y_1)$ $\zeta_2 \leftarrow \mathsf{PoK}_1(u : y_2, x'_2/y_1)$, sends $(y_1, y_2, \zeta_1, \zeta_2)$ to AS, and outputs $(\textsc{AuthResp}, sid, asid, \mathsf{uid}')$.
3. On message $(y'_1, y'_2, \zeta'_1, \zeta'_2)$ from RL, AS recovers $\langle asid, \mathsf{uid}, h_S, x_1, x_2 \rangle$ and verifies NIZK proofs (ζ'_1, ζ'_2) by $\mathsf{Vf}_2(\mathsf{pk}, \zeta'_1 : 1/h_S, h_S, y'_1)$ and $\mathsf{Vf}_1(\mathsf{pk}_1, \zeta'_2 : y'_2, x_2/y'_1)$.
   - If ζ'_1 and ζ'_2 are valid, AS checks that $y'_2 = x_1 \cdot h_S$:
     - If it holds, AS outputs $(\textsc{Auth}, sid, asid, \mathsf{accept})$;
     - Else AS outputs $(\textsc{Auth}, sid, asid, \mathsf{reject})$;
   - If ζ'_1 or ζ'_2 is invalid, AS outputs $(\textsc{Auth}, sid, asid, \perp)$.

Password File Update

- On input $(\textsc{Update}, sid, \mathsf{un}, (\mathsf{RL}, \Delta k), (\mathsf{RL}', \Delta k'))$, where $(\alpha, \beta, \gamma) \leftarrow \Delta k$ and $(\alpha', \beta', \gamma') \leftarrow \Delta k'$, for the file record $\langle \mathsf{file}, \mathsf{uid}, \mathsf{RL}, k_C, n_S, n_C, t, \rho \rangle$ s.t. $\mathsf{uid} \leftarrow H(\mathsf{un}, s)$, AS picks $r \leftarrow\!\!\$ \mathbb{Z}_q$, and computes new values by $k'_C \leftarrow {k_C}^{\alpha'/\alpha}$, $t' \leftarrow t \cdot g^r$, $\rho' \leftarrow ((\rho/(t^\beta \cdot {h_S}^\gamma))^{1/\alpha} \cdot {\hat{\mathsf{pk}}_1}^{r})^{\alpha'} \cdot t'^{\beta'} \cdot {h_S}^{\gamma'}$. After that AS updates the file record to $\langle \mathsf{file}, \mathsf{uid}, \mathsf{RL}', k'_C, n_S, n_C, t', \rho' \rangle$.

Fig. 7: Adaptive PH protocol RePhoenix. For a brief format of the evaluation and authentication protocols of RePhoenix and its differences from Phoenix [28], see Fig. 16 and Fig. 17 in Section 5.

from RL to generate the password record. This may lead to dire consequences. If the response is generated by corrupt RL without according to the protocol specification or falsified by the adversary during transmission, the password record stored by AS will naturally be invalid; that is, the password record is useless to verify login passwords. In this case, the lack of evaluation verifiability harms the user because he registered a password but cannot use it to log in to his account. Lai *et al.* [27] specially define the soundness property to avoid the same problem of their password-hardened encryption scheme. In RePhoenix, we introduce non-interactive zero-knowledge (NIZK) proof for online evaluation. On response (y, ζ, n_S) from RL, AS can use the NIZK proof ζ and pk to verify that the calculation of y follows the protocol specification, i.e., $y = H_S(\mathsf{uid}, n_S)^{k_S}$. If ζ is invalid, AS outputs a null value, meaning user registration fails.

More precise authentication. In Phoenix [28], RL verifies that (x_1, x_2, x_3) satisfy the equality relations: $x_2 = {x_1}^u{\cdot}{h_S}^{k_S}$ and $x_3 = {x_1}^{s_1}{\cdot}x_2/{h_S}^{{k_S}^{s_2}}$. RL returns $\perp$ as a response if any equation is not established. In this case, AS outputs "reject" without verification, meaning that the login password is wrong. If both equations hold, RL can conclude that the login password is correct. In this case, RL provides a NIZK proof ζ for the first equation, and AS verifies ζ; only if ζ is valid, AS outputs "accept". The above response policy of Phoenix [28] has two vulnerabilities: user privacy leakage regarding login behavior and unverifiable negative results. Specifically, RL can accurately know the correctness of the login password. In other words, RL learns the privacy of user login behavior, as a third party in the PH framework. Moreover, AS cannot verify server responses (i.e., $\perp$) for wrong passwords. That is, the adversary can tamper a positive result with a null value, which AS will accept. In our improved RePhoenix, RL uses its keys (u, k_S) to compute two new response values $y_1 = (1/h_s)^u{\cdot}{h_S}^{k_S}$ and $y_2 = (x_2/y_1)^{1/u}$, and provides the corresponding NIZK proofs ζ_1, ζ_2. On the response from RL, AS checks whether ζ_1, ζ_2 are valid, and then judges whether the login password is correct according to the equation $y_2 = x_1{\cdot}h_S$. RePhoenix has an advantage in that RL learns nothing about password correctness and ensures verifiability regardless of whether the password is correct.

Simplify computation. In Phoenix [28], the two equations $x_2 = {x_1}^u{\cdot}{h_S}^{k_S}$ and $x_3 = {x_1}^{s_1}{\cdot}x_2/{h_S}^{{k_S}^{s_2}}$ essentially ascertain the same relation, the latter of which does not equip with verifiability. Therefore, our RePhoenix deletes x_3, and only applies $x_2 = {x_1}^u{\cdot}{h_S}^{k_S}$ to verify login passwords.

Improve user anonymity. In Phoenix [28], un is inputted to RL in plaintext; the username is inappropriately revealed to RL, a third party in the PH framework. In RePhoenix, user names (registered on the same authentication server) are tweaked [11] by the same salted hash value before being sent to RL. All username tweaks with the same salt preserve RL's ability to link requests (critical for per-user rate limiting) without directly leaking plaintexts. In a multi-client scenario, requests cannot be chained across clients by username tweaks. However, Phoenix's un in requests allows linkability across clients [28], posing a significant threat to user privacy.

Adapt to multi-server scenarios. In Phoenix [28], k_C held by AS is a random value common to all passwords. However, this is not feasible for our multi-server RePhoenix, because AS will recalculate $k'_C = \alpha{\cdot}k_C$ during file record updates so that inconsistent k'_C will be derived with different update rhythms. For example, $\textsc{Update}((\mathsf{RL}(i), \alpha_i), (\mathsf{RL}, \alpha))$ and $\textsc{Update}((\mathsf{RL}(j), \alpha_j), (\mathsf{RL}, \alpha))$ $(\alpha_i{\neq}\alpha_j)$ lead to inconsistent k'_C $({k_C}^{\alpha/\alpha_i}{\neq}{k_C}^{\alpha/\alpha_j})$. Therefore, RePhoenix stores a separate k_C in the password file for each password. Although k_C storage is increased here, the total storage does not increase because the previous simplified computation saves one value of space. In addition, unless k_C is specially protected, such as being stored in secure hardware (no such assumption in Phoenix [28]), it is generally considered that k_C will be leaked along with the password file leakage, so this change will not reduce security.

UC security of RePhoenix. We prove the UC security of RePhoenix by proving the following theorem in Appendix B.

Theorem 5. *RePhoenix presented in Fig. 7 UC-realizes the ideal PH functionality $\mathcal{F}_{\mathrm{PH}}$ defined in Fig. 2 and Fig. 3, assuming the existence of an authenticated channel $\mathcal{F}_{\mathrm{AC}}$ used to publish public keys and a secure channel $\mathcal{F}_{\mathrm{SC}}$ used to transmit the key base $(\hat{u}, \hat{k_S})$ in the initialization phase, and under the Decisional Diffie-Hellman (DDH) assumption in the Random Oracle Model (ROM).*

3 The Reliable PH Functionality $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ Without Single Points of Failure (SPF)

In password hardening services [11,32,28,24,27] defined in the single-server setting, the authentication server delegates the cryptographically hardening passwords to an external PH server. However, there is a concern that if the single PH server goes offline due to a network failure or server attack, these PHs cannot continue supporting password authentication. This is known as Single Points of Failure (SPF). Since the authentication server cannot verify login passwords independently, any disruption in PH due to SPF can negatively impact all users' login. Moreover, SPF has an even more significant impact in a multi-client scenario where the PH server serves multiple authentication servers. An attacker who targets one PH server can disrupt multiple authentication servers and all their numerous users. Therefore, defining an uninterrupted and reliable PH without SPF is crucial to ensure availability.

In this section, we present the reliable PH functionality $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ in Fig. 8, based on the PH functionality $\mathcal{F}_{\mathrm{PH}}$ proposed in Section 2. Recall that $\mathcal{F}_{\mathrm{PH}}$ involves two parties, namely AS denoting the authentication server and RL denoting the PH server. Following all existing PH definitions [11,32,28,24], RL is online by default in $\mathcal{F}_{\mathrm{PH}}$ without accounting for the adversary's capability to disconnect RL. By the way, the authentication sessions are started from AS, which means that AS is online in the current session. Any single-server PH is vulnerable to SPF. To deal with disconnected RL, $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ includes a group $\mathcal{RL}$ of n PH servers. Only one $\mathsf{RL} \in \mathcal{RL}$ is on duty to respond to AS's authentication requests as in

Initialization

- On input $(\textsc{Init}, sid, \mathcal{RL})$ from $\mathsf{RL} \in \mathcal{RL}$, send $(\textsc{Init}, sid, \mathsf{RL})$ to $\mathcal{A}^*$, mark RL as fresh, and set $\mathsf{tx}(\mathsf{RL}) := 0$. After receiving all $\textsc{Init}$ messages for $\mathcal{RL}$, the functionality will deal with the following messages for sid.

Key compromise and Disconnection of rate-limiters (with permission from $\mathcal{E}$)

- On $(\textsc{Compromise}, sid, \mathsf{RL})$ from $\mathcal{A}^*$ for $\mathsf{RL}{\in}\mathcal{RL}$, insert RL into $\mathsf{compromised}$ if $\mathsf{RL}{\notin}\mathsf{compromised}$.
- $\underline{\text{On } (\textsc{Disconnect}, sid, \mathsf{RL}) \text{ from } \mathcal{A}^* \text{ for } \mathsf{RL}{\in}\mathcal{RL}, \text{ insert } \mathsf{RL} \text{ into } \mathsf{offline} \text{ if } \mathsf{RL} \notin \mathsf{offline}.}$

Online Evaluation

- On input $(\textsc{Eval}, sid, esid, \mathsf{RL}, \mathsf{un}, \mathsf{pw})$ from AS $\underline{\text{s.t. } \mathsf{RL} \in (\mathcal{RL} \setminus \mathsf{offline})}$, send $(\textsc{Eval}, sid, esid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid})$ for $\mathsf{uid}{\leftarrow}\mathsf{prf}(\mathsf{un})$ to $\mathcal{A}^*$, and record $\langle esid, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$.
- On $(\textsc{EvalResp}, sid, esid, \mathsf{RL}, \mathsf{uid}^*)$ from $\mathcal{A}^*$ for honest RL, output $(\textsc{EvalResp}, sid, esid, \mathsf{uid}^*)$ to RL and record $\langle \mathsf{count}, \mathsf{uid}^*, c := 0\rangle$ for rate-limiting online authentication.
- On $(\textsc{StorePwdFile}, sid, esid, \mathsf{AS}, \mathsf{flag})$ from $\mathcal{A}^*$, ignore this message if there is no session record $\langle esid, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$.
  - If $\mathsf{flag} = \top$, record $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$, mark the file record as FRESH, and output $(\textsc{Eval}, sid, esid, \rho)$ for $\rho{\leftarrow}\$\{0,1\}^\ell$ to AS.
  - If $\mathsf{flag} = \bot$, output $(\textsc{Eval}, sid, esid, \bot)$ to AS.

Stealing Password Data and Offline Password Test

- On $(\textsc{StealPwdFile}, sid, \mathsf{AS}, \mathsf{uid})$ from $\mathcal{A}^*$:
  - If there is a file record $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$ s.t. $\mathsf{uid} = \mathsf{prf}(\mathsf{un})$, re-mark the file record COMPROMISED if it is FRESH, and send "password file stolen" to $\mathcal{A}^*$.
    - If there is a record $\langle \mathsf{offlinetest}, \mathsf{pw}\rangle$, send pw to $\mathcal{A}^*$.
  - Otherwise, send "no password file" to $\mathcal{A}^*$.
- On $(\textsc{OfflineTestPwd}, sid, \mathsf{uid}, \mathsf{pw}^*)$ from $\mathcal{A}^*$, ignore this message if there is no file record $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$ s.t. $\mathsf{uid} = \mathsf{prf}(\mathsf{un})$, or if $\mathsf{RL}{\notin}\mathsf{compromised}$ and $\underline{|\mathsf{compromised}| < t}$.
  - If the file record is FRESH, record $\langle \mathsf{offlinetest}, \mathsf{pw}\rangle$.
  - Otherwise, check that $\mathsf{pw}^* = \mathsf{pw}$.
    - If $\mathsf{pw}^* = \mathsf{pw}$, return "correct guess" to $\mathcal{A}^*$.
    - Else return "wrong guess" to $\mathcal{A}^*$.

Online Authentication

- On input $(\textsc{Auth}, sid, asid, \overline{\mathcal{RL}}, \mathsf{un}, \mathsf{pw}')$ from AS $\underline{\text{s.t. } \overline{\mathcal{RL}}{\subset}(\mathcal{RL} \setminus \mathsf{offline})}$, ignore this message if there is no file record $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$, or if neither of the following conditions is met.
  - If $\underline{\mathsf{RL} \notin \mathsf{offline} \wedge \overline{\mathcal{RL}}{=}\{\mathsf{RL}\}}$, record $\langle asid, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}'\rangle$.
  - $\underline{\text{If } \mathsf{RL} \in \mathsf{offline} \wedge |\overline{\mathcal{RL}}'| = t, \text{ modify the file record to } \langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}', \mathsf{un}, \mathsf{pw}\rangle \text{ marked}}$ $\underline{\text{FRESH, where } \mathsf{RL}' \text{ is the first server in } \overline{\mathcal{RL}}, \text{ and record } \langle asid, \mathsf{AS}, \mathsf{RL}', \mathsf{un}, \mathsf{pw}'\rangle.}$

  Then send $(\textsc{Auth}, sid, asid, \mathsf{AS}, \overline{\mathcal{RL}}, \mathsf{uid})$ for $\mathsf{uid}{\leftarrow}\mathsf{prf}(\mathsf{un})$ to $\mathcal{A}^*$.
- On $(\textsc{AuthResp}, sid, asid, \mathsf{RL}, \mathsf{uid}^*)$ from $\mathcal{A}^*$ for honest RL, output $(\textsc{AuthResp}, sid, asid, \mathsf{uid}^*)$ to RL, retrieve $\langle \mathsf{count}, \mathsf{uid}^*, c\rangle$, and if $c < \mathsf{lmt}$, do: $c\,{+}{+}$, $\mathsf{tx}(\mathsf{RL})\,{+}{+}$, and return SUCC to $\mathcal{A}^*$.
- On $(\textsc{AuthResult}, sid, asid, \mathsf{AS}, \mathsf{flag})$ from $\mathcal{A}^*$, ignore this message if there is no session record $\langle asid, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}'\rangle$ or if $\mathsf{tx}(\mathsf{RL}) = 0$. Otherwise, set $\mathsf{tx}(\mathsf{RL})\,{-}{-}$.
  - If $\mathsf{flag} = \top$, retrieve the file record $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$.
    - If $\mathsf{pw}' = \mathsf{pw}$, output $(\textsc{Auth}, sid, asid, \text{"}\mathsf{accept}\text{"})$ to AS.
    - Else output $(\textsc{Auth}, sid, asid, \text{"}\mathsf{reject}\text{"})$ to AS.
  - If $\mathsf{flag} = \bot$, send $(\textsc{Auth}, sid, asid, \bot)$ to AS.

Online Password Test

- On $(\textsc{TestPwd}, sid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid}, \mathsf{pw}^*)$ from $\mathcal{A}^*$, ignore this message if there is no $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$ marked COMPROMISED s.t. $\mathsf{uid} = \mathsf{prf}(\mathsf{un})$, or if there is no $\langle \mathsf{count}, \mathsf{uid}, c\rangle$, or if $c = \mathsf{lmt}$. Otherwise, set $c\,{+}{+}$ and check that $\mathsf{pw}^* = \mathsf{pw}$.
  - If $\mathsf{pw}^* = \mathsf{pw}$, return "correct guess" to $\mathcal{A}^*$.
  - Else return "wrong guess" to $\mathcal{A}^*$.

Fig. 8: Reliable Password Hardening service functionality $\mathcal{F}_{\mathsf{rPH}}$. Underlined text indicates the *revisions* made by $\mathcal{F}_{\mathsf{rPH}}$ in comparison to $\mathcal{F}_{\mathrm{PH}}$

$\mathcal{F}_{\mathrm{PH}}$. In case the on-duty RL goes offline, $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ allows an alternative $\mathsf{RL}' \in \mathcal{RL}$ to take over. Below we show the *revisions* made by $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ in comparison to $\mathcal{F}_{\mathrm{PH}}$.

Multiple PH servers initialization. Every $\mathsf{RL} \in \mathcal{RL}$ ($|\mathcal{RL}| = n$) requests initialization via a ($\textsc{Init}, sid, \mathcal{RL}$) message. After receiving all $\textsc{Init}$ messages, $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ completes the initialization phase.

Adaptively disconnect a PH server. $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ defines an additional adversary capability, which allows $\mathcal{A}^*$, with permission of the environment, to adaptively disconnect RL via a new $\textsc{Disconnect}$ message. This captures a PH server going offline unexpectedly due to network failure or attack. The $\mathsf{offline}$ set collects the server identifier RL of servers disconnected via a $\textsc{Disconnect}$ message.

Only one PH server is on duty for evaluation and authentication. The evaluation phase is exactly the same as originally defined in $\mathcal{F}_{\mathrm{PH}}$. AS can request online evaluation for (un, pw) via a ($\textsc{Eval}, sid, esid, \mathsf{RL}, \mathsf{un}, \mathsf{pw}$) message. If the evaluation phase is successful, the file record of $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw} \rangle$ will be saved by $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$. After that, the recorded RL in file is duty-bound to participate in the authentication sessions for ($\mathsf{un}, \mathsf{pw}'$). Specifically, for $\textsc{Auth}$ messages, if the on-duty RL is online, $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ only processes those with the RL as parameter in exactly the same way as $\mathcal{F}_{\mathrm{PH}}$. In addition, $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ ignores the $\textsc{Eval}$ and $\textsc{Auth}$ messages, which include an offline PH server as a parameter.

Failover. The most important revision is that $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ supports failover to eliminate the single points of failure of $\mathcal{F}_{\mathrm{PH}}$. When the on-duty RL is offline, AS can still request online authentication from an online RL', which will take over as the new-stage on-duty server. Specifically, when the on-duty RL is offline, a $\textsc{Auth}$ message is required to include a sub-group of t online PH servers (i.e., $\overline{\mathcal{RL}}$) instead of RL as parameter. The first server RL' in $\overline{\mathcal{RL}}$ will be designated as the new on-duty server. On receiving such a ($\textsc{Auth}, sid, asid, \overline{\mathcal{RL}}, \mathsf{un}, \mathsf{pw}'$) message, $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ requires do the following additional steps:

– Verify that the on-duty RL is actually offline.Importantly, $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ disallows rotating the on-duty server and updating the password records via the above $\textsc{Auth}$ message when the current on-duty server is indeed online. This avoids unnecessary update operations caused by malicious requests for authentication that deliberately do not specify the on-duty server.
– Modify RL in the file record to RL'. After that, RL' is regarded on-duty. There is no difference between the file record modified to contain RL' via a $\textsc{Auth}$ message and the file record evaluated with RL' via a $\textsc{Eval}$ message; both are $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}', \mathsf{un}, \mathsf{pw} \rangle$. Like $\textsc{Update}$ in $\mathcal{F}_{\mathrm{PH}}$, failover satisfies forward security.
– Re-mark the updated file record as $\textsc{fresh}$, if it is $\textsc{compromised}$. This step is necessary. Otherwise, an additional attack path will exist to compromise a password file record: (1) $\mathcal{A}^*$ compromises a password file record evaluated by RL via a $\textsc{StealPwdFile}$ message; (2) disconnects the on-duty RL via $\textsc{Disconnect}$; and (3) actives a failover to rotate to RL' via a $\textsc{Auth}$ message. Along this path, $\mathcal{A}^*$ essentially compromises a file record evaluated by RL' to activate an offline password test (if RL' is compromised) or an online password test (if RL' is not compromised).

Both security and reliability rely on the threshold assumption. In $\mathcal{F}_{\mathrm{PH}}$, the ideal adversary $\mathcal{A}^*$ can query OfflineTestPwd only after compromising the evaluation server. $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ allows $\mathcal{A}^*$ to query OfflineTestPwd in two cases: $\mathcal{A}^*$ has compromised the evaluation (i.e., on-duty) server; $\mathcal{A}^*$ has compromised any t PH servers (i.e., $|\mathsf{compromised}|\geq t$). This means that $\mathcal{A}^*$ compromising any t PH servers is equivalent to compromising the on-duty PH server. To maintain the security defined in $\mathcal{F}_{\mathrm{PH}}$, $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ requires the threshold assumption to block the second case in OfflineTestPwd. Without loss of generality, we assume that the number of compromised servers does not reach t and the number of honest and online servers is at least t [4,21,33], where (n, t) are the threshold parameters. Additionally, the threshold assumption can avoid that $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ fails to deal with Auth messages, when the on-duty server is offline and all online servers are less than t.

Based on the above revisions, $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ works to eliminate the single points of failure in $\mathcal{F}_{\mathrm{PH}}$. AS sends an Auth message pointing to t online RL' to $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ for online authentication as per the rule. After confirmation that the on-duty RL, recorded in $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}, \mathsf{un}, \mathsf{pw}\rangle$, is offline, $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ designates the first RL' as the new-stage on-duty server by updating the password file record evaluated by the old on-duty RL to $\langle \mathsf{file}, \mathsf{AS}, \mathsf{RL}', \mathsf{un}, \mathsf{pw}\rangle$. Finally, $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ responds precisely like $\mathcal{F}_{\mathrm{PH}}$ would have. This results in an uninterruptible password hardening service.

4 TF-PH: A Compiler from PH to Reliable PH

We aim to eliminate single points of failure (SPF) without sacrificing efficiency or security, resulting in reliable password hardening services. In a multi-server setting, we ensure that only one PH server is on duty while the others remain on standby. This maintains efficiency by allowing the authentication server to perform authentication protocols with only the on-duty PH server, as in a single-server setting. As for other servers, they are strictly necessary for functioning except in case of failure in the on-duty server. The threshold solution [4,21,8,18], as an example to the contrary, relies on t servers participating in (n, t)-threshold protocols. Secondly, we aim to maintain password security by ensuring that PH key leakage from other $< t$ servers cannot compromise the on-duty PH key security. This is in contrast to redundancy, where the static storage of the PH key on multiple servers increases the attack surface of the key, thus resulting in a linear increase in the risk of key leakage.

Independent keys. All PH servers hold various random and indistinguishable keys. For sure, any key compromise does not affect the others. However, an off-duty server holding a PH key different from the on-duty key used to harden the password file record cannot take over the on-duty server in case of failover. Nonetheless, the authentication server can the update password file record to match an off-duty key using a token-based update protocol (as the Update functionality defined in $\mathcal{F}_{\mathrm{PH}}$), making failover feasible.

Threshold failover (TF). PH allows the authentication server to batch update password file records locally using a compact update token [24]. Recall that we define this token-based update functionality in $\mathcal{F}_{\text{PH}}$ by providing the message interface of $(\textsc{Update}, sid, \mathsf{un}, (\mathsf{RL}, \Delta k), (\mathsf{RL}', \Delta k'))$. The adversary can also access the update oracle. However, suppose the adversary gains access to an off-duty key and the corresponding update tokens (i.e., Δk and $\Delta k'$). In that case, it can crack passwords offline after stealing the password file. In a single-server PH, the PH server can discard its registered token in the initialization phase. While in a multiple-server reliable PH, the registered token is useful for failover by rotating the on-duty server. To reduce the risk of token leakage, a PH server can split its update token among all n PH servers and reconstruct it when needed. In the event of the on-duty server failure, the authentication server reconstructs the tokens $(\Delta k, \Delta k')$ with t online PH servers cooperatively and updates the password file locally via a $\textsc{Update}$ message. After that, an online server takes over the PH service. A secure channel is necessary to distribute token shares securely and prevent the adversary from impersonating the authentication server to obtain token shares. The above approach is called the threshold failover.

- Threshold parameters (n, t) and update token-length τ, polynomial in security parameter κ.
- On input $(\textsc{Init}, sid, \Delta k, \mathcal{RL})$ from $\mathsf{RL} \in \mathcal{RL}$ s.t. $|\mathcal{RL}| = n$, send $(\textsc{Init}, sid, \mathsf{RL})$ to $\mathcal{A}^*$, and record $\langle \mathsf{token}, \mathsf{RL}, \Delta k \rangle$. After receiving all $\textsc{Init}$ messages for $\mathcal{RL}$, the functionality will deal with the following messages for sid.
- On $(\textsc{Compromise}, sid, \mathsf{RL})$ from $\mathcal{A}^*$ for $\mathsf{RL} \in \mathcal{RL}$ (with permission from $\mathcal{E}$), insert RL into $\mathsf{compromised}$ if $\mathsf{RL} \notin \mathsf{compromised}$. If $|\mathsf{compromised}| = t$, send all token records to $\mathcal{A}^*$.
- On $(\textsc{Disconnect}, sid, \mathsf{RL})$ from $\mathcal{A}^*$ for $\mathsf{RL} \in \mathcal{RL}$ (with permission from $\mathcal{E}$), insert RL into $\mathsf{offline}$ if $\mathsf{RL} \notin \mathsf{offline}$.
- On input $(\textsc{Record}, sid, \mathsf{un}, \mathsf{RL})$ from AS, record $\langle \mathsf{onduty}, sid, \mathsf{un}, \mathsf{RL} \rangle$.
- On input $(\textsc{Query}, sid, \mathsf{un}, \overline{\mathcal{RL}})$ from AS s.t. $\overline{\mathcal{RL}} \subset (\mathcal{RL} \setminus \mathsf{offline})$, retrieve $\langle \mathsf{onduty}, sid, \mathsf{un}, \mathsf{RL} \rangle$. RL' is the first server in $\overline{\mathcal{RL}}$.
  - If $\mathsf{RL} \notin \mathsf{offline}$ and $\overline{\mathcal{RL}} = \{\mathsf{RL}\}$, output $(\textsc{Onduty}, sid, \mathsf{un}, \mathsf{RL})$ to AS.
  - If $\mathsf{RL} \in \mathsf{offline}$ and $|\overline{\mathcal{RL}}| = t$, send $(\textsc{Rotate}, sid, \mathsf{AS}, \overline{\mathcal{RL}})$ to $\mathcal{A}^*$.
    1. On $(\textsc{RotateResp}, sid, \mathsf{RL}'')$ from $\mathcal{A}^*$ for all $\mathsf{RL}'' \in \overline{\mathcal{RL}}$, retrieve $\langle \mathsf{token}, \mathsf{RL}, \Delta k \rangle$ and $\langle \mathsf{token}, \mathsf{RL}', \Delta k' \rangle$, and output $(\textsc{Update}, sid, \mathsf{un}, \Delta k, \Delta k')$ to AS.
    2. For every record $\langle \mathsf{onduty}, sid, \cdot, \mathsf{RL} \rangle$, modify it into $\langle \mathsf{onduty}, sid, \cdot, \mathsf{RL}' \rangle$.
    3. Finally, output $(\textsc{Onduty}, sid, \mathsf{un}, \mathsf{RL}')$ to AS.

Fig. 9: Threshold failover functionality $\mathcal{F}_{\mathsf{TF}}$.

4.1 The Threshold Failover Functionality $\mathcal{F}_{\text{TF}}$ and a Realization

We define the threshold failover functionality $\mathcal{F}_{\text{TF}}$ in Fig. 9 in the UC security framework. $\mathcal{F}_{\text{TF}}$ consists of three main phases: initialization, record, and query. In the initialization phase, every RL in $\mathcal{RL}$ registers its token by sending a $(\textsc{Init}, sid, \Delta k, \mathcal{RL})$ message. $\mathcal{F}_{\text{TF}}$ records the corresponding relationship between RL and Δk in $\langle \mathsf{token}, \mathsf{RL}, \Delta k \rangle$. Via a $\textsc{Record}$ message, AS specifies the on-duty server of un. $\mathcal{F}_{\mathrm{TF}}$ records the corresponding relationship between RL and un in $\langle \mathsf{onduty}, sid, \mathsf{un}, \mathsf{RL} \rangle$. In the query phase, AS can query the on-duty server for un. If the recorded on-duty RL is online, on receiving a QUERY message with $\{\mathsf{RL}\}$ as parameter, $\mathcal{F}_{\mathrm{TF}}$ returns RL. If the recorded on-duty RL is online, the QUERY message needs to include $\overline{\mathcal{RL}}$ as a parameter, where $\overline{\mathcal{RL}}$ denotes t online servers specified by AS to assist in threshold failover. By default, the first server RL' in $\overline{\mathcal{RL}}$ is the new on-duty server selected by AS. Only after receiving responses from all $\mathsf{RL}'' \in \overline{\mathcal{RL}}$ (forwarded by $\mathcal{A}^*$), $\mathcal{F}_{\mathrm{TF}}$ outputs the tokens of the old and new on-duty servers to AS for supporting its token-based update. In addition, $\mathcal{F}_{\mathrm{TF}}$ rotates the on-duty server of un from RL to RL' by modifying the onduty record in to $\langle \mathsf{onduty}, sid, \mathsf{un}, \mathsf{RL}' \rangle$, and returns the new on-duty RL' to AS. As for the ideal adversary $\mathcal{A}^*$, with permission from the environment $\mathcal{E}$, it can obtain all tokens by compromising t PH servers in $\mathcal{RL}$. It also can disconnect a PH server via a DISCONNECT message.

```
$\underline{\mathsf{GenShare}(n, t, s)}$
for $i \in [N]$ :
    $m_0 \leftarrow s_i, m_1, \cdots, m_{t-1} \leftarrow \$ \mathbb{Z}_q$
    $f_{s_i}(x) = m_0 \cdot x^0 + \cdots + m_{t-1} \cdot x^{t-1}$
    $[\![s_i]\!] \leftarrow \{f_{s_i}(1), \cdots, f_{s_i}(n)\}$
$[\![s]\!] \leftarrow \{([\![s_1]\!]_1, \cdots, [\![s_N]\!]_1),$
        $\cdots,$
        $([\![s_1]\!]_n, \cdots, [\![s_N]\!]_n)\}$
return $[\![s]\!]$
```

```
$\underline{\mathsf{ReSecret}(n, t, \{(j, [\![s]\!]_j)\}_{j \in D}), |D| = t}$
$\lambda_j(x) = \prod_{d \in D \setminus \{j\}} (x - d)/(j - d)$
$([\![s_1]\!]_j, \cdots, [\![s_N]\!]_j) \leftarrow [\![s]\!]_j, j \in D$
for $i \in [N]$ :
    $f_{s_i}(x) \leftarrow \sum_{j \in D} [\![s_i]\!]_j \cdot \lambda_j(x)$
    $s_i \leftarrow f_{s_i}(0)$
$s \leftarrow (s_1, \cdots, s_N)$
return $s$
```

Fig. 10: Secret share algorithm [33] for $s := (s_1, \cdots, s_N)$, $(s_1, \cdots, s_N \in \mathbb{Z}_q)$.

Shamir's secret sharing [33]. The Shamir's secret sharing [33] (SSS) splits and reconstructs a secret s based on the Lagrange interpolation polynomial over finite fields. When sharing s, it first generates a polynomial $f(x)$ of degree $t-1$, and $f(i)$ is a secret share and stored by a server S_i for $i \in [n]$. A colluding server set D of less than t has no advantage over an outsider attacker in guessing the secret s with their knowledge of secret shares $s_i, i \in D$. In other words, partial $(<t)$ secret shares can not infer the secret s. Therefore, secret sharing effectively improves the confidentiality of statically stored secrets. In addition, SSS enhances the availability of the secret storage since as long as any t-of-n servers participate honestly in the secret reconstruction process, the secret s can be reconstructed. Precisely, SSS consists of two algorithms, $[\![s]\!] \leftarrow \mathsf{GenShare}(n, t, s)$ and $s \leftarrow \mathsf{ReSecret}(n, t, \{j, s_j\}_{j \in D})$, with details shown in Fig. 10.

A Shamir's secret sharing-based realization of $\mathcal{F}_{\mathsf{TF}}$. In Fig. 11, we show a straightforward realization in the $\mathcal{F}_{\mathsf{sc}}$-hybrid world, named TF. In the

- On input $(\textsc{Init}, sid, \Delta k, \mathcal{RL})$, RL splits its token by $[\![\Delta k]\!] \leftarrow \mathsf{GenShare}(n, t, \Delta k)$, and for $i \in [n]$, sends $(\textsc{Send}, sid||\mathsf{RL}||\mathsf{RL}(i), \mathsf{RL}(i), [\![\Delta k]\!]_i)$ to $\mathcal{F}_{\text{SC}}$, where $\mathsf{RL}(i)$ denotes the i-th server in $\mathcal{RL}$.
- On $(\textsc{Sent}, sid||\mathsf{RL}||\mathsf{RL}(i), \mathsf{RL}, [\![\Delta k]\!]_i)$ from $\mathcal{F}_{\text{SC}}$, $\mathsf{RL}(i)$ stores $(\mathsf{RL}, [\![\Delta k]\!]_i)$.
- On input $(\textsc{Record}, sid, \mathsf{un}, \mathsf{RL})$, AS stores $(\mathsf{un}, \mathsf{RL})$, which denoting that the un-corresponding password registration is evaluated by RL.
- On input $(\textsc{Query}, sid, \mathsf{un}, \overline{\mathcal{RL}})$, AS retrieves $(\mathsf{un}, \mathsf{RL})$.
  - If RL is online and $\overline{\mathcal{RL}} = \{\mathsf{RL}\}$, AS outputs $(\textsc{Onduty}, sid, \mathsf{un}, \mathsf{RL})$ to AS.
  - If RL is offline and $|\overline{\mathcal{RL}}|{=}t$, do:
    1. AS reconstructs the tokens of RL and RL', where RL' is the first server in $\overline{\mathcal{RL}}$, by the following steps:
       (a) AS sends $(\textsc{Send}, sid||\mathsf{AS}||\mathsf{RL}(i), \mathsf{RL}(i), (\mathsf{RL}, \mathsf{RL}'))$ to $\mathcal{F}_{\text{SC}}$ for each $\mathsf{RL}(i) \in \overline{\mathcal{RL}}$.
       (b) On $(\textsc{Sent}, sid||\mathsf{AS}||\mathsf{RL}(i), \mathsf{AS}, (\mathsf{RL}, \mathsf{RL}'))$ from $\mathcal{F}_{\text{SC}}$, $\mathsf{RL}(i)$ retrieves records $(\mathsf{RL}, [\![\Delta k]\!]_i)$ and $(\mathsf{RL}', [\![\Delta k']\!]_i)$, and sends $(\textsc{Send}, sid||\mathsf{RL}(i)||\mathsf{AS}, \mathsf{AS}, (i, [\![\Delta k]\!]_i, [\![\Delta k']\!]_i))$ to $\mathcal{F}_{\text{SC}}$.
       (c) On t messages $(\textsc{Sent}, sid||\mathsf{RL}(i)||\mathsf{AS}, \mathsf{RL}(i), (i, [\![\Delta k]\!]_i, [\![\Delta k']\!]_i))$ from $\mathcal{F}_{\text{SC}}$ for all $\mathsf{RL}(i) \in \overline{\mathcal{RL}}$, AS reconstructs the tokens of RL and RL' by $\Delta k \leftarrow \mathsf{ReSecret}(n, t, \{(i, [\![\Delta k]\!]_i)\})$, $\Delta k' \leftarrow \mathsf{ReSecret}(n, t, \{(i, [\![\Delta k']\!]_i)\})$, and outputs $(\textsc{Update}, sid, \mathsf{un}, \Delta k, \Delta k')$ to AS.
    2. For every record $(\mathsf{un}, \mathsf{RL})$, AS modifies it into $(\mathsf{un}, \mathsf{RL}')$.
    3. Finally, AS outputs $(\textsc{Onduty}, sid, \mathsf{un}, \mathsf{RL}')$.

Fig. 11: Threshold failover protocol TF in the $\mathcal{F}_{\mathsf{sc}}$-hybrid world.

initialization phase, every PH server splits the registered token into n shares by $[\![\Delta k]\!] \leftarrow \mathsf{GenShare}(n, t, \Delta k)$, and distributes them to n servers in $\mathcal{RL}$ via a secure channel $\mathcal{F}_{\mathsf{sc}}$. During threshold failover from RL to RL', AS collects the token shares of RL and RL' from t servers in $\overline{\mathcal{RL}}$ via a secure channel $\mathcal{F}_{\mathsf{sc}}$, and reconstructs the tokens by $\Delta k \leftarrow \mathsf{ReSecret}(n, t, \{j, [\![\Delta k_j\}), \Delta k' \leftarrow \mathsf{ReSecret}(n, t, \{j, [\![\Delta k'_j\})$.

There is a simulator, a pass-through machine only forwards messages between the adversary $\mathcal{A}$ and the functionality $\mathcal{F}_{\mathsf{TF}}$. The environment cannot distinguish the views provided by $\mathcal{F}_{\mathsf{TF}}$ and the simulator in the ideal world and the views provided by the TF protocol and the adversary in the real world. Therefore, the TF protocol shown in Fig. 11 UC-realizes the ideal threshold failover functionality defined in Fig. 9.

4.2 Definition of the compiler TF-PH

In Fig. 12, we propose a compiler called TF-PH that transforms any PH into a reliable PH, which can resist SPF using threshold failover. In the previous sub-section, we formalize threshold failover in $\mathcal{F}_{\mathsf{TF}}$ and provide a TF realization by introducing Shamir's secret sharing [33].

As shown in Fig. 12, we describe our TF-PH protocol based on the PH functionality $\mathcal{F}_{\text{PH}}$ and the threshold failover functionality $\mathcal{F}_{\text{TF}}$. In most cases, TF-PH calls $\mathcal{F}_{\text{PH}}$ to handle and respond to messages, including all evaluation and the authentication that specifies the online on-duty server. In the case of

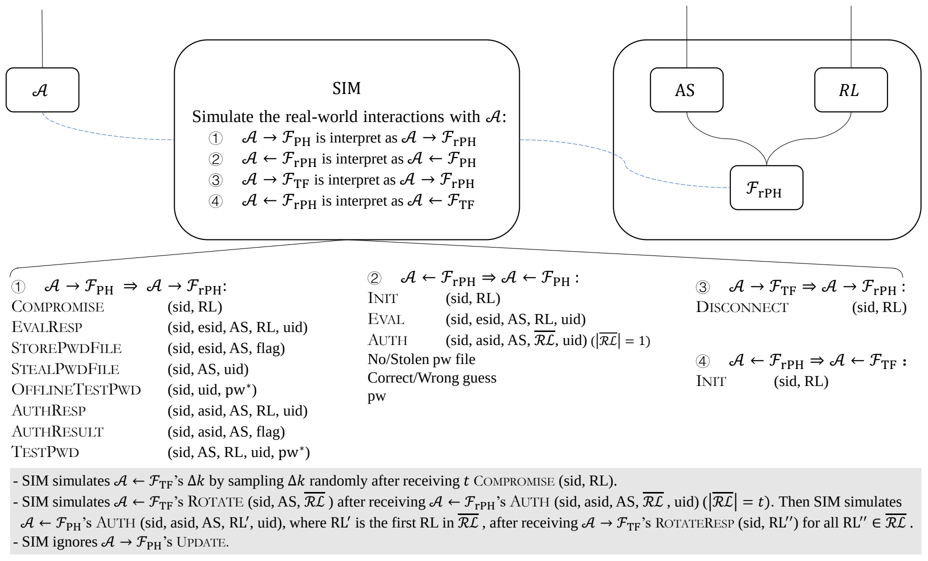
- Threshold parameters (n, t), output-length ℓ and update token-length τ, polynomial in security parameter κ.
- On input $(\textsc{Init}, sid, \mathcal{RL})$ to $\mathsf{RL} \in \mathcal{RL}$, RL samples $\Delta k \leftarrow\$ \{0,1\}^\tau$, and sends $(\textsc{Init}, sid, \Delta k)$ to $\mathcal{F}_{\mathsf{PH}}$ and $(\textsc{Init}, sid, \Delta k, \mathcal{RL})$ to $\mathcal{F}_{\mathsf{TF}}$.
- On input $(\textsc{Eval}, sid, esid, \mathsf{RL}, \mathsf{un}, \mathsf{pw})$, AS sends $(\textsc{Eval}, sid, esid, \mathsf{RL}, \mathsf{un}, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{PH}}$. On $\mathcal{F}_{\mathsf{PH}}$'s output $(\textsc{Eval}, sid, esid, \rho/\bot)$, AS outputs it. If the output is the former $\rho \in \{0,1\}^\ell$, AS sends $(\textsc{Record}, sid, \mathsf{un}, \mathsf{RL})$ to $\mathcal{F}_{\mathsf{TF}}$.
- On input $(\textsc{Auth}, sid, asid, \overline{\mathcal{RL}}, \mathsf{un}, \mathsf{pw}')$, AS sends $(\textsc{Query}, sid, \mathsf{un}, \overline{\mathcal{RL}})$ to $\mathcal{F}_{\mathsf{TF}}$.
  1. On $\mathcal{F}_{\mathsf{TF}}$'s output $(\textsc{Update}, sid, \mathsf{un}, \Delta k, \Delta k')$, AS forwards it to $\mathcal{F}_{\mathsf{PH}}$.
  2. On $\mathcal{F}_{\mathsf{TF}}$'s output $(\textsc{Onduty}, sid, \mathsf{un}, \mathsf{RL})$, AS sends $(\textsc{Auth}, sid, asid, \mathsf{RL}, \mathsf{un}, \mathsf{pw}')$ to $\mathcal{F}_{\mathsf{PH}}$. On $\mathcal{F}_{\mathsf{PH}}$'s output $(\textsc{Auth}, sid, asid, \mathsf{accept/reject}/\bot)$, AS outputs it.
- Additional adversarial behaviors:
  1. On $(\textsc{Eval}, sid, esid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid})$ from $\mathcal{F}_{\mathsf{PH}}$, $\mathcal{A}$ records $\langle \mathsf{onduty}, \mathsf{uid}, \mathsf{RL} \rangle$.
  2. On $(\textsc{Compromise}, sid, \mathsf{RL})$ for $\mathsf{RL} \in \mathcal{RL}$, $\mathcal{A}$ inserts RL into $\mathsf{compromised}$.
  3. If $|\mathsf{compromised}| = t$, $\mathcal{A}$ awaits $\langle \mathsf{token}, \mathsf{RL}, \Delta k \rangle$ from $\mathcal{F}_{\mathsf{TF}}$ for all $\mathsf{RL} \in \mathcal{RL}$, and set $\mathsf{broken} := 1$.
  4. Before sending $(\textsc{OfflineTestPwd}, sid, \mathsf{uid}, \mathsf{pw}^*)$ to $\mathcal{F}_{\mathsf{PH}}$, if $\mathsf{broken} = 1$, $\mathcal{A}$ retrieves $\langle \mathsf{onduty}, \mathsf{uid}, \mathsf{RL} \rangle$, and sends $(\textsc{Update}, sid, \mathsf{uid}, (\mathsf{RL}, \Delta k), (\mathsf{RL}', \Delta k'))$ to $\mathcal{F}_{\mathsf{PH}}$, where $\mathsf{RL}' \leftarrow\$ \mathsf{compromised}$.

Fig. 12: Reliable PH protocol in the $(\mathcal{F}_{\mathsf{PH}}, \mathcal{F}_{\mathsf{TF}})$-hybrid world.

- Threshold parameter (n, t) and update token-length τ, polynomial in security parameter κ.
- On $(\textsc{Init}, sid, \mathsf{RL})$ from $\mathcal{F}$, pass it to $\mathcal{A}$ as $\mathcal{F}_{\mathrm{PH}}$ and $\mathcal{F}_{\mathrm{TF}}$'s message to $\mathcal{A}$.
- On $(\textsc{Compromise}, sid, \mathsf{RL})$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathrm{PH}}$ and $\mathcal{F}_{\mathrm{TF}}$, pass it to $\mathcal{F}$. Sim inserts RL into $\mathsf{compromised}$. If $|\mathsf{compromised}| = 1$, Sim picks n $\Delta k \leftarrow\$ \{0,1\}^\tau$ and sends them to $\mathcal{A}$ as $\mathcal{F}_{\mathrm{TF}}$'s message to $\mathcal{A}$.
- On $(\textsc{Disconnect}, sid, \mathsf{RL})$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathrm{TF}}$, pass it to $\mathcal{F}$.
- On $(\textsc{Eval}, sid, esid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid})$ from $\mathcal{F}$, pass it to $\mathcal{A}$ as $\mathcal{F}_{\mathrm{PH}}$'s message to $\mathcal{A}$.
- On $(\textsc{EvalResp}, sid, esid, \mathsf{RL}, \mathsf{uid})$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathrm{PH}}$, pass it to $\mathcal{F}$.
- On $(\textsc{StorePwdFile}, sid, esid, \mathsf{AS}, \mathsf{flag})$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathrm{PH}}$, pass it to $\mathcal{F}$.
- On $(\textsc{StealPwdFile}, sid, \mathsf{AS}, \mathsf{uid})$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathrm{PH}}$, pass it to $\mathcal{F}$. On $\mathcal{F}$'s response, pass it to $\mathcal{A}$ as $\mathcal{F}_{\mathrm{PH}}$'s response.
- On $(\textsc{OfflineTestPwd}, sid, \mathsf{AS}, \mathsf{uid}, \mathsf{pw}^*)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathrm{PH}}$, pass it to $\mathcal{F}$. On $\mathcal{F}$'s response, pass it to $\mathcal{A}$ as $\mathcal{F}_{\mathrm{PH}}$'s response.
- On $(\textsc{Auth}, sid, asid, \mathsf{AS}, \overline{\mathcal{RL}}, \mathsf{uid})$ from $\mathcal{F}$:
  - If $|\overline{\mathcal{RL}}| = t$, send $(\textsc{Rotate}, sid, \mathsf{AS}, \overline{\mathcal{RL}})$ to $\mathcal{A}$ as $\mathcal{F}_{\mathrm{TF}}$'s response, and await $(\textsc{RotateResp}, sid, \mathsf{RL}'')$ from $\mathcal{A}^*$ aimed at $\mathcal{F}_{\mathrm{TF}}$ for all $\mathsf{RL}'' \in \overline{\mathcal{RL}}$.

  Send $(\textsc{Auth}, sid, asid, \mathsf{AS}, \mathsf{RL}', \mathsf{uid})$ to $\mathcal{A}$ as $\mathcal{F}_{\mathrm{PH}}$'s message to $\mathcal{A}$, where RL' is the first server of $\overline{\mathcal{RL}}$.
- On $(\textsc{AuthResp}, sid, asid, \mathsf{RL}, \mathsf{uid})$ from $\mathcal{F}$, pass it to $\mathcal{A}$ as $\mathcal{F}_{\mathrm{PH}}$'s message to $\mathcal{A}$.
- On $(\textsc{AuthResult}, sid, asid, \mathsf{AS}, \mathsf{flag})$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathrm{PH}}$, pass it to $\mathcal{F}$ as $\mathcal{F}_{\mathrm{PH}}$'s response.
- On $(\textsc{TestPwd}, sid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid}, \mathsf{pw}^*)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathrm{PH}}$, pass it to $\mathcal{F}$. On $\mathcal{F}$'s response, pass it to $\mathcal{A}$ as $\mathcal{F}_{\mathrm{PH}}$'s response.

Fig. 13: Simulator Sim for the TF-PHS protocol in Fig. 12. Sim is a pass-through machine, except when processing $\textsc{Auth}$ messages with $|\overline{\mathcal{RL}}| = t$ (which means a threshold rotation after the on-duty server is offline). ($\mathcal{F}$ denotes $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$).



Simulate the real-world interactions with $\mathcal{A}$:

① $\mathcal{A} \rightarrow \mathcal{F}_{\mathrm{PH}}$ is interpret as $\mathcal{A} \rightarrow \mathcal{F}_{\mathrm{rPH}}$

② $\mathcal{A} \leftarrow \mathcal{F}_{\mathrm{rPH}}$ is interpret as $\mathcal{A} \leftarrow \mathcal{F}_{\mathrm{PH}}$

③ $\mathcal{A} \rightarrow \mathcal{F}_{\mathrm{TF}}$ is interpret as $\mathcal{A} \rightarrow \mathcal{F}_{\mathrm{rPH}}$

④ $\mathcal{A} \leftarrow \mathcal{F}_{\mathrm{rPH}}$ is interpret as $\mathcal{A} \leftarrow \mathcal{F}_{\mathrm{TF}}$

① $\mathcal{A} \rightarrow \mathcal{F}_{\mathrm{PH}} \Rightarrow \mathcal{A} \rightarrow \mathcal{F}_{\mathrm{rPH}}$:

COMPROMISE	(sid, RL)
EVALRESP	(sid, esid, AS, RL, uid)
STOREPWDFILE	(sid, esid, AS, flag)
STEALPWDFILE	(sid, AS, uid)
OFFLINETESTPWD	(sid, uid, pw*)
AUTHRESP	(sid, asid, AS, RL, uid)
AUTHRESULT	(sid, asid, AS, flag)
TESTPWD	(sid, AS, RL, uid, pw*)

② $\mathcal{A} \leftarrow \mathcal{F}_{\mathrm{rPH}} \Rightarrow \mathcal{A} \leftarrow \mathcal{F}_{\mathrm{PH}}$:

INIT	(sid, RL)
EVAL	(sid, esid, AS, RL, uid)
AUTH	(sid, asid, AS, $\overline{\mathcal{RL}}$, uid) ($\lvert\overline{\mathcal{RL}}\rvert = 1$)
No/Stolen pw file	
Correct/Wrong guess	
pw	

③ $\mathcal{A} \rightarrow \mathcal{F}_{\mathrm{TF}} \Rightarrow \mathcal{A} \rightarrow \mathcal{F}_{\mathrm{rPH}}$:

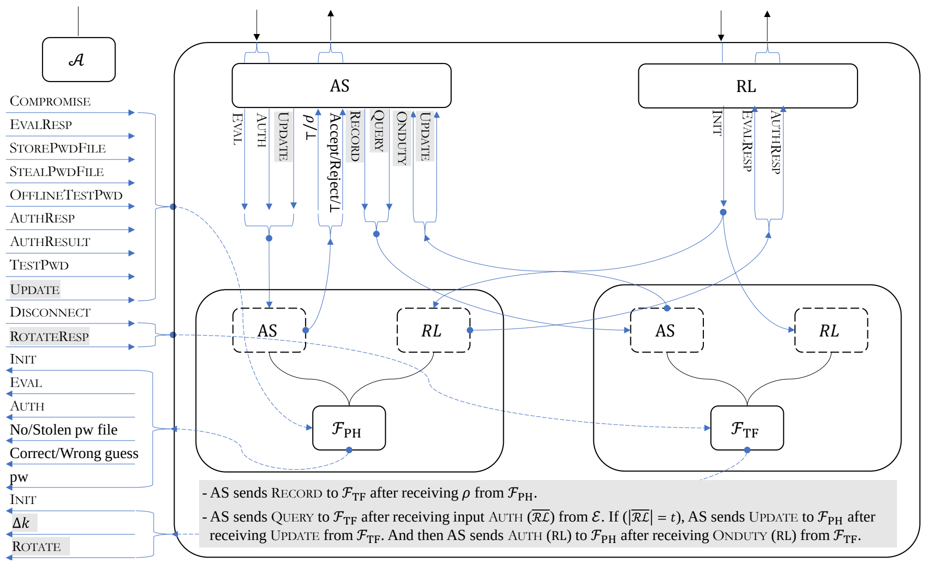
DISCONNECT	(sid, RL)

④ $\mathcal{A} \leftarrow \mathcal{F}_{\mathrm{rPH}} \Rightarrow \mathcal{A} \leftarrow \mathcal{F}_{\mathrm{TF}}$:

INIT	(sid, RL)

- SIM simulates $\mathcal{A} \leftarrow \mathcal{F}_{\mathrm{TF}}$'s Δk by sampling Δk randomly after receiving t COMPROMISE (sid, RL).
- SIM simulates $\mathcal{A} \leftarrow \mathcal{F}_{\mathrm{TF}}$'s ROTATE (sid, AS, $\overline{\mathcal{RL}}$) after receiving $\mathcal{A} \leftarrow \mathcal{F}_{\mathrm{rPH}}$'s AUTH (sid, asid, AS, $\overline{\mathcal{RL}}$, uid) ($\lvert\overline{\mathcal{RL}}\rvert = t$). Then SIM simulates $\mathcal{A} \leftarrow \mathcal{F}_{\mathrm{PH}}$'s AUTH (sid, asid, AS, RL′, uid), where RL′ is the first RL in $\overline{\mathcal{RL}}$, after receiving $\mathcal{A} \rightarrow \mathcal{F}_{\mathrm{TF}}$'s ROTATERESP (sid, RL″) for all RL″ $\in \overline{\mathcal{RL}}$.
- SIM ignores $\mathcal{A} \rightarrow \mathcal{F}_{\mathrm{PH}}$'s UPDATE.

(a) Ideal world.



- AS sends RECORD to $\mathcal{F}_{\mathrm{TF}}$ after receiving ρ from $\mathcal{F}_{\mathrm{PH}}$.
- AS sends QUERY to $\mathcal{F}_{\mathrm{TF}}$ after receiving input AUTH ($\overline{\mathcal{RL}}$) from $\mathcal{E}$. If ($\lvert\overline{\mathcal{RL}}\rvert = t$), AS sends UPDATE to $\mathcal{F}_{\mathrm{PH}}$ after receiving UPDATE from $\mathcal{F}_{\mathrm{TF}}$. And then AS sends AUTH (RL) to $\mathcal{F}_{\mathrm{PH}}$ after receiving ONDUTY (RL) from $\mathcal{F}_{\mathrm{TF}}$.

(b) Real world.

Fig. 14: Interactions between components in the ideal and real worlds. In the ideal world, the simulator Sim in Fig. 13 provides the same view to the adversary as in the real world. Except for simulation content in the shaded text, Sim operates as a pass-through machine. In the real world, AS/RL always processes all inputs and sub-routine outputs by calling the message interfaces of $\mathcal{F}_{\mathsf{PH}}$ and $\mathcal{F}_{\mathsf{TF}}$. Except for the shaded text, AS/RL forwards the environment input directly to the subroutine. An immediate conclusion is that the ideal world and the real world are indistinguishable to the environment.

the on-duty server going offline, TF-PH calls $\mathcal{F}_{\mathrm{TF}}$ to rotate to an online server, and then calls $\mathcal{F}_{\mathrm{PH}}$ to continue responding to the authentication messages.

Recall that, in $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$, the adversary $\mathcal{A}^*$ is allowed to carry out an offline password test via OfflineTestPwd, after compromising the on-duty server or any t server via Compromise (with the permission from $\mathcal{E}$). However, in $\mathcal{F}_{\mathrm{PH}}$, only when the on-duty server is compromised, $\mathcal{A}^*$ can test password offline via OfflineTestPwd. The adversary to TF-PH fills the above gap through the following attack behavior: (1) $\mathcal{A}^*$ learns the on-duty server RL from the $(\textsc{Eval}, sid, esid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid})$ message that is from $\mathcal{F}_{\mathrm{PH}}$; (2) $\mathcal{A}^*$ learns all tokens from $\mathcal{F}_{\mathrm{TF}}$ after compromising any t server except RL (with the permission from $\mathcal{E}$); and (3) $\mathcal{A}^*$ sends $(\textsc{OfflineTestPwd}, sid, \mathsf{uid}, \mathsf{pw}^*)$ to $\mathcal{F}_{\mathrm{PH}}$, following $(\textsc{Update}, sid, \mathsf{uid}, (\mathsf{RL}, \Delta k), (\mathsf{RL}', \Delta k'))$, where RL' is one of the compromised servers. After that $\mathcal{F}_{\mathrm{PH}}$ will process this OfflineTestPwd message as $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$.

4.3 Security Proof of TF-PH

We show that TF-PH eliminates single points of failure (SPF) of $\mathcal{F}_{\mathrm{PH}}$ by proving Theorem 6. Based on the UC theorem [5], TF-PH can be used as a compiler to develop a reliable PH protocol without SPF from any PH protocol that realizes $\mathcal{F}_{\mathrm{PH}}$. See in Section 5 for a concrete TF-PH protocol with $\mathcal{F}_{\mathrm{PH}} \rightarrow$ RePhoenix and $\mathcal{F}_{\mathrm{TF}} \rightarrow$ TF, named TF-RePhoenix.

Theorem 6. *The TF-PH protocol presented in Fig. 12 UC-realizes the reliable PH functionality $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ defined in Fig. 8.*

Proof. We construct a simulator Sim, as shown in Fig. 13. Without loss of generality, we assume that $\mathcal{A}$ is a dummy adversary (i.e., a pass-through machine that outsources all messages and computations to the environment $\mathcal{E}$).

Fig. 14 diagrams the interactions that occur in the real and simulated worlds. We now show that, the environment $\mathcal{E}$ cannot distinguish between these two worlds. The argument uses only a single game change from the real world **G0** to the simulated world **G1**. By $\mathsf{Dist}_{\mathcal{E}}^{G_0,G_1}$, we denote distinguisher $\mathcal{E}$'s distinguishing advantage between **G0** and **G1**. Specifically,

$$\mathsf{Dist}_{\mathcal{E}}^{G_0,G_1} = |Pr^{G_0}[\mathcal{E} \text{ output } 1] - Pr^{G_1}[\mathcal{E} \text{ output } 1]|$$

.

Game G_0: The real world. The distinguisher $\mathcal{E}$ interacts with TF-PH (Fig. 12) in the role of the honest parties (AS and RL) and the role of the adversary.

Game G_1: The simulated world. By inspection, Sim in interaction with $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$ behaves identically to the real-world TF-PH protocol, except that Δk sent to the adversary are different and that ρ output to AS are different. However, due to Δk being sampled from $\{0,1\}^\tau$ and ρ being sampled from $\{0,1\}^\ell$ in the two worlds, $\mathcal{E}$ cannot distinguish which world they are from.

Therefore, TF-PH UC-realize $\mathcal{F}_{\mathsf{r}\mathrm{PH}}$.

5 TF-RePhoenix: A Concrete Reliable PH

Fig. 15-Fig. 17 includes a concrete reliable PHS protocol called TF-RePhoenix, which is an instantiation of the $(\mathcal{F}_{\mathrm{PH}}, \mathcal{F}_{\mathrm{TF}})$-hybrid-world TF-PHS. TF-RePhoenix instantiates the $\mathcal{F}_{\mathrm{PH}}$ functionality with our UC-secure RePhoenix and the $\mathcal{F}_{\mathrm{TF}}$ functionality with our UC-secure TF.

Assumption on server compromise. We consider the adaptive corruption of the authentication server (AS) and PH servers RL. At any time after initialization, the adversary can compromise RL via a Compromise message, or steal a password file record from AS via a StealPwdFile. For all PH protocols, the resistance to offline password guessing attacks relies on a basic assumption [11] that AS and RL will not be compromised simultaneously. Otherwise, any PH will degrade into a trivial case that is vulnerable to offline password guessing upon server compromise. For our reliable PH, we need to assume that AS and the on-duty RL will not be compromised simultaneously.

For threshold assumption, we assume that at most $t-1$ PH servers are compromised and at least t PH servers are functioning correctly. In this way, there are always t PH servers ready for threshold failover, thus ensuring reliability.

Assumption on channels. TF-RePhoenix includes $\mathcal{F}_{\mathrm{AC}}$ (Fig. 5) and $\mathcal{F}_{\mathrm{SC}}$ (Fig. 6), which define the authenticated and secure channels, respectively. Public keys go through an authenticated channel $\mathcal{F}_{\mathrm{AC}}$, meaning that $\mathcal{A}^*$ can obtain but cannot tamper with them. This has the same meaning as public key publishing in practice. The key base is transmitted on a secure channel $\mathcal{F}_{\mathrm{SC}}$. In addition, all token shares are distributed and collected via a secure channel $\mathcal{F}_{\mathrm{SC}}$. Note that the reliance on a secure channel for token transmission is called for all existing PH works [11,32,28,27,4,24].

Security of the key base. At the beginning of initialization, the first RL generates the key base $\hat{\mathsf{sk}}$ and sends it to all other servers via a secure channel $\mathcal{F}_{\mathrm{SC}}$. At the end step of initialization, every server deletes the key base. We know that our functionality definitions receive $(\textsc{Compromise}, sid, \mathsf{RL})$ message only after initializing RL. All servers are honest in the initialization phase. Therefore, the key is not accessible to the adversary until deleted.

Security of the update token. The update token is secret-shared among n PH servers via a secure channel $\mathcal{F}_{\mathrm{SC}}$ in the initialization phase. After that, for compromising the update token the adversary needs to compromise t PH servers. During failover, honest AS is allowed to collect token shares from PH servers via a secure channel $\mathcal{F}_{\mathrm{SC}}$. For AS, the channel's and protocol's security are independent. Even after sending a StealPwdFile message to steal a password file record successfully, the adversary cannot break the secure channel to impersonate AS and obtain token shares from RL.

Correctness of threshold failover. Intuitively, the new password record should be equivalent to the one evaluated by the new on-duty server $\mathsf{RL}(i')$ from scratch. As shown in Fig. 16, the old password record consists of $(\mathsf{uid}, \mathsf{RL}(i), k_C, n_S, n_C, t, \rho)$, where $t = g^v, \rho = (\mathsf{pk}_1)^v \cdot h_C^{k_C} \cdot h_S^{k_S}$, hardened by the old on-duty server $\mathsf{RL}(i)$. During threshold failover, AS (i.e., the authentication server) first re-

```
**Initialization** (Secure channel)
$\underline{\mathsf{AS}(\kappa, n, t)}$
$s \leftarrow\!\$ \{0,1\}^{\kappa}$
return $s, \hat{\mathsf{pk}}, \{(i, \mathsf{pk})\}, i \in [n]$
$\underline{\mathsf{RL}(i, \kappa, n, t, \mathcal{RL}), i \in [n]}$
if $i = 1$ :
  $\hat{u}, \hat{k}_S \leftarrow\!\$ \mathbb{Z}_q$
  $\mathsf{RL}(i) \xrightarrow[\text{SC}]{\hat{u}, \hat{k}_S} \mathsf{RL}(j) \in \mathcal{RL}, j \in [2, n]$
  publish $\hat{\mathsf{pk}} \leftarrow (g^{\hat{u}}, g^{\hat{k_S}})$
else :
  $\mathsf{RL}(i) \xleftarrow[\text{SC}]{\hat{u}, \hat{k}_S} \mathsf{RL}(1)$
$\alpha, \beta, \gamma \leftarrow\!\$ \mathbb{Z}_q, \Delta k_i \leftarrow (\alpha, \beta, \gamma)$
$[\![\Delta k_i]\!] \leftarrow \mathsf{GenShare}(n, t, \Delta k_i)$
$\mathsf{RL}(i) \xrightarrow[\text{SC}]{[\![\Delta k_i]\!]_j} \mathsf{RL}(j), j \in [n] \setminus \{i\}$
$\mathsf{RL}(i) \xleftarrow[\text{SC}]{[\![\Delta k_j]\!]_i} \mathsf{RL}(j), j \in [n] \setminus \{i\}$
$u \leftarrow \alpha \cdot \hat{u} + \beta, k_S := \alpha \cdot \hat{k_S} + \gamma$
$\mathsf{sk} \leftarrow (u, k_S)$
publish $\mathsf{pk} \leftarrow (g^{u}, g^{k_S})$
delete $\hat{u}, \hat{k}_S$
return $\mathsf{sk}, \{(j, [\![\Delta k_j]\!]_i)\}, j \in [n]$
```

```
**Threshold Rotation** (Secure channel)
$\underline{\mathsf{AS}(n, t, i, \overline{\mathcal{RL}}), |\overline{\mathcal{RL}}| = t}$
# $\mathsf{RL}(i)$ is offline
# $\mathsf{RL}(i')$ is the first server in $\overline{\mathcal{RL}}$
$\mathsf{AS} \xrightarrow[\text{SC}]{i, i'}$ all $\mathsf{RL}(j) \in \overline{\mathcal{RL}}$
$\mathsf{AS} \xleftarrow[\text{SC}]{[\![\Delta k_i]\!]_j, [\![\Delta k_{i'}]\!]_j}$ all $\mathsf{RL}(j) \in \overline{\mathcal{RL}}$
$[\![\Delta k_i]\!] \leftarrow \mathsf{ReSecret}(n, t, \{j, [\![\Delta k_i]\!]_j\})$
$[\![\Delta k'_i]\!] \leftarrow \mathsf{ReSecret}(n, t, \{j, [\![\Delta k'_i]\!]_j\})$
for each $(\mathsf{uid}, \mathsf{RL}(i), k_C, n_S, n_C, t, \rho)$ :
  $h_S \leftarrow H_S(\mathsf{uid}, n_S)$
  $r \leftarrow\!\$ \mathbb{Z}_q$
  $t' \leftarrow t \cdot g^{r}$
  $\rho' \leftarrow ((\rho / (t^{\beta_i} \cdot {h_S}^{\gamma_i}))^{1/\alpha_i} \cdot {\hat{\mathsf{pk}}_1}^{r})^{\alpha_{i'}} \cdot {t'}^{\beta_{i'}} \cdot {h_S}^{\gamma_{i'}}$
  $k'_C \leftarrow \alpha'_i / \alpha_i \cdot k_C$
  update $(\mathsf{uid}, \mathsf{RL}(i'), k'_C, n_S, n_C, t', \rho')$
```

Fig. 15: Concrete initialization protocol and threshold rotation protocol.

constructs the update tokens of $\mathsf{RL}(i)$ and $\mathsf{RL}(i')$: $\Delta k_i = (\alpha_i, \beta_i, \gamma_i), \Delta k' = (\alpha'_i, \beta'_i, \gamma'_i)$. Then, AS uses $\Delta k_i, \Delta k_{i'}$ to update t, ρ to t', ρ' as follows:

$$\begin{aligned}
t' &= t \cdot g^{r} = g^{v+r} = g^{v'};\\
\rho' &= ((\rho / (t^{\beta_i} \cdot {h_S}^{\gamma_i}))^{1/\alpha_i} \cdot {\hat{\mathsf{pk}}_1}^{r})^{\alpha_{i'}} \cdot {t'}^{\beta_{i'}} \cdot {h_S}^{\gamma_{i'}}\\
&= (\hat{\rho} \cdot \hat{\mathsf{pk}}_1^{\,v-\hat{v}} \cdot \hat{\mathsf{pk}}_1^{\,r})^{\alpha_{i'}} \cdot {t'}^{\beta_{i'}} \cdot {h_S}^{\gamma_{i'}}\\
&= (\hat{\rho} \cdot \hat{\mathsf{pk}}_1^{\,v+r-\hat{v}})^{\alpha_{i'}} \cdot {t'}^{\beta_{i'}} \cdot {h_S}^{\gamma_{i'}}\\
&= g^{(\alpha_{i'} \cdot \hat{u} + \beta_{i'}) \cdot (v+r)} \cdot h_C^{\alpha_{i'} \cdot \hat{k_C}} \cdot h_S^{\alpha_{i'} \cdot \hat{k}_S + \gamma_{i'}}\\
&= (\mathsf{pk}'_1)^{v'} \cdot h_C^{k'_C} \cdot h_S^{k'_S}, k'_C = \alpha_{i'} \cdot \hat{k_C}.
\end{aligned}$$

The updated ρ' are hardened by the keys of $u' = \alpha_{i'} \cdot \hat{u} + \beta_{i'}, k'_S = \alpha_{i'} \cdot \hat{k_S} + \gamma_{i'}$. This demonstrates the correctness of the record update in threshold failover.

Evaluation

$\mathsf{AS}(s, \mathsf{pk}, \mathsf{un}, \mathsf{pw})$		$\mathsf{RL}(\mathsf{sk})$
$v, k_C \leftarrow\!\$\, \mathbb{Z}_q, n_C \leftarrow\!\$\, \{0,1\}^{\kappa}$		$n_S \leftarrow\!\$\, \{0,1\}^{\kappa}$
$\mathsf{uid} \leftarrow H(\mathsf{un}, s)$	$\xrightarrow{\mathsf{uid}}$	$h_S \leftarrow H_S(\mathsf{uid}, n_S)$
$h_C \leftarrow H_C(\mathsf{un}, \mathsf{pw}, n_C)$	$\xrightarrow{\mathsf{un}}$	$h_S \leftarrow H_S(\mathsf{un}, n_S)$ /* Leak un */
$t \leftarrow g^{v}$		
$t_3 \leftarrow \mathsf{pk}_2{}^{v}$ /* Redundant t_3 */		$y \leftarrow h_S{}^{k_S}$
if $\mathsf{Vf}_1(\mathsf{pk}_2, \zeta : H_S(\mathsf{uid}, n_S), y)$:	$\xleftarrow{y, \zeta, n_S}$	$\zeta \leftarrow \mathsf{PoK}_1(k_S : h_S, y)$
$\quad \rho \leftarrow \mathsf{pk}_1{}^{v} \cdot h_C{}^{k_C} \cdot y$	$\xleftarrow{y, n_S}$	/* Unverifiable y */
$\quad$ delete $v, y, \mathsf{un}, \mathsf{pw}$		
$\quad$ store $(\mathsf{uid}, \mathsf{RL}, k_C, n_S, n_C, t, t_3, \rho)$		

Fig. 16: Concrete evaluation protocol. The commented text in gray indicates the modified part of Phoenix [28] to solve the problem in /* parentheses */.

Improved failover. The new on-duty server RL' performs a key rotation before threshold failover. In detail, RL' requests its token shares from t online servers via a secure channel and reconstructs its token $\Delta k = (\alpha, \beta, \gamma)$; RL' picks α', β', γ' randomly and generates new keys $u = \alpha' \cdot u + \beta, k_S = \alpha' \cdot k_S + \gamma$; then AS shares the new token $\Delta k' = (\alpha' \cdot \alpha, \alpha' \cdot \beta + \beta', \alpha' \cdot \gamma + \gamma')$ via a secure channel. The key rotation prevents the failover from becoming a vulnerability that the adversary can exploit by taking the current on-duty server offline and attempting to make a compromised server be selected to take over password authentication.

Key rotation protocol. We provide a key rotation protocol by which we can restore protocol security as appropriate (e.g., regularly or after a certain number of server compromises). We do not define the key rotation functionality because the old token is not saved by RL in $\mathcal{F}_{\mathrm{PH}}$, but it is necessary for updating the password file record. We will leave this issue to our future work.

It is worth noting that the key base also needs to be updated during the key rotation process. Otherwise, the key rotation protocol cannot fully restore security when t PH servers have been compromised. In this case, the adversary can calculate the key base with a compromised PH key and its corresponding token. Suppose the key base is not updated, even after rotating all compromised PH keys. In that case, the adversary can compromise any server to calculate its token, without having to compromise at least t PH servers.

Additionally, a key rotation should render an old on-duty PH key useless to the adversary for learning password information from the updated password record. Following Phoenix [28], we consider a forward security experiment with a leakage function $\mathcal{L}$ and prove that the update in the key rotation is $\mathcal{L}$-forward secure (detailed in Section C), which states that the rotated keys and the updated password records are indistinguishable from freshly generated ones.

Authentication

$\mathsf{AS}(s, \mathsf{pk}, \mathsf{un}, \mathsf{pw}')$		$\mathsf{RL}(\mathsf{sk})$
$\mathsf{uid} \leftarrow H(\mathsf{un}, s)$		
retrieve $(\mathsf{uid}, \mathsf{RL}, k_C, n_S, n_C, t, t_3, \rho)$		
$h'_C \leftarrow H_C(\mathsf{un}, \mathsf{pw}', n_C)$		
$z \leftarrow \$ \mathbb{Z}_q, x_1 \leftarrow t \cdot g^z$		
$x_2 \leftarrow \rho \cdot (\mathsf{pk}_1)^z / {h'_C}^{k_C}$	$\xrightarrow{\mathsf{uid}, n_S, x_2}$	$h_S \leftarrow H_S(\mathsf{uid}, n_S)$
$x_3 \leftarrow t_3 \cdot \mathsf{pk}_2{}^z$	$\xrightarrow{\mathsf{un}, n_S, x_1, x_2, x_3}$	$h_S \leftarrow H_S(\mathsf{un}, n_S)$ /* Leak un */
		$y_1 \leftarrow (1/h_S)^u \cdot h_S^{k_S}$
		$y_2 \leftarrow (x_2/y_1)^{1/u}$
		$\zeta_1 \leftarrow \mathsf{PoK}_2(u, k_S : h_S^{-1}, h_S, y_1)$
if $\mathsf{Vf}_2(\mathsf{pk}, \zeta_1 : h_S^{-1}, h_S, y_1)$	$\xleftarrow{y_1, y_2, \zeta_1, \zeta_2}$	$\zeta_2 \leftarrow \mathsf{PoK}_1(u : y_2, x_2/y_1)$
$\wedge\ \mathsf{Vf}_1(\mathsf{pk}_1, \zeta_2 : y_2, x_2/y_1)$:		
if $y_2 = x_1 \cdot h_S$:		
return "accept"		
else return "reject"		**if** $x_2 = x_1^u \cdot h_S \wedge x_3 = x_1{}^{s_1} \cdot (x_2/h_s)^{s_2}$:
if ζ :	$\xleftarrow{\zeta}$	/* Leak pw′'s correctness */
if $\mathsf{Vf}_2(\mathsf{pk}, \zeta : x_1, h_S, x_2)$		$\zeta \leftarrow \mathsf{PoK}_2(u, k_S : x_1, h_S, x_2)$
return "accept"		
else return "reject"/* Unverifiable "reject" */		

Fig. 17: Concrete authentication protocol. The commented text in gray indicates the modified part of Phoenix [28] to solve the problem in /* parentheses */.

Discuss the trade-offs between availability, security, and efficiency. To improve PH, we find a new balance between eliminating SPF and maintaining efficiency, without prioritizing the incidental benefit of elevating protocol security to the threshold standard. In traditional password authentication schemes without PH, users get responses after one round of interaction with the authentication server ($\mathsf{U} \rightleftharpoons \mathsf{AS}$). However, in the PH-introduced password authentication schemes, two rounds of three-party communication ($\mathsf{U} \rightleftharpoons \mathsf{AS} \rightleftharpoons \mathsf{RL}$) are necessary, which inevitably results in extended wait times for users, especially when PH servers are deployed off-site [11,28,24]. If the SPF of PH is solved at the expense of increased latency, too long waiting may cause user frustration. Therefore, efficiency is our priority criterion for solutions to SPF. Our TF-RePhoenix deals with SPF via a fast failover. As analyzed above, it performs optimally without any efficiency loss compared to RePhoenix. (See the performance evaluation and comparison in Appendix D.)

In comparison, the threshold solution [4] guarantees its resistance to SPF if any t-of-n servers are honest. It additionally enhances security by establishing a

Key Rotation (Secure channel)

$\mathsf{AS}(n, t, s, \hat{\mathsf{pk}}, \{(i, \mathsf{pk})\}_{i\in[n]})$		$\mathsf{RL}(i, \mathsf{sk}, \{\llbracket \Delta k_j \rrbracket_i\}_{j\in[n]}), i \in [n]$
$v, \alpha, \beta, \gamma \leftarrow\$ \mathbb{Z}_q$	$\xrightarrow[\text{SC}]{\alpha, \beta, \gamma}$	
$\hat{\mathsf{pk}}' \leftarrow (\hat{\mathsf{pk}}_1{}^{\alpha} \cdot g^{\beta}, \hat{\mathsf{pk}}_2{}^{\alpha} \cdot g^{\gamma})$		$\alpha'_i, \beta'_i, \gamma'_i \leftarrow\$ \mathbb{Z}_q$
$\mathsf{AS} \xleftarrow[\text{SC}]{j, \llbracket \Delta k_i \rrbracket_j} \mathsf{RL}(j), j \in [n]$		$(u, k_S) \leftarrow \mathsf{sk}$
for $i \in [n]$:	$\xleftarrow[\text{SC}]{\alpha'_i, \beta'_i, \gamma'_i}$	$u' \leftarrow (\alpha \cdot \alpha'_i) \cdot u + (\alpha'_i \cdot \beta + \beta'_i)$
$\quad \Delta k_i \leftarrow \mathsf{ReSecret}(n, t, \{(j, \llbracket \Delta k_i \rrbracket_j)\})$		$k_S{}' \leftarrow (\alpha \cdot \alpha'_i) \cdot k_S + (\alpha'_i \cdot \gamma + \gamma'_i)$
$\quad (\alpha_i, \beta_i, \gamma_i) \leftarrow \Delta k_i$		$\mathsf{sk}' \leftarrow (u', k_S{}')$
$\quad$ for each $(\mathsf{uid}, \mathsf{RL}(i), k_C, n_S, n_C, t, \rho)$:		publish $\mathsf{pk}' \leftarrow (g^{u'}, g^{k_S{}'})$
$\qquad h_S \leftarrow H_S(\mathsf{uid}, n_S)$		$\Delta k_i{}' \leftarrow (\alpha'_i, \beta'_i, \gamma'_i)$
$\qquad \hat{\rho} \leftarrow (\rho / (t^{\beta_i} \cdot h_S{}^{\gamma_i}))^{\alpha/\alpha_i} \cdot t^{\beta} \cdot h_S{}^{\gamma}$		$\llbracket \Delta k_i{}' \rrbracket \leftarrow \mathsf{GenShare}(n, t, \Delta k_i{}')$
$\qquad t' \leftarrow t \cdot g^{v}$		$\mathsf{RL} \xrightarrow[\text{SC}]{\llbracket \Delta k_i{}' \rrbracket_j} \mathsf{RL}(j), j \in [n] \setminus \{i\}$
$\qquad \rho' \leftarrow (\hat{\rho} \cdot \mathsf{pk}_1{}^{v})^{\alpha'_i} \cdot t'^{\beta'_i} \cdot h_S{}^{\gamma'_i}$		$\mathsf{RL} \xleftarrow[\text{SC}]{\llbracket \Delta k_j{}' \rrbracket_i} \mathsf{RL}(j), j \in [n] \setminus \{i\}$
$\qquad k'_C \leftarrow \alpha'_i \cdot (\alpha/\alpha_i) \cdot k_C$		return $\mathsf{sk}', \{(j, \llbracket \Delta k_j{}' \rrbracket_i)\}_{j\in[n]}$
$\qquad$ update $(\mathsf{uid}, \mathsf{RL}(i), k_C, n_S, n_C, t', \rho')$		
return $\hat{\mathsf{pk}}', \{(i, \mathsf{pk}')\}_{i\in[n]}$		

Fig. 18: Key rotation protocol.

threshold security standard, which ensures that the adversary must corrupt, at a minimum, t-of-n servers to compromise key/protocol security. Nevertheless, the threshold solution poses certain drawbacks to efficiency. For example, T-PHE [4] has a higher latency compared to the original PHE [27], i.e., 2x for encryption and 3x for the decryption protocol (even in the same setting of $t = n = 1$). And its latency linearly increases with t. As for the redundancy solution, it reduces security since multiple static storage of the PH key may increase the risk of key leakage.

5.1 Proof of Security

Theorem 7. *The TF-RePhoenix protocol from Fig. 15-Fig. 17 UC-realizes the reliable PH functionality $\mathcal{F}_{\mathsf{r}\text{PH}}$ defined by Fig. 8.*

Proof. Given that TF-PH$^{\mathcal{F}_{\text{PH}}, \mathcal{F}_{\text{TF}}}$ (i.e. the $(\mathcal{F}_{\text{PH}}, \mathcal{F}_{\text{TF}})$-hybrid-world TF-PH from Fig. 12) UC-realizes the reliable PH functionality $\mathcal{F}_{\mathsf{r}\text{PH}}$, RePhoenix presented in Fig. 7 UC-realizes the PH functionality $\mathcal{F}_{\text{PH}}$, TF presented in Fig. 11 UC-realizes the TF functionality $\mathcal{F}_{\text{TF}}$, we can conclude that our TF-PHS$^{\mathcal{F}_{\text{PH}} \rightarrow \text{RePhoenix}, \mathcal{F}_{\text{TF}} \rightarrow \text{TF}}$ (i.e. TF-RePhoenix shown in Fig. 15 and Fig. 17) UC-realizes $\mathcal{F}_{\mathsf{r}\text{PH}}$, according to the UC theorem [5].

References

1. Agrawal, S., Miao, P., Mohassel, P., Mukherjee, P.: PASTA: password-based threshold authentication. In: ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 2042–2059 (2018)
2. Arapakis, I., Bai, X., Cambazoglu, B.B.: Impact of response latency on user behavior in web search. In: International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR). p. 103–112 (2014)
3. Borokhovich, M., Schiff, L., Schmid, S.: Provable data plane connectivity with local fast failover: Introducing openflow graph algorithms. In: Hot topics in software defined networking (HotSDN). pp. 121–126 (2014)
4. Brost, J., Egger, C., Lai, R.W., Schmid, F., Schröder, D., Zoppelt, M.: Threshold password-hardened encryption services. In: ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 409–424 (2020)
5. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: IEEE Symposium on Foundations of Computer Science (FOCS). pp. 136–145 (2001)
6. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. In: Advances in Cryptology–EUROCRYPT. pp. 404–421 (2005)
7. D. Goodin: Anatomy of a hack: How crackers ransack passwords like "qeadzcwrsfxv1331" (5 2013), http://arstechnica.com/security/2013/05/how-crackers-make-minced-meat-out-of-your- passwords/
8. Das, P., Hesse, J., Lehmann, A.: DPaSE: Distributed password-authenticated symmetric-key encryption, or how to get many keys from one password. In: Proc. ACM AsiaCCS 2022. pp. 682–696
9. Diomedous, C., Athanasopoulos, E.: Practical password hardening based on tls. In: Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA). pp. 441–460 (2019)
10. Enterprise, V.: 2023 Data Breach Investigations Report, https://www.verizon.com/business/resources/reports/2023-data- breach-investigations-report-dbir.pdf
11. Everspaugh, A., Chaterjee, R., Scott, S., Juels, A., Ristenpart, T.: The Pythia PRF service. In: USENIX Security Symposium (USENIX Security). pp. 547–562 (2015)
12. Freeman, D., Jain, S., Dürmuth, M., Biggio, B., Giacinto, G.: Who are you? a statistical approach to measuring user authenticity. In: Symposium on Network and Distributed System Security (NDSS). vol. 16, pp. 21–24 (2016)
13. Gentry, C., MacKenzie, P., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Advances in Cryptology – CRYPTO. pp. 142–159 (2006)
14. Goodin, D.: 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/
15. Goodin, D.: 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/
16. Grassi, P.A., Fenton, J.L., Newton, E.M., Perlner, R.A., Regenscheid, A.R., Burr, W.E., Richer, J.P., et al.: NIST 800-63B digital identity guidelines: Authentication and lifecycle management. Tech. rep., National Institute of Standards and Technology (2017)

17. Griffiths, K.: The falcon web framework (2022), https://falcon.readthedocs.io/en/stable/
18. Gu, Y., Jarecki, S., Kedzior, P., Nazarian, P., Xu, J.: Threshold pake with security against compromise of all servers. In: Advances in Cryptology – ASIACRYPT (2024)
19. Jarecki, S., Kiayias, A., Krawczyk, H.: Round-optimal password-protected secret sharing and t-pake in the password-only model. In: Advances in Cryptology – ASIACRYPT. pp. 233–253 (2014)
20. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In: IEEE European Symposium on Security and Privacy (EuroS&P). pp. 276–291 (2016)
21. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: TOPPSS: Cost-minimal password-protected secret sharing based on threshold oprf. In: Applied Cryptography and Network Security (ACNS). pp. 39–58 (2017)
22. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In: Advances in Cryptology – EUROCRYPT. pp. 456–486 (2018)
23. Jarecki, Stanislaw and Krawczyk, Hugo and Xu, Jiayu: On the (In) security of the Diffie-Hellman oblivious PRF with multiplicative blinding. In: Public-Key Cryptography – PKC. pp. 380–409 (2021)
24. Jia, C., Wu, S., Wang, D.: Reliable password hardening service with opt-out. In: Symposium on Reliable Distributed Systems (SRDS). pp. 250–261 (2022)
25. Johns Hopkins University ISI: Charm-crypto docs (2022), https://jhuisi.github.io/charm/index.html
26. Kiner, E., April, T.: Google mitigated the largest DDoS attack to date, peaking above 398 million rps (Oct 2023), https://cloud.google.com/blog/products/identity-security/google-cloud-mitigated-largest-ddos-attack-peaking-above-398-million-rps
27. Lai, R.W., Egger, C., Reinert, M., Chow, S.S., Maffei, M., Schröder, D.: Simple password-hardened encryption services. In: USENIX Security Symposium (USENIX Security). pp. 1405–1421 (2018)
28. Lai, R.W., Egger, C., Schröder, D., Chow, S.S.: Phoenix: Rebirth of a cryptographic password-hardening service. In: USENIX Security Symposium (USENIX Security). pp. 899–916 (2017)
29. Muffett, A.: Facebook password hashing and authentication., https://www.youtube.com/watch?v=7dPRFoKteIU
30. Pal, B., Daniel, T., Chatterjee, R., Ristenpart, T.: Beyond credential stuffing: Password similarity models using neural networks. In: IEEE Symposium on Security and Privacy (S&P). pp. 417–434 (2019)
31. Saeed, N.: Top Insights From Our 2022 State of Secure Identity Report (Sep 2022), https://auth0.com/blog/top-insights-from-our-2022-state-of-secure-identity-report/
32. Schneider, J., Fleischhacker, N., Schröder, D., Backes, M.: Efficient cryptographic password hardening services from partially oblivious commitments. In: ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 1192–1203 (2016)
33. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)

34. Shea, R., Wong, V., Rusnak, P., Halfmoon Labs: Secret Sharing: A library for sharding and sharing secrets (like Bitcoin private keys), using shamir's secret sharing scheme (2016), https://github.com/shea256/secret-sharing.git
35. Stephens, B., Cox, A.L.: Deadlock-free local fast failover for arbitrary data center networks. In: IEEE International Conference on Computer Communications (INFOCOM). pp. 1–9 (2016)
36. Tyagi, N., Celi, S., Ristenpart, T., Sullivan, N., Tessaro, S., Wood, C.A.: A fast and simple partially oblivious PRF, with applications. In: Advances in Cryptology – EUROCRYPT. pp. 674–705 (2022)
37. UK National Cyber Security Centre: Password policy: Updating your approach (Nov 2018), https://www.ncsc.gov.uk/collection/passwords/updating-your-approach
38. Wang, D., Zou, Y., Zhang, Z., Xiu, K.: Password guessing using random forest. In: USENIX Security Symposium (USENIX Security) (2023)
39. Yang, R., Zhang, Y., Garraghan, P., Feng, Y., Ouyang, J., Xu, J., Zhang, Z., Li, C.: Reliable computing service in massive-scale systems through rapid low-cost failover. IEEE Transactions on Services Computing (TSC) **10**(6), 969–983 (2016)
40. Zhang, Y., Xu, C., Li, H., Yang, K., Cheng, N., Shen, X.: PROTECT: efficient password-based threshold single-sign-on authentication for mobile users against perpetual leakage. IEEE Transactions on Mobile Computing **20**(6), 2297–2312 (2020)

Universally Composable Password Hardening Service Against Single Points of Failure (The Supplementary Material)

A Preliminaries

NIZK proof: $y = x^k$ [11]

```
$\mathsf{PoK}_1(k : x, y)$
$v \leftarrow \$ \mathbb{Z}_q, a_1 \leftarrow g^v, a_2 \leftarrow x^v$
$h \leftarrow H_{\mathsf{NIZK}_1}(g, g^k, x, v, a_1, a_2)$
$w \leftarrow v - h \cdot k$
return $\zeta \leftarrow (h, w)$
```

```
$\mathsf{Vf}_1(g^k, \zeta : x, y)$
$(h, w) \leftarrow \zeta$
$a'_1 \leftarrow g^w \cdot (g^k)^h, a'_2 \leftarrow x^w \cdot y^h$
$h' \leftarrow H_{\mathsf{NIZK}_1}(g, g^k, x, v, a'_1, a'_2)$
return $b \leftarrow (h = h')$
```

NIZK proof: $y = {x_1}^{k_1} \cdot {x_2}^{k_2}$ [28]

```
$\mathsf{PoK}_2(k_1, k_2 : x_1, x_2, y)$
$r_1, r_2 \leftarrow \$ \mathbb{Z}_q$
$g_1 \leftarrow g^{r_1}, g_2 \leftarrow g^{r_2}$
$x'_1 \leftarrow x_1^{r_1}, x'_2 \leftarrow x_2^{r_2}$
$c \leftarrow H_{\mathsf{NIZK}_2}(g, g^{k_1}, g^{k_2}, x_1, x_2, y, x'_1, x'_2)$
$k'_1 \leftarrow r_1 + c \cdot k_1, k'_2 \leftarrow r_2 + c \cdot k_2$
return $\zeta \leftarrow (x'_1, x'_2, k'_1, k'_2)$
```

```
$\mathsf{Vf}_2(g^{k_1}, g^{k_2}, \zeta : x_1, x_2, y)$
$(x'_1, x'_2, k'_1, k'_2) \leftarrow \zeta$
$c \leftarrow H_{\mathsf{NIZK}_2}(g, g^{k_1}, g^{k_2}, x_1, x_2, y, x'_1, x'_2)$
$b_1 \leftarrow (g^{k'_1} = g_1 \cdot (g^{k_1})^c)$
$b_2 \leftarrow (g^{k'_2} = g_2 \cdot (g^{k_2})^c)$
$b_3 \leftarrow ({x_1}^{k'_1} \cdot {x_2}^{k'_2} = x'_1 \cdot x'_2 \cdot y^c)$
return $b_1 \wedge b_2 \wedge b_3$
```

Fig. 19: Concrete Non-Interactive Zero-Knowledge (NIZK) algorithms for proving $y = x^k$ and $y = {x_1}^{k_1} \cdot {x_2}^{k_2}$.

B Proof of Theorem 5

We present a PHS protocol, RePhoenix, that improved over Phoenix [28] mainly in terms of verifiability, efficiency, and user anonymity (preserving linkability), in Section 2.2. We state the UC security of our RePhoenix in Theorem 5. In this section, we prove it by proving that for any adversary against RePhoenix, there is a simulated adversary that produces a view in the ideal world that no environment $\mathcal{E}$ can distinguish with an advantage better than $2 \cdot (n_e/q + n_a/q^2)$ where q is the order of $\mathbb{Z}_q$ and n_e, n_a respectively are the request numbers to EVAL and AUTH, under the DDH assumption in ROM with $H(\cdot), H_S(\cdot), H_{\mathsf{NIZK}_1}(\cdot), H_{\mathsf{NIZK}_2}(\cdot)$ modeled as random oracles.

- Security parameter κ. Group $\mathbb{G}$ of prime order q and generator g.
- NIZK schemes $(\mathsf{PoK}_1, \mathsf{Vf}_1)$ and $(\mathsf{PoK}_2, \mathsf{Vf}_2)$.
- Collision-resistant hash functions H, H_C with range $\{0,1\}^{2\kappa}, \mathbb{G}$.
- Sim defines a random sequence by $r_1, \cdots, r_N \leftarrow\$ \mathbb{Z}_q$, $g_1 := g^{r_1}, \cdots, g_N := g^{r_N}$, and initiates $I = J = D = 1$. Sim maintains a table for the random oracle $H_S(\cdot)$. For a new query (uid, n_S) s.t. $\mathsf{uid} \in \{0,1\}^{2\kappa}$ and $n_S \in \{0,1\}^{\kappa}$, Sim answers with g^{r_I}, records (uid, n_S, r_I) in table T_{H_S}, and sets $I++$. For an old query $((\mathsf{uid}, n_S, r_{\hat{I}}) \in T_{H_S})$, answer with the $g^{r_{\hat{I}}}$.

Initialization
- On $(\textsc{Init}, sid, \mathsf{RL})$ from $\mathcal{F}$, pick RL's secret keys $u, k_S \leftarrow\$ \mathbb{Z}_q$, record $\langle \mathsf{RL}, u, k_S \rangle$.

Server Compromise
- On $(\textsc{Compromise}, sid, \mathsf{RL})$ from $\mathcal{A}$, forward it to $\mathcal{F}$, and send (u, k_S) to $\mathcal{A}$.

Online Evaluation
- On $(\textsc{Eval}, sid, esid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid})$ from $\mathcal{F}$, send $(sid, esid, \mathsf{uid})$ to $\mathcal{A}$ as AS's message to RL, and record $\langle esid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid} \rangle$.
- On $(sid, esid, \mathsf{uid}')$ from $\mathcal{A}$, send $(\textsc{EvalResp}, sid, esid, \mathsf{uid}')$ to $\mathcal{F}$, sample $n_S \leftarrow\$ \{0,1\}^{\kappa}$, assign $h_S \leftarrow H_S(\mathsf{uid}', n_S)$, computes $y \leftarrow {h_S}^{k_S}$, $\zeta \leftarrow \mathsf{PoK}_1(k_S : h_S, y)$, and send $(sid, esid, y, \zeta, n_S)$ to $\mathcal{A}$ as RL's message to AS.
- On $(sid, esid, y', \zeta', n'_S)$ from $\mathcal{A}$, recover $\langle esid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid} \rangle$ and $\langle \mathsf{RL}, u, k_S \rangle$, verify $\mathsf{Vf}_1(g^{k_S}, \zeta' : h_S, y')$ for $h_S \leftarrow H_S(\mathsf{uid}, n'_S)$.
  - If ζ' is valid:
    - If $y' = {h_S}^{k_S}$, send $(\textsc{StorePwdFile}, sid, esid, \mathsf{AS}, \top)$ to $\mathcal{F}$, and record $\langle \mathsf{AS}, \mathsf{uid}, n'_S, r_{\hat{I}} \rangle$, where $(\mathsf{uid}, n'_S, r_{\hat{I}}) \in T_{H_S}$.
    - Else, <u>Sim aborts</u>.
  - If ζ' is invalid, send $(\textsc{StorePwdFile}, sid, esid, \mathsf{AS}, \bot)$ to $\mathcal{F}$.

Stealing Password Data and Offline Password Test
- On $(\textsc{StealPwdFile}, sid, \mathsf{AS}, \mathsf{uid})$ from $\mathcal{A}$, forward it to $\mathcal{F}$.
  - On $\mathcal{F}$'s response of "password file stolen", retrieve $\langle \mathsf{AS}, \mathsf{uid}, n_S, r_{\hat{I}} \rangle$, pick $k_C, n_C, t, \rho \leftarrow\$ \mathbb{Z}_q$, and reveal $(\mathsf{uid}, k_C, n_S, n_C, t, \rho)$ to $\mathcal{A}$.
  - On $\mathcal{F}$'s response of "no password file" or pw, forward it to $\mathcal{A}$.
- On $(\textsc{OfflineTestPwd}, sid, \mathsf{uid}, \mathsf{pw}^*)$ from $\mathcal{A}$, pass it to $\mathcal{F}$, and on $\mathcal{F}$'s response, forward it to $\mathcal{A}$.

Online Authentication
- On $(\textsc{Auth}, sid, asid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid})$ from $\mathcal{F}$, retrieve $\langle \mathsf{AS}, \mathsf{uid}, n_S, r_{\hat{I}} \rangle$, compute $x_1 \leftarrow g^{r_J}$, $x_2 \leftarrow g^{u \cdot r_J + k_S \cdot r_{\hat{I}}}$, record $\langle asid, \mathsf{AS}, \mathsf{RL}, x_1, x_2, r_{\hat{I}}, r_J \rangle$, and set $J++$. Then send $(sid, asid, \mathsf{uid}, n_S, x_2)$ to $\mathcal{A}$ as AS's message to RL.
- On message $(sid, asid, \mathsf{uid}', n'_S, x'_2)$ from $\mathcal{A}$, send $(\textsc{AuthResp}, sid, asid, \mathsf{uid}')$ to $\mathcal{F}$. On the response of $\textsc{succ}$ from $\mathcal{F}$ (limit not reached), compute $h_S \leftarrow H_s(\mathsf{uid}', n'_S)$, $y_1 \leftarrow {h_S}^{-u+k_S}$, $y_2 \leftarrow (x'_2/y_1)^{1/u}$, $\zeta_1 \leftarrow \mathsf{PoK}_2(u, k_S : h_S^{-1}, h_S, y_1)$, $\zeta_2 \leftarrow \mathsf{PoK}_1(u : y_2, x'_2/y_1)$, send $(sid, asid, y_1, y_2, \zeta_1, \zeta_2)$ to $\mathcal{A}$ as RL's message to AS.
- On message $(sid, asid, y'_1, y'_2, \zeta'_1, \zeta'_2)$ from $\mathcal{A}$, recover $\langle asid, \mathsf{AS}, \mathsf{RL}, x_1, x_2, r_{\hat{I}}, r_{\hat{J}} \rangle$ and $\langle \mathsf{RL}, u, k_S \rangle$, and check that $\mathsf{Vf}_2(g^u, g^{k_S}, \zeta'_1 : 1/g^{r_{\hat{I}}}, g^{r_{\hat{I}}}, y'_1)$ and $\mathsf{Vf}_1(g^u, \zeta'_2 : y'_2, x_2/y'_1)$.
  - If ζ'_1, ζ'_2 are valid:
    - If $y'_1 = g^{r_{\hat{I}} \cdot (-u+k_S)} \wedge y'_2 = g^{r_{\hat{J}} + r_{\hat{I}}}$, send $(\textsc{AuthResult}, sid, asid, \mathsf{AS}, \top)$ to $\mathcal{F}$.
    - Else, <u>Sim aborts</u>.
  - If ζ'_1 or ζ'_2 is invalid, send $(\textsc{AuthResult}, sid, asid, \mathsf{AS}, \bot)$ to $\mathcal{F}$.

Online Password Test
- On $(\textsc{TestPwd}, sid, \mathsf{AS}, \mathsf{RL}, \mathsf{uid}, \mathsf{pw}^*)$ from $\mathcal{A}$, pass it to $\mathcal{F}$, and on $\mathcal{F}$'s response, forward it to $\mathcal{A}$.

Fig. 20: Simulator Sim for the RePhoenix protocol ($\mathcal{F}$ denotes $\mathcal{F}_{\text{PH}}$).

Proof. As shown in Fig. 20, We construct a simulator Sim. Sim interacts with adversary $\mathcal{A}$ and the ideal UC functionality of $\mathcal{F}_{\mathrm{PH}}$ ($\mathcal{F}$ in short). Without loss of generality, we assume that $\mathcal{A}$ is a dummy adversary (i.e., it is merely a pass-through machine that outsources all its computation to the environment $\mathcal{E}$). We now prove the Theorem 5 by arguing that Sim generates a view to $\mathcal{E}$, indistinguishable from the real-world view interacting with RePhoenix.

We first compare the differences in the environment's views in the real and ideal worlds and then prove that the differences are indistinguishable.

We start with an analysis of the views of environment $\mathcal{E}$ through the parties AS and RL, which is as follows:

- Send $(\textsc{Eval}, sid, esid, \mathsf{AS}, \mathsf{un}, \mathsf{pw})$ from AS to $\mathcal{F}$ in the ideal world and to RePhoenix in the real world.
- Receive $(\textsc{EvalResp}, sid, esid, \mathsf{uid}^*)$ from RL, where uid^* comes from $\mathcal{A}$ in both world. In the ideal world, the uid^* is specified in message $(\textsc{EvalResp}, sid, esid, \mathsf{RL}, \mathsf{uid}^*)$ from $\mathcal{A}$; And in the real world, the uid^* is specified by $\mathcal{A}$ as it is sent by AS to RL on a public channel.
- Receive $(\textsc{Eval}, sid, esid, \rho)$ from AS if validation successes. In the ideal world, $\mathcal{F}$ defines verifiability as flag, where $\mathsf{flag} = \top$ indicates that the RL response meets the protocol specification. In the real-world RePhoenix, it uses NIZK schemes to realize the verifiability functionality of flag. As for the evaluation value, $\mathcal{F}$ outputs a random value, i.e., $\rho \leftarrow\!\!\$ \{0,1\}^{\ell}$. While in the real world, RePhoenix computes $\rho \leftarrow \mathsf{pk_1}^{v} \cdot H_C(\mathsf{un}, \mathsf{pw}, n_C)^{k_C} \cdot {h_S}^{k_S}$, where v, n_C, k_C are randoms.
- Receive $(\textsc{Eval}, sid, esid, \perp)$ from AS if validation fails in both worlds.
- Send $(\textsc{Update}, sid, (\mathsf{RL}, \Delta k), (\mathsf{RL}', \Delta k'))$ from AS to $\mathcal{F}$ in the ideal world and RePhoenix in the real world.
- Send $(\textsc{Auth}, sid, asid, \mathsf{RL}, \mathsf{un}, \mathsf{pw}')$ from AS to $\mathcal{F}$ in the ideal world and to RePhoenix in the real world;
- Receive $(\textsc{AuthResp}, sid, asid, \mathsf{uid}^*)$ from RL, where uid^* comes from $\mathcal{A}$ in both world.
- Receive $(\textsc{Auth}, sid, asid, \mathsf{accept}/\mathsf{reject}/\perp)$ from AS, (accept if $\mathsf{pw}' = \mathsf{pw}$, or reject if $\mathsf{pw}' \neq \mathsf{pw}$, or $\perp$ if validation fails). As the $\textsc{Eval}$ message from AS, $\mathcal{F}$ defines verifiability using the parameter flag, and RePhoenix realizes the verifiability using NIZK proofs. If the verification fails, $\mathcal{F}$ and RePhoenix outputs the same $\perp$. If the validation successes, $\mathcal{F}$ verifies $\mathsf{pw} = \mathsf{pw}'$, and outputs accept (resp. reject) if it is true (resp. not true). While RePhoenix checks that $y_2' = x_1 \cdot h_S$ and returns accept or reject depending on the result. $y_2' = (x_2^{1/u} / {h_S}^{k_S/u}) \cdot h_S = x_1 \cdot (h_C / h_C')^{k_C} \cdot h_S$, where $h_C = H_C(\mathsf{un}, \mathsf{pw}, n_C), h_C' = H_C(\mathsf{un}, \mathsf{pw}', n_C)$. Therefore, $y_2' = x_1 \cdot h_S$ equals to $h_C = h_C'$, $H_C(\mathsf{un}, \mathsf{pw}, n_C) = H_C(\mathsf{un}, \mathsf{pw}', n_C)$. Assuming that H_C is a collision-resistant cryptographic hash function, we have that $y_2' = x_1 \cdot h_S$ can be used to verify that $\mathsf{pw} = \mathsf{pw}'$.

The only difference is evaluation output if Sim does not abort. $\mathcal{F}$ outputs a random value, i.e. $\rho \leftarrow\!\!\$ \{0,1\}^{\ell}$; while in RePhoenix, the output is $\rho \leftarrow \mathsf{pk_1}^{v} \cdot h_C^{k_C} \cdot {h_S}^{k_S}$ for $h_C = H_C(\mathsf{un}, \mathsf{pw}, n_C)$, $h_S = (\mathsf{uid}, n_S)$ and randoms v, k_C, n_C. The ${h_S}^{k_S}$ will be revealed to $\mathcal{E}$ via $\mathcal{A}$ in the evaluation process; $\mathcal{E}$ can learn ${h_C}^{k_C}$

upon the AS compromise; pk_1 is the public information. Due to the blinding factor v, the environment can not distinguish the real-world $\rho \leftarrow {\mathsf{pk}_1}^v \cdot h_C^{k_C} \cdot {h_S}^{k_S}$ from the ideal-world $\rho \leftarrow\!\!\$\ \{0,1\}^\ell$.

Next, we show that the probability that Sim aborts is negligible. The abort event happens in the simulated world when the NIZK proof ζ is valid under the intended public key. Still, the server response y is not calculated using the function corresponding to the secret key. This means the NIZK scheme fails because $\mathcal{A}$ successfully constructs a valid NIZK proof for a y' that is generated outside the protocol specification.

On responding message $(sid, esid, y', \zeta')$ from $\mathcal{A}$, Sim Will also verify that $\mathsf{pk}' = \mathsf{pk}$ if the NIZK proof ζ' is valid under the intended public key pk $(= (g^u, g^{k_S}))$. pk' denotes the actual public key for computing the y'. Suppose the verification of $\mathsf{pk}' = \mathsf{pk}$ does not pass. In that case, Sim will abort, which means that the NIZK scheme fails, because $\mathcal{A}$ successfully constructs a valid NIZK proof for a y' generated outside the protocol specification. In particular, since Sim simulates h_S with g^{r_I}, we have $y' = {h_S}^{k'_S} = g^{r_I \cdot k'_S} = (g^{k'_S})^{r_I}$. In this way, Sim can extract pk' by $\mathsf{pk}' \leftarrow (y')^{1/r_I}$ and compare it with pk.

On responding message $(sid, ssid_2, y'_1, y'_2, \zeta_1, \zeta_2)$, Sim verifies that $y'_1{=}\mathsf{pk}_1^{-r_I} \cdot \mathsf{pk}_2^{r_I}$ and $y'_2{=}g^{r_J + r_I}$ if the NIZK proofs ζ_1 and ζ_2 are valid, and outputs FAIL if any of the two equations does not hold. The two verification equations are used to verify that the server responses y'_1, y'_2 are calculated under the intended keys corresponding to pk. In particular, Sim simulates h_S with g^{r_I} and simulates x_2 with $g^{u \cdot r_J + k_S \cdot r_I}$. Suppose that (u', k'_S) is the actual key used to calculate (y'_1, y'_2), we have that $y'_1 = g^{-u' \cdot r_I + k'_S \cdot r_I}$ and $y'_2 = g^{((u \cdot r_J + k_S \cdot r_I) - (-u' \cdot r_I + k'_S \cdot r_I))/u'} = g^{(u/u') \cdot r_J + (k_S - k'_S + 1) \cdot r_I}$. Therefore, the two verification equations of Sim can be used to constitute the solution equation set for (u', k'_S) as follows:

$$\begin{cases} y'_1{=}\mathsf{pk}_1^{-r_I} \cdot \mathsf{pk}_2^{r_I}, \\ y'_2{=}g^{r_J + r_I}, \end{cases} \Rightarrow \begin{cases} g^{-u' \cdot r_I + k'_S \cdot r_I}{=}\mathsf{pk}_1^{-r_I} \cdot \mathsf{pk}_2^{r_I}, \\ g^{(u/u') \cdot r_J + (k_S - k'_S + 1) \cdot r_I}{=}g^{r_J + r_I}. \end{cases}$$

It has a unique solution, i.e. $(u', k'_S) = (u, k_S)$.

We follow the spirit of [19] to estimate the upper limit of NIZK failure probability. Suppose that the appropriate input was never queried to H_{NIZK_1} at the time that the honest AS (played by Sim) performs the NIZK verification. Considering that H_{NIZK_1} is modeled as a random oracle that randomly samples outputs from $\mathbb{Z}_q$, the probability that the H_{NIZK_1} output falls on precisely the verification hash value is $1/q$. Suppose that the adversary queried the appropriate input at some point before delivery to Sim. Similarly, given that (g, pk, x, y) is not a DDH tuple and the NIZK verification hash value is random over $\mathbb{Z}_q$, the probability of the verification equation being successful is $1/q$. Therefore, the failure probability of the $(\mathsf{PoK}_1, \mathsf{Vf}_1)$ scheme is $1/q + 1/q$. Similarly, the failure probability of the $(\mathsf{PoK}_2, \mathsf{Vf}_2)$ scheme is $1/q + 1/q$.

We conclude that the abort events happen with a probability of at most $2/q + 2/q^2$.

We continue to analyze the views of environment $\mathcal{E}$ through adversary $\mathcal{A}$. For brevity, we omit session identifiers from all messages.

In the real world, RePhoenix provides the following views to $\mathcal{E}$ through $\mathcal{A}$. We assume that the transmission channel of evaluation and authentication is public. When $\mathcal{A}$ acts as AS, it can obtain the messages from AS to RL.

– Receive uid from AS after $\mathcal{E}$ sends the EVAL message to RePhoenix for $(\mathsf{un}, \mathsf{pw})$, where $\mathsf{uid} = H(\mathsf{un}, s)$.
– Receive (uid, n_S, x_2) from AS after $\mathcal{E}$ sends the AUTH message to RePhoenix for $(\mathsf{un}, \mathsf{pw}')$. In RePhoenix, $\mathsf{uid} = H(\mathsf{un}, s)$, and n_S is recovered from the record of (un, n_S). RePhoenix computes (x_1, x_2) by $x_1 = t{\cdot}g^z = g^{v \cdot z}$ for random z and $x_2 = \rho{\cdot}\mathsf{pk}_1{}^z/{h'_C}^{k_C} = g^{u\cdot(v+z)}{\cdot}(h_C/h'_C)^{k_C}{\cdot}{h_S}^{k_S}$ for $h_C = H_C(\mathsf{un}, \mathsf{pw}, n_C), h'_C = H_C(\mathsf{un}, \mathsf{pw}', n_C)$.

When $\mathcal{A}$ acts as AS, it can learn the messages from RL to AS.

– Receive (y, ζ, n_S) from RL after $\mathcal{A}$ forwards uid' to RL. RePhoenix computes $h_S = H_S(\mathsf{uid}', n_S)$ for random n_S sampled from $\{0,1\}^\kappa$, computes $y = {h_S}^{k_S}$ and its NIZK proof $\zeta = \mathsf{PoK}_1(k_S : h_S, y)$.
– Receive $(y_1, y_2, \zeta_1, \zeta_2)$ from S after $\mathcal{A}$ forwards $(\mathsf{uid}', n'_S, x'_2)$ to S. RePhoenix computes $h_S = H_S(\mathsf{uid}', n'_S), y_1 = (1/h_S)^u{\cdot}{h_S}^{k_S} = {h_S}^{-u+k_S}, y_2 = (x'_2/y_1)^{1/u}$ and their NIZK proofs $\zeta_1 = \mathsf{PoK}_2(u, k_S : 1/h_S, h_S, y_1), \zeta_2 = \mathsf{PoK}_1(u : y_2, x'_2/y_1)$.

Moreover, $\mathcal{A}$ can learn the leakage of their static storage from RL and AS when $\mathcal{E}$ permits it to compromise the parties.

– Receive (u, k_S) from RL after $\mathcal{A}$ sends COMPROMISE to RePhoenix under environment permission, where RePhoenix initiates the server keys (u, k_S) with two random values from $\mathbb{Z}_q$.
– Receive $(\mathsf{uid}, k_C, n_S, n_C, t, \rho)$ from AS after $\mathcal{A}$ sends STEALPWDFILE(uid) to RePhoenix under environment permission. In RePhoenix, k_C, n_C, v are random values. For the user name and password pair of $(\mathsf{un}, \mathsf{pw})$ and the server keys of (u, k_S), $\rho = g^{u\cdot v}{\cdot}{h_C}^{k_C}{\cdot}{h_S}^{k_S}$, where h_C is the password salted hash, i.e. $h_C = H_C(\mathsf{un}, \mathsf{pw}, n_C)$ for random n_C sampled by AS and h_S is the user name hash, i.e. $h_S = H_S(H(\mathsf{un}, s), n_S)$ from random n_S sampled by RL.

The simulator Sim simulates the above real-world views piece by piece to provide an indistinguishable view of the environment in the simulated world. When $\mathcal{A}$ acts as RL, Sim strives to act as AS as follows to send indistinguishable responses to $\mathcal{A}$.

– Receive uid from AS after $\mathcal{E}$ sends the EVAL message to $\mathcal{F}$ for $(\mathsf{un}, \mathsf{pw})$. As the definition of $\mathcal{F}$, it sends the $(\text{EVAL}, \cdots, \mathsf{AS}, \mathsf{RL}, \mathsf{uid})$ message to Sim. Then Sim forwards the uid to RL.
– Receive (uid, n_S, x_2) from AS after $\mathcal{E}$ sends the AUTH message to $\mathcal{F}$ for $(\mathsf{un}, \mathsf{pw}')$. As the definition of $\mathcal{F}$, it sends the $(\text{AUTH}, \cdots, \mathsf{AS}, \mathsf{RL}, \mathsf{uid})$ message to Sim. For the consistency of n_S, Sim recovers n_S from $\langle C, \mathsf{uid}, n_S, r_I \rangle$. Sim simulates x_1 with g^{r_J} and simulates x_2 with $g^{u \cdot r_J + k_S \cdot r_I}$.

When $\mathcal{A}$ acts as RL, Sim simulates RePhoenix's calculation of (y, ζ) and $(y_1, y_2, \zeta_1, \zeta_2)$ in a copied manner as follows. Therefore, $\mathcal{A}$ cannot distinguish them from the real-world messages in RePhoenix.

– Receive (y, ζ, n_S) from RL after $\mathcal{A}$ forwards uid' to RL, where $y = {h_S}^{k_S}$ s.t. $h_S = H_S(\mathsf{uid}', n_S)$ for random n_S, $\zeta = \mathsf{PoK}_1(k_S : h_S, y)$ for the server key k_S sampled by Sim randomly in the initiation phase;
– Receive $(y_1, y_2, \zeta_1, \zeta_2)$ from RL after $\mathcal{A}$ forwards $(\mathsf{uid}', n'_S, x'_2)$ to RL. Sim computes $h_S = H_S(\mathsf{uid}', n'_S)$, $y_1 = {h_S}^{k_S - u}$, $y_2 = (x'_2/y_1)^{1/u}$, and their NIZK proofs $\zeta_1 = \mathsf{PoK}_2(u, k_S : 1/h'_S, h'_S, y_1), \zeta_2 = \mathsf{PoK}_1(u : y_2, x'_2/y_1)$ with the server keys (u, k_S).

Additionally, Sim simulates information leakage upon RL/AS compromise as follows. Note that all messages between $\mathcal{A}$ and $\mathcal{F}$ pass through the ideal adversary (i.e., Sim).

– Receive (u, k_S) after $\mathcal{A}$ sends COMPROMISE to $\mathcal{F}$ under environment permission. The server keys (u, k_S) in Sim are random values from $\mathbb{Z}_q$.
– Receive $(\mathsf{uid}, k_C, n_S, n_C, t, \rho)$ after $\mathcal{A}$ sends STEALPWDFILE to $\mathcal{F}$ with environment permission. Sim retrieves n_S from the record $\langle \mathsf{AS}, \mathsf{uid}, n_S, r_I \rangle$ indexed by uid. For k_C, n_C, t, ρ, Sim simulates them with random values from $(\mathbb{Z}_q, \{0,1\}^\kappa, \mathbb{G}, \mathbb{G})$.

As stated above, environment $\mathcal{E}$ cannot distinguish the real and simulated worlds using messages sent by AS to RL that are eavesdropped by adversary $\mathcal{A}$. Next, We analyze the difference in messages from AS to RL.

– $uid = H(\mathsf{un}, s)$ in RePhoenix and $uid = \mathsf{prf}(\mathsf{un})$ in Sim. We assume that $H(\cdot)$ is a random oracle. Therefore, $\mathcal{E}$ can not distinguish the two worlds via uid.
– $x_2 = g^{u\cdot(v+z)} \cdot (h_C/h'_C)^{k_C} \cdot {h_S}^{k_S}$ in RePhoenix and $x_2 = g^{u \cdot r_J} \cdot {h_S}^{k_S}$ in Sim, where r_J is the next available element of the random sequence. $\mathcal{A}$ can learn the value of ${h_S}^{k_S}$ from the public channel from RL to AS in the process of evaluation. The main task of $\mathcal{A}$ here is to distinguish the rest sub-term of x_2, i.e. distinguish the real-world $g^{u\cdot(v+z)} \cdot (h_C/h'_C)^{k_C}$ and the simulated-world $g^{u \cdot r_J}$. Due to v, z, r_J are random values from group $\mathbb{Z}_q$ and always kept secret from $\mathcal{E}$, it cannot distinguish between the two rest sub-terms of x_2 in RePhoenix and Sim, even for $\mathcal{A}$ who compromises the server key u and the file record.

By compromising RL and AS under environment permission, $\mathcal{A}$ learns their keys and the password file. The keys (including (u, k_S)) and the salt s are random values both in the real-world RePhoenix and the simulated-world Sim and, therefore, indistinguishable in the two worlds. As for the password file of $(\mathsf{un}, \mathsf{pw})$, i.e. $(\mathsf{uid}, k_C, n_S, n_C, t, \rho)$, k_C, n_C are random value that are indistinguishable in RePhoenix and Sim. The t and ρ from Sim are also random values. In RePhoenix, $t = g^v$ is also a random value. Therefore, $\mathcal{A}$ cannot distinguish the two worlds via t. In the real world, RePhoenix computes $\rho = g^{u \cdot v} \cdot {h_C}^{k_C} \cdot {h_S}^{k_S}$. $\mathcal{A}$ can query H_C for $(\mathsf{un}, \mathsf{pw})$ to obtain h_C and compute ${h_C}^{k_C}$ with the compromised k_C. Additionally, $\mathcal{A}$ has learnt the ${h_S}^{k_S}$ from the public evaluation message sent by RL to AS. Based on the existing knowledge of ${h_C}^{k_C}$ and ${h_S}^{k_S}$, $\mathcal{A}$'s ability to distinguish ρ between the two worlds is equivalent to distinguishing the sub-term $g^{u \cdot v}$ from a random value. Due to v being a random value from group $\mathbb{Z}_q$ and always kept secret from $\mathcal{E}$, $g^{u \cdot v}$ can be seen as a random value for $\mathcal{A}$. Therefore, $\mathcal{A}$ cannot distinguish the two worlds via ρ. We can conclude that the environment cannot

```
$\mathsf{Exp}^{b}_{\text{TF-RePhoenix},\mathcal{L},\mathcal{A}}$
1 :  $\mathsf{p} \leftarrow (n, t, \kappa)$
2 :  $(k_C, u, k_S, T, \mathsf{un}, \mathsf{pw}, \mathsf{sta}) \leftarrow\$ \mathcal{A}_1(\mathsf{p})$
3 :  $b_0 := (T = \mathsf{Eval}\{C(k_C, \mathsf{un}, \mathsf{pw}), S(u, k_S)\})$
4 :  $(u', k'_S, \mathsf{pk}', [\![\Delta k]\!]) \leftarrow\$ S'.\mathsf{Init}(\mathsf{p})$
5 :  if $b = 0$ then update password record and key rotation.
6 :    $\Delta k \leftarrow\$ \mathsf{ReSecret}\{C(k_C), S'(\Delta k_1), S_2(\Delta k_2), \cdots, S_t(\Delta k_t)\}$
7 :    $(T', k'_C) \leftarrow\$ C.\mathsf{Update}(\Delta k, T)\{$
8 :      $(\alpha, \beta, \gamma) \leftarrow \Delta k, (h, t_1, t_2, n_S, n_C) \leftarrow T$
9 :      $v^* \leftarrow\$ \mathbb{Z}_q, t'_1 = t_1 \cdot g^{v^*}, t'_2 = (t_2)^{\alpha} \cdot (t_1)^{\beta} \cdot (h_S)^{\gamma} \cdot \mathsf{pk}_1^{\alpha \cdot v^*} \cdot g^{\beta \cdot v^*}$
10 :      $T' \leftarrow (h, t'_1, t'_2, n_S, n_C), k'_C \leftarrow k_C^{\alpha}$   $\}$
11 :  else generate password record by the evaluation protocol.
12 :    $\mathsf{aux} \leftarrow \mathcal{L}(T)$
13 :    $T' \leftarrow\$ \mathsf{Eval}\{C(k'_C, \mathsf{un}, \mathsf{pw}, \mathsf{aux}), S(u', k'_S, \mathsf{aux})\}$
14 :  endif
15 :  $b' \leftarrow\$ \mathcal{A}_2(\mathsf{sta}, k'_C, u', k'_S, T')$
16 :  $b_1 := (b = b')$
17 :  return $b_0 \wedge b_1$
```

Fig. 21: Game-based experiment for forward security.

distinguish between the real world and the simulated world through the view of adversary $\mathcal{A}$.

Based on the above analysis, we can conclude that environment $\mathcal{E}$, from the view of the party AS, can distinguish the real world from the ideal world due to the abort event of Sim. The upper bound of the distinguishability probability is negligible, i.e., $2/q + 2/q^2$. If there are n_e evaluation requests via EVAL and n_a authentication requests via AUTH, the total distinguishability probability is $2 \cdot (n_e/q + n_a/q^2)$. However, from the view came from RL and $\mathcal{A}$, $\mathcal{E}$ cannot distinguish whether it is interacting with the ideal-word functionality $\mathcal{F}$ and simulated-word simulator Sim or the real-world RePhoenix protocol.

C Forward Security

We define a forward security experiment played between a challenger and a two-stage adversary $\mathcal{A}$, following the forward security experiment for Phoenix [28]. The challenger acts as the client C and $t+1$ servers, including the on-duty server S, and t off-duty servers. As shown in Fig. 21, the first-stage adversary $\mathcal{A}_1$ outputs k_C of C, (u, k_S) of S, and a tuple $(T = \{\mathsf{uid}, n_s, n_c, t, \rho\}, \mathsf{un}, \mathsf{pw})$, which are verified in line 3. An off-duty server S' is initiated and determined as the next on-duty server. According to the selection bit b, the challenger C either rotates keys by using $\mathsf{ReSecret}$ and updates the password record by using Update (Update denotes the token-based update function of threshold rotation), or generates a fresh password record for $(\mathsf{un}, \mathsf{pw})$ by using an evaluation protocol with S'. In the evaluation protocol, both C and S' additionally take some auxiliary information $\mathsf{aux} = \mathcal{L}(T)$ as input, where $\mathcal{L}$ defines the leakage function that outputs (uid, n_S, n_C). The second-stage adversary learns updated/fresh keys and password records and outputs its guess b' for b. Finally, the experiment returns $b_0 \wedge b_1$, which is true if and only if $\mathcal{A}_1$ provides a valid experiment tuple $(T, \mathsf{un}, \mathsf{pw})$ and $\mathcal{A}_2$ guesses correctly.

Next, we prove that the probability of the true result is negligible. Since that (uid, n_S, n_C) of the updated T' are exactly the same as that of the fresh T', we only need to discuss t' and ρ'. Due to the update correctness proved in Section 5, we have the password record in the same form before and after the update. Therefore, we have $t' = g^{v'}, \rho' = (\mathsf{pk}_1')^{v'} \cdot H_C(\mathsf{un}, \mathsf{pw}, n_C)^{k_C'} \cdot H_S(h, s)^{k_S'}$ when $b = 0$. Due to $(\mathsf{uid}, n_C, n_S) \leftarrow \mathcal{L}(T)$ as auxiliary information inputting to the fresh evaluation protocol when $b = 1$, so $t' = g^{v''}, \rho' = (\mathsf{pk}_1'')^{v'} \cdot H_C(\mathsf{un}, \mathsf{pw}, n_C)^{k_C'} \cdot H_S(h, s)^{k_S'}$. The only difference is the exponent of t', which is $v' := v + v^*$ in the updated record and $v'' := v + v^{**}$. The v^* and v^{**} are indistinguishable because they are independently selected random values. Thus, we can conclude that the adversary cannot distinguish between the updated password record and the fresh one. This means the forward security with the leakage function $\mathcal{L}$, called $\mathcal{L}$-forward security, of TF-RePhoenix.

D Performance Evaluation

D.1 Implement and Experiment Setting

We implemented TF-RePhoenix's cryptographic algorithms based on the Charm-Crypto cryptographic framework [25] and using NIST P 256 as the group. The Shamir's secret sharing [33] was implemented based on an open-source code [34]. The PH server was built as a web application using the Falcon Python Web framework [17]. The communication was implemented with the Python httplib2 library, where messages were passed to the server as GET request parameters and returned to the client as JSON. All experiments run on a machine equipped with Intel(R) Core(TM) i7-8850H/2.60GHz $\times 2$ & 3.8 GiB RAM, installed with 64-bit Ubuntu 20.04.3 LTS.

D.2 Performance Comparison

The evaluation and authentication protocols are critical protocols of PH. After a user submits the username and password, he will generally wait for the registration or log in to be completed. High-latency evaluation and authentication protocols can reduce the user's favorable impression of the PH. Therefore, it is essential to ensure their low latency. Our TF-RePhoenix solution to the single points of failure (SPF) does not change the original RePhoenix's evaluation and authentication protocols. As shown in Table 1, TF-RePhoenix has the same high efficiency as RePhoenix in terms of computation overhead and round. In other words, our threshold failover solution to SPF has **optimal performance**.

For comparison, Table 1 summarizes the performance of T-PHE [4], which is an instantiated threshold solution to SPF, compared with the original PHE with SPF. Both PHE [27] and T-PHE [4] are proposed in a PH-derived field for hardened password-based encryption. We can see that while T-PHE eliminates the SPF of PHE [27], it also results in a general decline in performance. Under the setting of $n = t = 1$ (with the lowest cost in this setting), the evaluation latency of T-PHE [4] is 1.2 times that of PHE, and the authentication latency is 5.3 times. And they linearly increase with t [4].

D.3 Performance Evaluation of Failover

We evaluate the latency of failover in TF-RePhoenix when failure happens. We test failover in two phases: C (i.e., the authentication server) reconstructs the update token from t servers; and C uses the token to update all password records to the new ones hardened by the new key. All results are based on the averages of 1,000 independent executions.

First phase: Time to reconstruct an update token. This experiment measures the time it takes for TF-RePhoenix to reconstruct an update token

Table 1: The comparison of computation overhead, rounds, and storage cost of the client between the original and improved schemes.

Scheme			**Computation overhead**		**Rounds**		**Storage**
			Eval	Auth	Eval	Auth	Client
PHE	USENIX SEC'18	[27]	$8H^{\dagger}$+12E	8H+13E	1	2	$2L_\kappa+2L_g$
T-PHE	CCS'20	[4]	6H+17E	6H+91E	3	6	$3L_\kappa+2L_g$
Performance ratio			**1.2**	**5.3**	**3**	**3**	**1.7**
Phoenix[‡]	USENIX SEC'17	[28]	2H+5E	2H+15E	2	2	$3L_\kappa+3L_g$
RePhoenix		Sec. 2	2H+10E	2H+12E	2	2	$3L_\kappa+3L_g$
TF-RePhoenix		Sec. 5	2H+10E	2H+12E	2	2	$3L_\kappa+3L_g$
Performance ratio			**1**	**1**	**1**	**1**	**1**

[†] H denotes the hash function mapped to group $\mathbb{G}$, and E represents the exponential operation in group $\mathbb{G}$. L_κ denotes the length of the element in $\{0,1\}^\lambda$, and L_g denotes the length of the element in $\mathbb{G}$. In our experiments with NIST P 256 and SHA256, each H takes $24.26ms$, and each E takes $39.84ms$, which are used to estimate the latency ratio of the original and improved protocols.
[‡] Note that Phoenix's registration protocol does not have verifiability.

Table 2: Experiment results of time cost both of recovering the update token in various threshold parameters and updating password records (in ms).

Threshold parameter n	5	7	9	11	13	15
Time to recover update token (t=n)	0.475	0.692	0.874	1.116	1.389	1.671
Time to recover update token (t=4)	0.387	0.385	0.384	0.385	0.389	0.387
Number of updated records	1	10	100	1,000	10,000	–
Time to update records	0.368	3.680	36.896	364.823	3,641.635	–

under different threshold parameters. In Table 2, results show that the time cost is linear with the number of shares required to reconstruct the token (i.e., t). Increasing the number of PH servers (i.e., n) affects the run time. For $(n, t) = (15, 4)$, the run time needed to reconstruct a token is very short, only 0.387 milliseconds.

Second phase: Time to update password records. As shown in Table 2, the time to update password records is linear with the number of password records. The amortized time to update a password record is 0.36 milliseconds. It can be estimated that updating one million records takes around six minutes. For more extensive password records, like Facebook's approximately three billion users, updates can be easily parallelized on-demand, making it even more efficient.

From the above experimental results, we can conclude that our TF-RePhoenix provides a low-latency failover, taking a very short time to recover the password-hardening service from its failure. Even if the password storage records are of millions of levels, our protocol only needs a few minutes to failover. Restoring the update token and updating a record takes less than milliseconds, so even user requests submitted at the moment of the failure can continue to be processed after only a millisecond latency.

E The Verifiable and Partially Oblivious Pseudo-Random Function (VPOPRF)

In Everspaugh *et al.*'s PHS scheme [11], the PH protocol is equivalent to a verifiable and partially oblivious PRF (VPOPRF). Both the password evaluation and authentication phases involve the server performing the same deterministic computation task (i.e., $(\cdot)^k$). If the output value from the authentication phase matches the corresponding password record output from the evaluation phase, the login password is deemed correct. In this section, we propose the ideal functionality of VPOPRF ($\mathcal{F}_{\text{VPOPRF}}$) in the UC security framework and prove that Everspaugh *et al.*'s PH protocol (called Pythia) UC-realizes $\mathcal{F}_{\text{VPOPRF}}$, which answers the question left over from [11] regarding the level of pseudo-randomness security that their PH protocol is provably capable of. Other UC formalization works include OPRF [20], VOPRF [19], TOPRF [21], adaptive VOPRF [22], CorOPRF [23], and vedpOPRF [8]. Additionally, [36] introduces a non-UC definition of VPOPRF.

In the main text, we focus on more commonly used PHs, whose authentication protocols are separate from the evaluation protocols (examples include [32,28,24]). One point of discussion is how these differ from VPOPRF-type PHs [11] described above. Firstly, the evaluation protocols require randomness security, as opposed to the deterministic outputs of VPOPRF. This means the evaluation protocol will produce different outputs each time, even with the same password input. This prevents attackers from verifying passwords through the evaluation protocol. Secondly, only the authentication protocol requires unpredictability

Functionality $\mathcal{F}_{\text{VPOPRF}}$

Public Parameter: Evaluation output-length ℓ, polynomial in security parameter κ.
Conventions: For $sid, \mathsf{pk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}$, return $F_{sid,\mathsf{pk}}(x_{\mathsf{pub}}, x_{\mathsf{pri}})$ if defined; else, assign and return $F_{sid,\mathsf{pk}}(x_{\mathsf{pub}}, x_{\mathsf{pri}}) \leftarrow\!\$\ \{0,1\}^{\ell}$.

Initialization

- On $(\textsc{Init}, sid, S)$ from $\mathcal{E}$, if this is the first $\textsc{Init}$ message for sid, send it to $\mathcal{A}^*$.
- On $(\textsc{Parameter}, sid, S, \mathsf{pk})$ from $\mathcal{A}^*$, ignore this message if $\langle \mathsf{param}, S, \cdot\rangle$ already exists. Otherwise, record $\langle \mathsf{param}, S, \mathsf{pk}\rangle$ and mark it FRESH, initialize $\mathsf{tx}(S) := 0$. If S is honest send $(\textsc{Parameter}, sid, \mathsf{pk})$ to S; else, mark S's param record COMPROMISED, and insert pk in CPK.

Server Compromise

- On $(\textsc{Compromise}, sid, S, \mathsf{pk}^*)$ from $\mathcal{A}^*$, if there is a record $\langle \mathsf{param}, S, \mathsf{pk}\rangle$ s.t. $\mathsf{pk}^* = \mathsf{pk}$ marked FRESH, re-mark it COMPROMISED, and insert pk in CPK.

Offline Evaluation

- On $(\textsc{OfflineEval}, sid, S, \mathsf{pk}^*, x_{\mathsf{pub}}, x_{\mathsf{pri}})$ from $P \in \{S, \mathcal{A}^*\}$, send $(\textsc{OfflineEval}, sid, S, \mathsf{pk}^*, F_{sid,\mathsf{pk}}(x_{\mathsf{pub}}, x_{\mathsf{pri}}))$ to P if any of following holds:
  - For $P = S$, S is compromised or $\mathsf{pk}^* = \mathsf{param}(S)$.
  - For $P = \mathcal{A}^*$, S is compromised or $\mathsf{pk}^* \neq \mathsf{param}(S)$.

Online Evaluation

- On $(\textsc{Eval}, sid, ssid, S, x_{\mathsf{pub}}, x_{\mathsf{pri}})$ from $P \in \{C, \mathcal{A}^*\}$, send $(\textsc{Eval}, sid, ssid, P, S, x_{\mathsf{pub}})$ to $\mathcal{A}^*$. On prfx from $\mathcal{A}^*$, ignore this message if prfx is used before; else record $\langle ssid, P, \mathsf{prfx}\rangle$ and send $(\textsc{Prfx}, sid, ssid, \mathsf{prfx})$ to P. If there is no tuple $\langle \mathsf{count}, sid, x_{\mathsf{pub}}, c\rangle$, record it for rate-limiting and initialize $c := 0$ (c is reset to 0 at the beginning of every time window). If this is the first $\textsc{Eval}$ message for $ssid$, record $\langle sid, ssid, C, S, \mathsf{pk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}\rangle$.
- On $(\textsc{SvrComplete}, sid, ssid, S)$ from S, retrieve $\langle sid, ssid, C, S, \mathsf{pk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}\rangle$ and $\langle \mathsf{count}, sid, x_{\mathsf{pub}}, c\rangle$, ignore this message if $c = \mathsf{limit}$. Otherwise, set $c++$ and send $(\textsc{SvrComplete}, sid, ssid, S)$ to $\mathcal{A}^*$ for some honest S. On prfx' from $\mathcal{A}^*$, send $(\textsc{Prfx}, sid, ssid, \mathsf{prfx}')$ to S. If there is $\langle ssid, C, \mathsf{prfx}\rangle$ s.t. $\mathsf{prfx} = \mathsf{prfx}'$, change it to $\langle ssid, C, \mathsf{OK}\rangle$. If $\mathsf{prfx} \neq \mathsf{OK}$, $\mathsf{tx}(S)++$.
- On $(\textsc{CltComplete}, sid, ssid, C, \mathsf{pk}^*, \mathsf{flag})$ from $\mathcal{A}^*$, ignore this message if no $\langle sid, ssid, C, S, \mathsf{pk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}\rangle$ or $\langle ssid, P, \mathsf{prfx}\rangle$ exists.
  - For $(\mathsf{flag} = \top \wedge \mathsf{pk}^* = \mathsf{pk})$, if $\mathsf{prfx} \neq \mathsf{OK}$ and $\mathsf{tx}(S) = 0$, ignore this message. Otherwise return $(\textsc{Eval}, sid, ssid, F_{sid,\mathsf{pk}}(x_{\mathsf{pub}}, x_{\mathsf{pri}}))$. If $\mathsf{prfx} \neq \mathsf{OK}$, $\mathsf{tx}(S)--$. If there is no $\langle \mathsf{file}, sid, ssid, C, S, \mathsf{pk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}\rangle$, record it and mark it FRESH.
  - In other cases, return $(\textsc{Eval}, sid, ssid, \perp)$.

Stealing Password Data

- On $(\textsc{StealPwdFile}, sid, ssid, S)$ from $\mathcal{A}^*$, return "no password file" to $\mathcal{A}^*$ if there is no record $\langle \mathsf{file}, sid, ssid, C, S, \mathsf{pk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}\rangle$. Otherwise, if it is marked FRESH, mark it COMPROMISED.
  - If $\langle \mathsf{param}, S, \mathsf{pk}\rangle$ is marked COMPROMISED and there is $\langle \mathsf{offlinetest}, \mathsf{pk}, x_{\mathsf{pri}}\rangle$, return x_{pri} to $\mathcal{A}^*$.
  - Else return "password file stolen" to $\mathcal{A}^*$.
- On $(\textsc{OfflineTestPwd}, sid, S, x_{\mathsf{pub}}, {x_{\mathsf{pri}}}^*)$ from $\mathcal{A}^*$, ignore this message if $\langle \mathsf{param}, S, \mathsf{pk}\rangle$ is not marked COMPROMISED. Otherwise, if there is no record $\langle \mathsf{file}, sid, \cdot, C, S, \mathsf{pk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}\rangle$ marked COMPROMISED, record $\langle \mathsf{offline}, \mathsf{pk}, x_{\mathsf{pri}}\rangle$; else do: if ${x_{\mathsf{pri}}}^* = x_{\mathsf{pri}}$ return "correct guess" to $\mathcal{A}^*$, else return "wrong guess".

Active Session Attacks

- On $(\textsc{TestPwd}, sid, ssid, C, S, x_{\mathsf{pub}}, {x_{\mathsf{pri}}}^*)$ from $\mathcal{A}^*$, retrieve records $\langle \mathsf{file}, sid, \cdot, C, S, \mathsf{pk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}\rangle$ marked COMPROMISED and $\langle \mathsf{count}, sid, x_{\mathsf{pub}}, c\rangle$, ignore this message if no such records or $c = \mathsf{limit}$. Otherwise, set $c++$ and if ${x_{\mathsf{pri}}}^* = x_{\mathsf{pri}}$ return "correct guess" to $\mathcal{A}^*$, else return "wrong guess" to $\mathcal{A}^*$.

Fig. 22: Verifiable and Partially Oblivious Pseudo-Random Function (VPOPRF) functionality $\mathcal{F}_{\text{VPOPRF}}$ with adaptive compromise.

Public Parameters and Components

- Security parameter κ. Finite field $\mathbb{Z}_q$ over prime order q, groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ over order q, and bilinear pairing $e : \mathbb{G}_T \leftarrow \mathbb{G}_1 \times \mathbb{G}_2$. Hash functions H, H_1, H_2 with ranges $\{0,1\}^\kappa, \mathbb{G}_1, \mathbb{G}_2$.
- NIZK schemes: $(\mathsf{PoK}_1, \mathsf{Vf}_1)$ for proving $y = x^k$.
- C picks a salt s in initialization phase.

Initialization

- On $(\textsc{Init}, sid, S)$, S picks $k \leftarrow\!\!\$\ \mathbb{Z}_q$ (public key $\mathsf{pk} := g^k$), stores $\langle sid, k\rangle$, and returns $(\textsc{Parameter}, sid, S, \mathsf{pk})$.

Server Compromise

- On $(\textsc{Compromise}, sid, S, \mathsf{pk}^*)$ from $\mathcal{A}$, reveal S's key k to $\mathcal{A}$ if $\mathsf{pk}^* = pk$.

Online Evaluation

- On $(\textsc{Eval}, sid, ssid, S, w \leftarrow H(\mathsf{un}, s), \mathsf{pw})$, C sends (w, x) to S, for x s.t. $r \leftarrow\!\!\$\ \mathbb{Z}_q, x \leftarrow H_2(\mathsf{pw})^r$, stores $(sid, ssid, w, \mathsf{pw}, r)$, and outputs $(\textsc{Prfx}, sid, ssid, (w, x))$.
- On $(\textsc{SvrComplete}, sid, ssid, S)$ and message (w, x) from C, S computes $x' \leftarrow e(H_1(w), x), t \leftarrow (x')^k$ and a proper NIZK proof $\zeta \leftarrow \mathsf{PoK}_1(k : x', y)$, sends (y, ζ) to C and outputs $(\textsc{Prfx}, sid, ssid, (w, x))$.
- On message (y, ζ) and pk from S, C verifies ζ. If not, output $(\textsc{Eval}, sid, ssid, \perp)$; otherwise, recover $(sid, ssid, w, \mathsf{pw}, r)$ and output $(\textsc{Eval}, sid, ssid, y^{1/r})$. If there is no record $\langle \mathsf{file}, sid, \cdot, w, \mathsf{pw}, \rho\rangle$, C records it with $ssid$ and $\rho := y^{1/r}$; otherwise C does: if $\rho = y^{1/r}$, output "correct guess", else output "wrong guess".

Stealing Password Data

- On $(\textsc{StealPwdFile}, sid, ssid, C)$ from $\mathcal{A}$, if there is record $\langle \mathsf{file}, sid, ssid, \mathsf{un}, \mathsf{pw}, \rho\rangle$, send ρ to $\mathcal{A}$.

Fig. 23: Adaptive Pythia [15].

security, which prevents online attackers from using a single online request to verify multiple password guesses.

The verifiable and partially oblivious pseudo-random function (VPOPRF) [11] is a two-party protocol, consisting of a client holding the inputs (i.e., $(x_{\mathsf{pub}}, x_{\mathsf{pri}})$) and a server holding the key k, generating a PRF output $F_k(x_{\mathsf{pub}}, x_{\mathsf{pri}})$ to the client, while the server knows nothing but x_{pri}. It is first proposed by Everspaugh *et al.* [11] for their PH protocol Pythia and first formally defined by Tyagi *et al.* [36] based on security games. In this section, we introduce a new security notion of VPOPRF in the UC security framework that covers input privacy, output pseudo-randomness, obliviousness, and verifiability.

There are two VPOPRF constructions, Pythia [11] based on a bilinear pairing and 3HashSDHI [36] based on the 2HashDH OPRF and the Dodis-Yampolskiy PRF, both with game-based security proof. Tyagi *et al.* [36] proved that the security (pseudorandomness) of 3HashSDHI in the random oracle model and based on a one-more gap strong Diffie-Hellman inversion assumption that can be from the q-DL assumption in the algebraic group model. Everspaugh *et al.* [11] proved the one-more unpredictability security and the one-more PRF security of their Pythia scheme. However, they believe Pythia cannot be proven secure relative to the oblivious PRF security with the partially oblivious setting, and they leave the question of what security level Pythia can be proven for. In this section, we prove that Pythia UC-realizes our defined VPOPRF functionality.

Public Parameters and Components

- Security parameter κ. Finite field $\mathbb{Z}_q$ over prime q, groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ over prime q, and bilinear pairing $e : \mathbb{G}_T \leftarrow \mathbb{G}_1 \times \mathbb{G}_2$. Hash functions H, H_1 with ranges $\{0,1\}^\kappa, \mathbb{G}_1$.
- NIZK scheme: $(\mathsf{PoK}_1, \mathsf{Vf}_1)$ for proving $y = x^k$.
- Pick $r_1, \cdots, r_N \leftarrow\!\!\$\, \mathbb{Z}_q$. Set $g_1 := g^{r_1}, \cdots, g_N := g^{r_N}$, and I, J.
- On $\mathcal{A}$'s fresh query x_{pri} to H_2, set $H_2(x_{\mathsf{pri}}) \leftarrow g_I$, record $\langle H_2, x_{\mathsf{pri}}, r_I \rangle$, $I++$.

Initialization

- On $(\textsc{Init}, sid, S)$ from $\mathcal{F}$, pick $k \leftarrow\!\!\$\, \mathbb{Z}_q, \mathsf{pk} := g^k$ s.t. $\mathsf{pk} \notin \mathsf{PK}$, record $\langle S, k, \mathsf{pk} \rangle$, insert pk in PK and return $(\textsc{Parameter}, sid, S, \mathsf{pk})$. Add pk to CPK if S is corrupted.

Server Compromise

- On $(\textsc{Compromise}, sid, S, \mathsf{pk}^*)$ from $\mathcal{A}$, pass it to $\mathcal{F}$. If $\mathsf{pk}^* = \mathsf{pk}$ (pk is S's public key), insert pk^* into CPK.

Online Evaluation

- On $(\textsc{Eval}, sid, ssid, C, S, x_{\mathsf{pub}})$ from $\mathcal{F}$, Sim sets $x \leftarrow g_J$, responds with $\mathsf{prfx} := (x_{\mathsf{pub}}, x)$ to $\mathcal{F}$, sends $(sid, ssid, x_{\mathsf{pub}}, x)$ to $\mathcal{A}$ as C's message to S, records $\langle ssid, C, x_{\mathsf{pub}}, r_J \rangle$ and sets $J++$.
- On $(\textsc{SvrComplete}, sid, ssid, S)$ from $\mathcal{F}$ and message $(sid, ssid, x_{\mathsf{pub}}, x^*)$ from $\mathcal{A}$, send $\mathsf{prfx} := (x_{\mathsf{pub}}, x^*)$ to $\mathcal{F}$, compute $y := e(H_1(x_{\mathsf{pub}}), x^*)^k$ and a proper NIZK proof ζ, send $(sid, ssid, y, \zeta)$ to $\mathcal{A}$ as S's response.
- On message $(sid, ssid, y^*, \zeta^*)$ and pk from $\mathcal{A}$ on behalf of S's response to C, recover $\langle ssid, C, x_{\mathsf{pub}}, r_J \rangle$.
  - If $\mathsf{pk} \in \mathsf{PK}$ and ζ is valid, check that $e(H_1(x_{\mathsf{pub}}), \mathsf{pk})^{r_J} = y^*$ only if $\mathsf{pk} \notin \mathsf{CPK}$. If not, abort and output $\textsc{Fail}$. Send $(\textsc{CltComplete}, sid, ssid, C, \mathsf{pk}, \top)$ to $\mathcal{F}$.
  - If $\mathsf{pk} \in \mathsf{PK}$, and ζ is invalid, send $(\textsc{CltComplete}, sid, ssid, C, \mathsf{pk}, \bot)$ to $\mathcal{F}$.
  - If $\mathsf{pk} \notin \mathsf{PK}$, record $\langle S, \cdot, \mathsf{pk} \rangle$, insert pk into PK, send $(\textsc{Parameter}, sid, S, \mathsf{pk})$ to $\mathcal{F}$, and send $(\textsc{CltComplete}, sid, ssid, C, \mathsf{pk}, \mathsf{flag})$ to $\mathcal{F}$ where $\mathsf{flag} = \bot/\top$ depending on where ζ is valid or invalid.

Stealing Password Data

- On $(\textsc{StealPwdFile}, sid, ssid, C)$ from $\mathcal{A}$, pass it to $\mathcal{F}$. On $\mathcal{F}$'s response, forward it to $\mathcal{A}$. In addition, if $\mathcal{F}$ returns x_{pri}, retrieve $\langle \mathsf{file}, sid, ssid, C, S, x_{\mathsf{pub}}, \cdot \rangle$ (record $\langle \mathsf{file}, sid, ssid, C, S, x_{\mathsf{pub}}, \bot \rangle$ if no such tuple). If the last item is $\bot$, change it to pw.
- On $(\textsc{OfflineTestPwd}, sid, ssid, S, x_{\mathsf{pub}}, x^*_{\mathsf{pri}})$ from $\mathcal{A}$, ignore this message if $\mathsf{pk} \notin \mathsf{CPK}$, else pass this message to $\mathcal{F}$. On $\mathcal{F}$'s response, forward it to $\mathcal{A}$. In addition, if $\mathcal{F}$ returns "correct guess", retrieve $\langle \mathsf{file}, sid, ssid, C, S, x_{\mathsf{pub}}, \cdot \rangle$, and if the last item is $\bot$, change it to x^*_{pri}.

Active Session Attacks

- On $(\textsc{TestPwd}, sid, ssid, C, S, x_{\mathsf{pub}}, x^*_{\mathsf{pri}})$ from $\mathcal{A}$, ignore this message if no record $\langle \mathsf{file}, sid, \cdot, C, S, x_{\mathsf{pub}}, \cdot \rangle$ marked $\textsc{compromised}$. Otherwise, send $(\textsc{TestPwd}, sid, ssid, C, S, x_{\mathsf{pub}}, x^*_{\mathsf{pri}})$ to $\mathcal{F}$ and pass $\mathcal{F}$'s response to $\mathcal{A}$.

Fig. 24: The simulator Sim for Pythia [11] ($\mathcal{F}_{\text{VPOPRF}}$ abbreviated $\mathcal{F}$).

E.1 The VPOPRF functionality $\mathcal{F}_{\mathrm{VPOPRF}}$

Following the ideal functionalities of VOPRF [19], OPRF [20], and adaptive OPRF [22], we define the VPOPRF functionality with adaptive corruptions, as shown in Fig. 22. The StealPwdFile message for stealing password files, the OfflineTeatPwd message for offline password guessing, and the TestPwd message for online password guessing, are not part of the VPOPRF functionality but are components of the VPOPRF-type PH functionality.

E.2 Review of Pythia [11]

Fig. 23 shows the Pythia protocol [11], only syntactically modified to realize the adaptive VPOPRF functionality.

E.3 Proof of security

We prove that Pythia UC-realizes our defined VPOPRF functionality $\mathcal{F}_{\mathrm{VPOPRF}}$, which answers the question left over from [11] on proving pseudo-randomness (i.e., what level of security is their PH protocol provably).

Theorem 8. *The protocol Pythia UC-realizes the functionality $\mathcal{F}_{\mathrm{VPOPRF}}$ with $H_1(\cdot), H_2(\cdot), H_{\mathsf{NIZK}_1}$ modeled as the random oracle.*

Precisely, for any adversary against Pythia, there is a simulated adversary that produces a view in the ideal world that no environment $\mathcal{E}$ can distinguish with an advantage better than $2n_q/q$ where n_q is the request number affected by adversary $\mathcal{A}$ and q is the order of the range $\mathbb{Z}_q$ of $H_{\mathsf{NIZK}_1}(\cdot)$.

Proof. We construct a simulator Sim, as shown in Fig. 24. Without loss of generality, we assume that $\mathcal{A}$ is a dummy adversary. Note that $x_{\mathsf{pub}} = w$ such that $w \leftarrow H(\mathsf{un}, s)$ and $x_{\mathsf{pri}} = \mathsf{pw}$ in Pythia. We prove Theorem 8 by showing that the probability of distinguishing between the real and ideal worlds from the environment $\mathcal{E}$'s view is negligible.

There is a difference in the environment's view through $\mathcal{A}$ between the real and simulated worlds. In the simulated world, on $\mathcal{A}$'s fresh query x_{pri} to H_2, the simulator Sim returns the next available element g_I of the random sequence, i.e., $H(x_{\mathsf{pri}}) := g^{r_I}$, and records $\langle H_2, x_{\mathsf{pri}}, r_I \rangle$. When $\mathcal{A}$ acts as an honest client and sends the Eval message with the same x_{pri} to $\mathcal{F}$, Sim will send g_J to $\mathcal{A}$, the next available element g_J of the different random sequence (i.e., I and J are independent of each other). The Sim prescribes different values (g_I and $g_J^{1/r}$) for the same hash queries in the two responses. However, since r is hidden from the $\mathcal{A}$'s view, $\mathcal{E}$ cannot rely on $\mathcal{A}$ to distinguish this inconsistency.

There is another difference in the environment's view through C between the real and ideal worlds. For the same Eval queries with $(x_{\mathsf{pub}}, x_{\mathsf{pri}})$, if the Fail event not happen, $\mathcal{F}$ returns a evaluated $F_{sid,\mathsf{pk}}(x_{\mathsf{pub}}, x_{\mathsf{pri}})$, a pseudo-random value from $\mathbb{G}_T$, and the real-world protocol Pythia returns $\rho \leftarrow e(H_1(x_{\mathsf{pub}}), H_2(x_{\mathsf{pri}}))^k$. Under the random oracle model, we program $H_1(\cdot)$ as a PRF function: on a fresh query x_{pub}, sample $m_1 \leftarrow\!\!\$\ \mathbb{Z}_q$ and record $\langle x_{\mathsf{pub}}, g_1^{m_1}\rangle$ where g_1 is a generator of $\mathbb{G}_1$; on an old query x_{pub}, retrieve $\langle x_{\mathsf{pub}}, g_1^{m_1}\rangle$ and output $g_1^{m_1}$. In the same way, $H_2(\cdot)$ is programmed as $\langle x_{\mathsf{pri}}, g_2^{m_2}\rangle$ such that g_2 is a generator of $\mathbb{G}_2$ and m_2 is a random value from $\mathbb{Z}_q$. Then we have $e(H_1(x_{\mathsf{pub}}), H_2(x_{\mathsf{pri}}))^k = (e(g_1, g_2)^k)^{m_1 \cdot m_2}$. Due to the non-degeneracy of e, there must be $e(g_1, g_2)^k \neq 1_{G_T}$; that is, $g_T \leftarrow e(g_1, g_2)^k$ is a generator of $\mathbb{G}_T$. Therefore, the evaluation function of Pythia is equivalent to the PRF function $\langle (x_{\mathsf{pub}}, x_{\mathsf{pri}}), g_T^m\rangle$, where $m = m_1 \cdot m_2$ is a random value from $\mathbb{Z}_q$. In the ideal world, $F_{sid,\mathsf{pk}}(\cdot)$ is defined as a PRF function. Thus, from environment $\mathcal{E}$'s view, the output difference is indistinguishable.

In addition, the Fail event only exists in the simulated world. If the Fail event happens, environment $\mathcal{E}$ will know that it is interacting with $\mathcal{F}$ and Sim in the ideal world. According to the definition of Sim, the Fail event means that the NIZK proof ζ is valid but $e(H_1(x_{\mathsf{pub}}), \mathsf{pk})^{r_I} \neq y$ (i.e., $e(H_1(x_{\mathsf{pub}}), x)^k \neq y$). This happens when the NIZK verification equation is falsely passed due to a collision of values chosen by the random oracle H_{NIZK_1} for different inputs, whose probability is $2/q$. Therefore, we can conclude that environment $\mathcal{E}$ can distinguish its views in two worlds with probability bound by $2/q$. Assume that there are n_q requests for the Eval message affected by adversary $\mathcal{A}$, the total distinguishability probability is at most $2n_q/q$.

E.4 A compiler from VPOPRF with SPF to VPOPRF without SPF

Our threshold failover solution can be applied to VPOPRF-type PHS that has the same update functionality as that of the PH defined in Fig. 3 to eliminate single points of failure. The formal definition based on $\mathcal{F}_{\mathrm{VPOPRF}}$ can be combined with TF defined in Fig. 9 through simple syntax modifications from reliable PH protocol provided in Fig. 12.

For reliable Pythia, where each of n servers holds a server key k and a share of the key token Δk such that $k' = k^{\Delta k}$, the client can failover by a threshold rotation protocol as follows after the on-duty server (that holds k) fails:

- Collect t token shares from other online servers through secure channels and reconstruct the key token Δk.
- Update all records evaluated by the on-duty server via $\rho' \leftarrow \rho^{\Delta k}$ locally.
- Active the off-duty server that holds k'.