Phoenix: Rebirth of a Cryptographic Password-Hardening Service

Russell W. F. Lai, ***Friedrich-Alexander-University Erlangen-Nürnberg, Chinese University of Hong Kong;*** **Christoph Egger and Dominique Schröder,** ***Friedrich-Alexander-University Erlangen-Nürnberg;*** **Sherman S. M. Chow,** ***Chinese University of Hong Kong***

https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lai

This paper is included in the Proceedings of the 26th USENIX Security Symposium

August 16–18, 2017 • Vancouver, BC, Canada

ISBN 978-1-931971-40-9

Open access to the Proceedings of the 26th USENIX Security Symposium is sponsored by USENIX

Phoenix: Rebirth of a Cryptographic Password-Hardening Service

Russell W.F. Lai
Friedrich-Alexander University Erlangen-Nürnberg, Chinese University of Hong Kong

Christoph Egger
Friedrich-Alexander University Erlangen-Nürnberg

Dominique Schröder
Friedrich-Alexander University Erlangen-Nürnberg

Sherman S.M. Chow
Chinese University of Hong Kong

Abstract

Password remains the most widespread means of authentication, especially on the Internet. As such, it is the Achilles heel of many modern systems. Facebook pioneered using external cryptographic services to harden password-based authentication in a large scale. Everspaugh *et al.* (Usenix Security '15) provided the first comprehensive treatment of such a service and proposed the Pythia PRF-Service as a cryptographically secure solution. Recently, Schneider *et al.* (ACM CCS '16) proposed a more efficient solution which is secure in a weaker security model.

In this work, we show that the scheme of Schneider *et al.* is vulnerable to offline attacks just after a single validation query. Therefore, it defeats the purpose of using an external crypto service in the first place and it should not be used in practice. Our attacks do not contradict their security claims, but instead show that their definitions are simply too weak. We thus suggest stronger security definitions that cover these kinds of real-world attacks, and an even more efficient construction, Phoenix, to achieve them. Our comprehensive evaluation confirms the practicability of Phoenix: It can handle up to 50% more requests than the scheme of Schneider *et al.* and up to three times more than Pythia.

1 Introduction

In spite of the research and development in authentication mechanisms such as public-key infrastructure or secure hardware tokens, the reality has shown that password-based authentication remains the most widespread means, especially on the Internet. As such, password-based authentication is the Achilles heel of many modern systems. Following a suggestion from the 70s, passwords are commonly stored as salted hash values. Yet, it is no longer adequate in the face of the increasing number of attacks. Prominent breaches of user accounts include Adobe, Yahoo, and much more [24]. The financial consequences are also dramatic. Verizon asked for a 1 **billion discount** on acquiring Yahoo [20] after knowing it had been hacked (1.5 billion+ accounts). We see an urgent need for action.

Obvious Weaknesses in Current Systems. Almost all web services store passwords as salted hash values as shown in Figure 1. The security of the

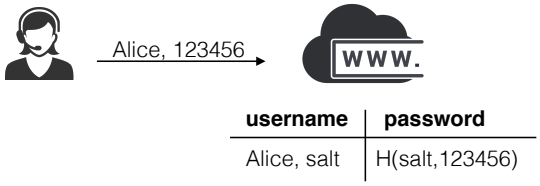

Figure 1: Password-based Authentication

passwords relies crucially on the assumption that the databases are kept secret from external attackers, and the internal administrators are trusted for not disclosing the databases or guessing the passwords themselves. However, the reality shows that databases get stolen. This is disastrous as passwords usually have low entropy and therefore can be guessed by a brute-force attack easily.

Under the aforementioned threat, there is a need for new solutions to protect passwords in a setting where the attacker has full access to the compromised service provider, including its secret keys and databases. It is not hard to see that any solution in which the web service can verify a given password alone is not viable, as a compromised service provider has all the knowledge (*e.g.*, secret keys) to carry out a brute-force guessing attack (*e.g.*, decrypting by the secret key of the web service) as in a

normal validation. Additional cryptographic mechanisms are needed to enhance security.

Moreover, an ideal solution should not change the infrastructure from the point of view of users. This is challenging as it rules out solutions which requires the end users to perform cryptographic operations.

External Password Hardening Services. A promising approach for the web service provider is to use external crypto services [4], where a crypto server carries out certain cryptographic operations, such as the computation of pseudorandom functions (PRF). Its general advantage is that it abstracts crypto away from developers, freeing them from the selection and implementation of suitable algorithms and the involved issue of key management.

Cryptographic PRF services are used in practice by Facebook [16] for password-based authentication. In this setting, the end-user Alice enters her user-

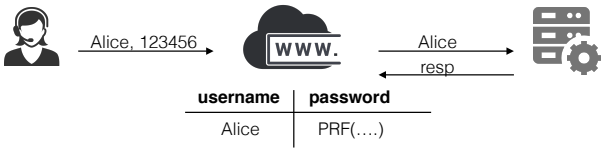

Figure 2: Password-based Authentication

name and the corresponding password into the web service as usual, as in Figure 2. The service provider no longer stores salted hash values, but only pseudorandom values which can only be computed with the help of the external PRF server, *i.e.*, the service provider acts as a client of the PRF service.

While used in practice, such kind of password hardening services did not receive much attention from the academia until the seminal work of Everspaugh *et al.* [9]. They formalize partially-oblivious PRFs (PO-PRF) with several security properties that conventional PRF services do not offer, with a fairly efficient construction, Pythia. Even if the web server is compromised and the database (of pseudorandom values) is stolen, brute-force offline attacks are no longer possible. The reason is that the adversary must interact with the PRF service to confirm a guess. The *partial obliviousness* ensures that the external crypto server does not learn the password but can still see the username when answering PRF requests. *Rate limiting* can thus be applied to make sure that an adversary cannot guess too many times. For incident response after key compromise, or to update the key proactively as a prudent practice, both the web server and the crypto service should be able to rotate their keys, without the end users noticing anything. Efficient *key rotation* [9, 21] means the amounts of communication and server computation are independent of the size of the database. It is an important security feature that cryptographic password hardening services must have [9, 21].

Using Pythia for password hardening is not without disadvantages. For example, it is only secure under a strong assumption [21], and is based on pairings, which is not as efficient as one can hope for. Very recently, Schneider *et al.* [21] claimed that all the properties expected by Pythia can be achieved by a weaker cryptographic primitive called partially-oblivious commitments (PO-COM). Using PO-COM, the PRF values are replaced with "enrollment records" which can be jointly computed by the client and the server via an enrollment protocol. The main difference lies in how a password is verified. Instead of jointly computing a PRF value, the client and the server engage in a validation protocol to verify whether a candidate password matches an enrollment record. Schneider *et al.* [21] also suggested a scheme that is twice as efficient as Pythia. Unfortunately, as we will show, their scheme is vulnerable to offline dictionary attacks. This motivates us to develop a new solution which is secure against such attacks while achieving even better efficiency.

1.1 Overview of Our Contribution

Formal security definitions are important even from a practical standpoint. They precisely describe what level of security can be achieved and serve as a basis for comparison between different solutions. Finding the "right" security definitions is challenging. They should be strong enough to cover all real-world attacks, but not to exclude efficient solution. In this work, we revisit the security notions of Everspaugh *et al.* [9] and Schneider *et al.* [21]. We argue that both fail to cover key rotation and rate limiting, while the latter even leaves room for practically-relevant attacks.

In response, we propose strengthened security definitions for password hardening schemes. Next, we propose a new construction, Phoenix, which is 1) extremely efficient, 2) reasonably simple, and 3) secure based on simple and well-known assumptions. With these properties, we believe that Phoenix may attract deployment interest. Below highlights our contributions in more details.

Formalizing Key Rotation. The literature [9, 21] highlights the importance of key rotation since it renders the old key useless and preserves the (forward-)security of the system as long as both par-

ties are not compromised simultaneously. Somewhat surprisingly, none of the existing security definitions take key rotation into account. To fill the gap, we formalize forward security of password hardening services, which captures the security guarantee in the presence of key rotation mechanisms.

Modeling Online Attacks. We argue that the definition of obliviousness (renamed to hiding in our work) given by Schneider *et al.* [21] is too weak. The property is supposed to protect the passwords when the client is compromised. Ideally, it should be guaranteed that the best attack strategy for guessing a password is to brute-force by repeatedly interacting with the crypto service online (modeled by the validation oracle in the security definition). Unfortunately, obliviousness, as defined by Schneider *et al.* [21], fails to capture this intuition, as its security experiment denies the adversary access to the validation oracle after receiving the challenge enrollment record. Indeed, permitting the adversary such accesses would allow it to trivially distinguish between two possible passwords. Moreover, by resorting to a crypto server to harden the passwords, one naturally expects it can perform rate limiting. While it is obviously a crucial feature, we are not aware of any definition which takes this into account. To resolve these issues, we suggest a "correct" security definition, which covers both online attacks and rate limiting. The latter is guaranteed by upper-bounding the advantage of the adversary by the loss of entropy in guessing and validation.

Attacking against Schneider et al. [21]. The shortcoming of the obliviousness definition by Schneider *et al.* [21] is not just a definitional deficiency. We detail how to perform highly efficient (essentially only one exponentiation for each trial) offline dictionary or direct attacks against their scheme by just a single interaction with the crypto server! We stress that our attack is *outside of the security model* of Schneider *et al.* [21]. Below we only show part of the scheme which matters in the attack.

In their scheme, an enrollment record, stored by the client $\mathcal{C}$ using the crypto service provided by $\mathcal{S}$, can be seen as an ElGamal encryption under a secret key s_x of $\mathcal{S}$, in the form of $(T_1, T_2) = (g^y, g^{ys_x} \cdot \mathsf{pw}^{\mathsf{sk}_\mathcal{C}}) \in \mathbb{G}^2$, where $\mathbb{G}$ is a (multiplicative) finite cyclic group. To validate that this record corresponds to a password pw for some username un, $\mathcal{C}$ sends (T_1, un, v) for $v = \mathsf{pw}^{r \cdot \mathsf{sk}_\mathcal{C}}$. *Without any validity checking*, $\mathcal{S}$ returns π_2, a zero-knowledge proof of s_x with respect to (g^y, g^{ys_x}). This opens the door for the following generic offline dictionary attack, without exploiting the structure of the zero-knowledge proof: An adversary $\mathcal{A}$ who compromised $\mathcal{C}$ (and hence obtained $\mathsf{sk}_\mathcal{C}$ and (T_1, T_2)) sends (g^y, un, h) to $\mathcal{S}$ where h is a random group element independent of any passwords. After getting π_2 from $\mathcal{S}$, $\mathcal{A}$ can then try different passwords pw by testing if the proof π_2 is for $(T_1, T_2/\mathsf{pw}^{\mathsf{sk}_\mathcal{C}})$. This is doable since the entropy of pw is assumed to be low. By further exploiting the structure of the specific instantiation of the proof, the adversary can even extract the password directly: It first extracts the value g^{ys_x} from the proof π_2, then computes $\mathsf{pw} = (T_2/g^{ys_x})^{1/\mathsf{sk}_\mathcal{C}}$. We conclude that one *must not use* the scheme of Schneider *et al.*, as our attack defeats its purpose of using an external crypto service.

Reviving the Broken Scheme. In the spirit of providing password hardening services using a weaker tool than PO-PRFs [21], we present PHOENIX, a conceptually simple construction from standard cryptographic primitives. It achieves two seemingly contradicting goals: a high security level without sacrificing the efficiency. Our scheme can, in fact, handle roughly 50% more request per second than that of Schneider *et al.* [21], and three times more than PYTHIA [9]. Since the scheme of Schneider *et al.* [21] is vulnerable to the one validation-query offline dictionary attacks, ours is the first efficient and fully secure solution based on standard decisional Diffie-Hellman assumption.

1.2 Notations

Let $\lambda \in \mathbb{N}$ be the security parameter. By $x \leftarrow\!\!\$\, S$ we denote the uniform drawing of a random element x from set S. Unless stated otherwise, all algorithms run in probabilistic polynomial time (PPT). $x \leftarrow\!\!\$\, \mathcal{A}(y)$ denotes the event that $\mathcal{A}$ on input y outputs x. If $\mathcal{A}$ is deterministic we write $x \leftarrow \mathcal{A}(y)$ instead. For two PPT interactive algorithms $\mathcal{A}, \mathcal{B}$, we denote by $(a, b) \leftarrow\!\!\$\, \langle \mathcal{A}(x), \mathcal{B}(y) \rangle_X$ the event that $\mathcal{A}$ and $\mathcal{B}$ engage in the protocol X on input x and y, and produce local outputs a and b, respectively. If there is only one output, then it is assumed to be for $\mathcal{A}$. We write $\mathcal{B}^{\langle \mathcal{A}(x), \cdot \rangle}(y)$ if $\mathcal{B}$ can invoke an unbounded number of executions of the interactive protocol with $\mathcal{A}$ in an arbitrarily interleaved order.

2 Crypto Password Hardening (PH)

Both previous works formulated cryptographic primitives [9, 21] which were supposed to cover the properties of cryptographic password hardening (PH) services. We do not follow this approach. Instead,

we define PH directly, which is simpler and more natural. A direct definition removes the need for bridging the security requirements of the underlying primitives to those expected by PH (*e.g.*, the main feature of key rotation seems to make more sense in PH than in the underlying commitment [21]).

2.1 Overview

Our formalization of PH is closely related to the definition of partially-oblivious commitments (POCOM) defined by Schneider *et al.* [21], with the main difference being that we consider key rotation in all security definitions. Roughly speaking, a PH scheme PH is a two party protocol that is partitioned in phases. The first phase is the setup phase, in which a client $\mathcal{C}$ and the server $\mathcal{S}$ set up their public and secret keys individually without communication. Each phase after the first is either an enrollment, a validation, or a key rotation phase, in an arbitrary order.

In an *enrollment* phase, the client and the server cooperate to generate an enrollment record T for a username un, and a password pw, where un is an input available for both and pw is a private input from the client. The client then stores the record T.

Subsequently, in a *validation* phase, the client can interact with the server to verify if a pair $(\mathsf{un}, \mathsf{pw})$ is stored in a record T. Similar to the enrollment phase, un is a common input and pw is a private input from the client. We note that in the original syntax [21], while un is not an input of the server, it is supposed to be revealed to the server during the interactions in the protocol for rate limiting.

Suppose an adversary, who may have knowledge of some enrollment records, compromises either the client or the server secret key. It can then act as the compromised party and interacts with the other in the protocols to figure out the underlying passwords of the enrollment records. As soon as the incident is discovered, the (true) client and the (true) server communicate to refresh their keys and all enrollment records. Instead of regenerating them from scratch, they enter a *key rotation* phase to update their secret keys. In addition to an updated client secret key, the client also obtains some auxiliary information, using which it is able to update each enrollment record locally, without further communicating with the server, nor knowing the underlying password of the record. Note that our syntax of the key rotation phase is significantly different. In the original definition [21], the key rotation protocol updates a single enrollment record instead of all records stored by the client. We believe that this was an oversight.

2.2 Definition of PH

We provide a formal definition of cryptographic password hardening schemes. Some algorithms in our formalization get as input some auxiliary input, such as a random session identifier. Under normal circumstances, the auxiliary information is an empty string denoted by ϵ. Non-empty auxiliary information is only used in defining forward-security.

Definition 1 (PH) *Let $\mathcal{U}$ and $\mathcal{P}$ be the username and password space respectively. A* cryptographic password hardening service PH *consists of the efficient algorithms* $(\mathsf{Setup}, \mathsf{KGen}_{\mathcal{C}}, \mathsf{KGen}_{\mathcal{S}}, \langle \mathcal{C}, \mathcal{S} \rangle_{\mathtt{enrl}}, \langle \mathcal{C}, \mathcal{S} \rangle_{\mathtt{val}}, \langle \mathcal{C}, \mathcal{S} \rangle_{\mathtt{rot}}, \mathsf{Udt})$*, to be executed in four phases:*

Setup Phase. *On input the security parameter λ,* $\mathsf{Setup}(1^{\lambda})$ *outputs the public parameter* pp*. On input the public parameter* pp*, the client runs* $\mathsf{KGen}_{\mathcal{C}}(\mathsf{pp})$ *to generate a client public key* $\mathsf{pk}_{\mathcal{C}}$*, and a client secret* $\mathsf{sk}_{\mathcal{C}}$*, while the server runs* $\mathsf{KGen}_{\mathcal{S}}(\mathsf{pp})$ *to generate a server public key* $\mathsf{pk}_{\mathcal{S}}$*, a server secret* $\mathsf{sk}_{\mathcal{S}}$*. All parties will take as input the public parameter* pp*, the client public key* $\mathsf{pk}_{\mathcal{C}}$*, and the server public key* $\mathsf{pk}_{\mathcal{S}}$ *in all subsequent protocols.*

Enrollment Phase. *In the enrollment protocol* $\langle \mathcal{C}(\mathsf{sk}_{\mathcal{C}}, \mathsf{un}, \mathsf{pw}, \mathsf{aux}), \mathcal{S}(\mathsf{sk}_{\mathcal{S}}, \mathsf{un}, \mathsf{aux}) \rangle_{\mathtt{enrl}}$*, the client inputs its secret key* $\mathsf{sk}_{\mathcal{C}}$*, a username* $\mathsf{un} \in \mathcal{U}$*, a password* $\mathsf{pw} \in \mathcal{P}$*, and some auxiliary information* aux*. The server inputs its secret key* $\mathsf{sk}_{\mathcal{S}}$*, a username* un*, and some auxiliary information* aux*. The client outputs an enrollment record T, while the server outputs nothing. We say that the enrollment record T stores the tuple* $(\mathsf{un}, \mathsf{pw})$*. The client stores the tuple* (T, un)*, and securely deletes the password* pw *and all intermediate values that are computed locally or obtained from the server. The server is also supposed to delete all intermediate values.*

Validation Phase. *In the validation protocol* $\langle \mathcal{C}(\mathsf{sk}_{\mathcal{C}}, T, \mathsf{un}, \mathsf{pw}), \mathcal{S}(\mathsf{sk}_{\mathcal{S}}, \mathsf{un}) \rangle_{\mathtt{val}}$*, the client inputs its secret key* $\mathsf{sk}_{\mathcal{C}}$*, an enrollment record T, a username* $\mathsf{un} \in \mathcal{U}$*, and a password* $\mathsf{pw} \in \mathcal{P}$*. The server inputs its secret key* $\mathsf{sk}_{\mathcal{S}}$*, and a username* un*. The client outputs a decision $b \in \{0,1\}$ of whether T stores the tuple* $(\mathsf{un}, \mathsf{pw})$*, while the server outputs nothing.*

Key Rotation Phase. *In the key rotation protocol* $\langle \mathcal{C}(\mathsf{sk}_{\mathcal{C}}), \mathcal{S}(\mathsf{sk}_{\mathcal{S}}) \rangle_{\mathtt{rot}}$*, the client and the server input their secret keys* $\mathsf{sk}_{\mathcal{C}}$ *and* $\mathsf{sk}_{\mathcal{S}}$ *respectively. The client outputs an updated client public key* $\mathsf{pk}'_{\mathcal{C}}$*, an updated client secret key* $\mathsf{sk}'_{\mathcal{C}}$ *and an update token τ. The server outputs an updated server public key*

$\mathsf{pk}'_{\mathcal{S}}$, *and an updated secret key* $\mathsf{sk}'_{\mathcal{S}}$. *Using the update token* τ, *the client runs the update algorithm* $\mathsf{Udt}(\tau, T, \mathsf{un})$ *to update each of the old enrollment records* T *into new ones* T'.

Correctness. We require that all honestly generated enrollment records can pass validation. Formally, a cryptographic password hardening service scheme PH is *correct* if for all security parameter $\lambda \in \mathbb{N}$, public parameters $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$, key pairs $(\mathsf{pk}_{\mathcal{C}}, \mathsf{sk}_{\mathcal{C}}) \in \mathsf{KGen}_{\mathcal{C}}(\mathsf{pp})$ and $(\mathsf{pk}_{\mathcal{S}}, \mathsf{sk}_{\mathcal{S}}) \in \mathsf{KGen}_{\mathcal{S}}(\mathsf{pp})$, username $\mathsf{un} \in \mathcal{U}$, password $\mathsf{pw} \in \mathcal{P}$, enrollment records $T \in \langle \mathcal{C}(\mathsf{sk}_{\mathcal{C}}, \mathsf{un}, \mathsf{pw}, \epsilon), \mathcal{S}(\mathsf{sk}_{\mathcal{S}}, \mathsf{un}, \epsilon) \rangle_{\mathtt{enrl}}$, it holds that $\langle \mathcal{C}(\mathsf{sk}_{\mathcal{C}}, T, \mathsf{un}, \mathsf{pw}), \mathcal{S}(\mathsf{sk}_{\mathcal{S}}, \mathsf{un}) \rangle_{\mathtt{val}} = 1$. Note that it is unnecessary to define the correctness of key rotation, as it will be captured by forward security to be introduced below.

2.3 Security of PH

Our security definitions are fundamentally different from, and arguably stronger than, the originals [21]. In particular, our notions cover important real-world attacks and ensures security in the presence of key rotation, as discussed in the introduction. In the following, we first give an overview of our definitions, and discuss the differences in details in Section 2.4.

A cryptographic password hardening service is required to be (partially) oblivious, hiding, binding, and forward secure. Roughly speaking, partial oblivious means that it is infeasible, even for a malicious server, to tell which password pw is used by the client in the enrollment and validation protocols. It is partial in the sense that the username un can be revealed. In fact, un is required to be revealed to the server for rate limiting. We therefore simply let un be a common input for both parties in the enrollment and the validation protocols.

Hiding means that, given the client secret key $\mathsf{sk}_{\mathcal{C}}$, a username un, and an enrollment record T of $(\mathsf{un}, \mathsf{pw})$ for some hidden password pw, the best strategy of any adversary to guess pw is by launching an *online* dictionary attack which requires interaction with the server via the validation protocol.

Binding requires that it is computationally infeasible, even for a malicious server, to convince the client that an enrollment record T is valid for two distinct pairs $(\mathsf{un}, \mathsf{pw})$ and $(\mathsf{un}', \mathsf{pw}')$.

Forward security means that compromising either the client or the server secret key does not help the adversary to determine the underlying password of an enrollment record. We formalize this intuition in an even stronger property. It requires that even if both the client and server secret keys are adversarially generated, the updated keys and enrollment records are indistinguishable from the freshly generated ones. This formalization is simpler because we do not need to argue about the security of secret keys which can be rotated for many times.

2.3.1 Partial Obliviousness

Partial obliviousness protects against a malicious server that wishes to learn the password pw behind an enrollment record after observing its creation and several validations. The property is partial since it does not guarantee anything about the secrecy of the username un. In fact, in the syntax defined above, we let the client reveal the username un to the server explicitly by regarding un as a common input.

Technically, we consider a security experiment played between a challenger acting as the client and an adversary acting as the malicious server. The challenger generates the client secret key and keeps it secret (Line 1). Furthermore, it simulates executions of the enrollment, validation, and key rotation protocol, where only the client secret key input is fixed (Line 2). The adversary can provide all other client inputs, as well as the server side code. The embedded client secret key can be updated by executing the key rotation protocol. The client outputs of all protocol executions, except for $\mathsf{sk}'_{\mathcal{C}}$ from the key rotation protocol, are given to the adversary.

The experiment then proceeds in two stages, a learning phase and a challenge phase. In the learning phase, the adversary is free to interact with the challenger in the above protocols. At the end of this phase, the adversary outputs a username un^*, and two passwords pw_0^* and pw_1^* (Line 3). It will then be challenged on one of the passwords and the attacker has to guess the password. Formally, the challenger generates the challenge record T^* (for the password pw_b) together with the adversary (line 7). In addition to the previous protocols, the adversary gets access to an additional embedded-password validation protocol, which embeds either $(\mathsf{un}^*, \mathsf{pw}_0^*)$ or $(\mathsf{un}^*, \mathsf{pw}_1^*)$ (Lines 8). Note that the adversary may query the (normal) enrollment and validation protocol on most username-password pairs, and the protocols only return $\perp$ for the pairs $(\mathsf{un}^*, \mathsf{pw}_{b''}^*)$ for $b'' \in \{0,1\}$ (Lines 10 and 11) to avoid it from winning trivially. Finally, the adversary outputs b' guessing which tuple is embedded (Line 9).

Definition 2 (Partial Obliviousness) *A cryptographic password hardening service* PH *is* partially oblivious *if, for any three-stage* PPT *adversary*

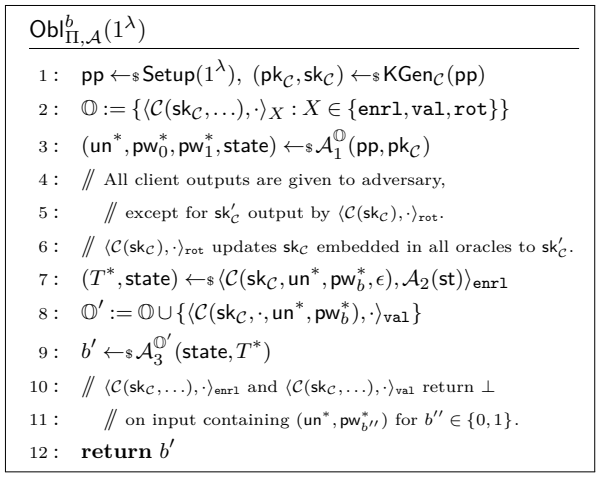

Figure 3: Partial Obliviousness Experiment

$\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr\left[\mathsf{Obl}^0_{\Pi,\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Obl}^1_{\Pi,\mathcal{A}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

where the randomness is taken over the random coins of the experiments and the adversary. Figure 3 defines the two experiments.

2.3.2 Hiding

The hiding property protects the passwords from an adversary who compromises the client, learns its secret key and all enrollment records, and wishes to learn the underlying password behind one of the records. We formalize this intuition by letting the adversary play the client role in the enrollment, validation, and key rotation protocols.

Inevitably, since passwords are assumed to have low entropy, the adversary always succeeds if it attempts to validate the target record with all possible passwords. Our formulation covers this fact by adjusting the success determination accordingly. To explain our idea, consider the following experiment: The challenger chooses a random password. The adversary is given access to a magical oracle which, when given a guess, answers whether the guess equals the chosen password. Suppose that only Q guesses are allowed. Obviously, the best strategy of the adversary is to asks for the Q most probable passwords. If one of them returns true, then the adversary wins by outputting that password. Otherwise, its best strategy is to output the most probable password which is not yet guessed, *i.e.*, the $(Q+1)$-th most probable password. Since the adversary can use the server as the magical oracle by interacting with it in the validation protocol, the best we can hope for is that the adversary cannot perform significantly better than the above strategy.

Technically, we consider a security experiment (see Figure 4) played between a challenger acting as the server and an adversary acting as the malicious client. The challenger generates the server secret key honestly and keeps it secret (Line 1). The adversary can interact with the challenger in the enrollment, validation, and key rotation protocols using arbitrary client side codes (Line 2). Eventually, the adversary outputs a client secret key $\mathsf{sk}_\mathcal{C}$, a username un^*, and a distribution χ of passwords (Line 6). The distribution models the real-world situations where the passwords to be protected are not uniformly random in $\{0,1\}^\lambda$ but instead follow a certain distribution possibly with low entropy, which might be known by the adversary. The challenger then chooses a random password pw^* (Line 7) from the distribution χ and computes a fresh challenge enrollment record T^* for the tuple $(\mathsf{un}^*, \mathsf{pw}^*)$ using the honest client and server code (Line 8). The challenger sends T^* to the adversary (Line 9). The adversary can continue to interact with the server and finally outputs pw'. It wins if pw^* is equal to pw'.

Using the above strategy, the adversary wins with probability at least $\sum_{i=1}^{Q+1} p_i$, where Q is the number of times the validation oracle is queried on inputs containing un^*, and p_i is the i-th most probable event in χ. We therefore require that the success probability of the adversary be negligibly close to $\sum_{i=1}^{Q+1} p_i$. We remark that similar bounds are used in the context of password-authenticated key exchange [3].

Definition 3 (Hiding) *A cryptographic password hardening service* PH *is* hiding *if, for any two-stage* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr\left[\mathsf{Hiding}_{\Pi,\mathcal{A}}(1^\lambda) = 1\right] \leq \sum_{i=1}^{Q+1} p_i + \mathsf{negl}(\lambda)$$

where the randomness is taken over the random coins of the experiment and the adversary, p_i *is the probability of the* i*-th most probable event in the distribution* χ *specified by* $\mathcal{A}_1$ *in the experiment, and* Q *is the number of times* $\langle \cdot, \mathcal{S}(\mathsf{sk}_\mathcal{S}, \cdot)\rangle_{\mathtt{val}}$ *is queried by* $\mathcal{A}_2$ *on server input* un^**. Figure 4 defines the experiment* $\mathsf{Hiding}_{\Pi,\mathcal{A}}$*.*

2.3.3 Binding

Similar to commitments, binding guarantees it is infeasible to open an enrollment record into two dis-

$\mathsf{Hiding}_{\Pi,\mathcal{A}}(1^\lambda)$

$\mathsf{pp} \leftarrow\!\!\$\, \mathsf{Setup}(1^\lambda)$, $(\mathsf{pk}_\mathcal{S}, \mathsf{sk}_\mathcal{S}) \leftarrow\!\!\$\, \mathsf{KGen}_\mathcal{S}(\mathsf{pp})$
$\mathbb{O} := \{\langle \cdot, \mathcal{S}(\mathsf{sk}_\mathcal{S}, \ldots)\rangle_X : X \in \{\mathtt{enrl}, \mathtt{val}, \mathtt{rot}\}\}$
// All server outputs are given to adversary,
  // except for $\mathsf{sk}'_\mathcal{S}$ output by $\langle \cdot, \mathcal{S}(\mathsf{sk}_\mathcal{S})\rangle_{\mathtt{rot}}$.
// $\langle \cdot, \mathcal{S}(\mathsf{sk}_\mathcal{S})\rangle_{\mathtt{rot}}$ updates $\mathsf{sk}_\mathcal{S}$ embedded in all oracles to $\mathsf{sk}'_\mathcal{S}$.
$(\mathsf{sk}_\mathcal{C}, \mathsf{un}^*, \chi, \mathsf{state}) \leftarrow\!\!\$\, \mathcal{A}_1^{\mathbb{O}}(\mathsf{pp}, \mathsf{pk}_\mathcal{S})$
$\mathsf{pw}^* \leftarrow\!\!\$\, \chi$
$T^* \leftarrow\!\!\$\, \langle \mathcal{C}(\mathsf{sk}_\mathcal{C}, \mathsf{un}^*, \mathsf{pw}^*, \epsilon), \mathcal{S}(\mathsf{sk}_\mathcal{S}, \mathsf{un}^*, \epsilon)\rangle_{\mathtt{enrl}}$
$\mathsf{pw}' \leftarrow\!\!\$\, \mathcal{A}_2^{\mathbb{O}}(\mathsf{state}, T^*)$
return $(\mathsf{pw}^* = \mathsf{pw}')$

Figure 4: Hiding Experiment

tinct passwords. In our setting, however, the enrollment record is never opened but only validated. The binding property in this context prevents a malicious server from convincing the client that an enrollment record T^* is valid for distinct tuples $(\mathsf{un}_0^*, \mathsf{pw}_0^*)$ and $(\mathsf{un}_1^*, \mathsf{pw}_1^*)$. Since by the correctness requirement if T is the enrollment record for $(\mathsf{un}, \mathsf{pw})$, then $(T, \mathsf{un}, \mathsf{pw})$ must pass validation. Thus, the binding property implicitly guarantees that a malicious server can never convince the client that an invalid enrollment record is valid.

Technically, we consider a security experiment played between a challenger acting as the client and an adversary acting as the malicious server. At the beginning, the adversary outputs a client secret key $\mathsf{sk}_\mathcal{C}$, an enrollment record T^*, and a tuple $(\mathsf{un}_0^*, \mathsf{pw}_0^*)$. The challenger and the adversary then interact in the validation protocol to validate the tuple $(T^*, \mathsf{un}_0^*, \mathsf{pw}_0^*)$, where the adversary can use arbitrary server-side code. After observing the communication transcript, the adversary outputs another tuple $(\mathsf{un}_1^*, \mathsf{pw}_1^*)$. It interacts with the challenger again to validate the tuple $(T^*, \mathsf{un}_1^*, \mathsf{pw}_1^*)$. The adversary wins if $(\mathsf{un}_0^*, \mathsf{pw}_0^*)$ and $(\mathsf{un}_1^*, \mathsf{pw}_1^*)$ are distinct and both validations output 1. We require that the probability of this happening is negligible.

Definition 4 (Binding) *A cryptographic password hardening service* PH *is* binding *if, for any four-stage* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4)$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr\left[\mathsf{Binding}_{\Pi,\mathcal{A}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

where the randomness is taken over the random coins of the experiment and the adversary. Figure 5 defines the binding experiment.

$\mathsf{Binding}_{\Pi,\mathcal{A}}(1^\lambda)$

$(\mathsf{pk}_\mathcal{S}, \mathsf{sk}_\mathcal{C}, T^*, \mathsf{un}_0^*, \mathsf{pw}_0^*, \mathsf{state}) \leftarrow\!\!\$\, \mathcal{A}_1(1^\lambda)$
$(b_0, \mathsf{state}) \leftarrow\!\!\$\, \langle \mathcal{C}(\mathsf{sk}_\mathcal{C}, T^*, \mathsf{un}_0^*, \mathsf{pw}_0^*), \mathcal{A}_2(\mathsf{state})\rangle_{\mathtt{val}}$
$(\mathsf{un}_1^*, \mathsf{pw}_1^*, \mathsf{state}) \leftarrow\!\!\$\, \mathcal{A}_3(\mathsf{state})$
$(b_1, \mathsf{state}) \leftarrow\!\!\$\, \langle \mathcal{C}(\mathsf{sk}_\mathcal{C}, T^*, \mathsf{un}_1^*, \mathsf{pw}_1^*), \mathcal{A}_4(\mathsf{state})\rangle_{\mathtt{val}}$
$b_2 \leftarrow ((\mathsf{un}_0^*, \mathsf{pw}_0^*) \neq (\mathsf{un}_1^*, \mathsf{pw}_1^*))$
return $b_0 \wedge b_1 \wedge b_2$

Figure 5: Binding Experiment

2.3.4 Forward Security

Intuitively, the key rotation should render an old client or server key useless to the adversary. Further, an old client or server secret key should not help in recovering information from an updated enrollment record. To formalize this intuition, one possible but complicated way is to define a security experiment which gives the adversary accesses of a special key rotation oracle apart from the usual enrollment and validation oracles. The key rotation oracle leaks either the client or the server secret key to the adversary, and at the same time rotates the old keys to the new ones. The goal of the adversary is to find out the underlying password of an enrollment record. Alternatively, we consider a simpler definition based on the intuition that the rotated keys and enrollment records are indistinguishable from freshly generated ones.

Technically, we consider a security experiment played between a challenger, acting as both the client and the server, and an adversary acting as a malicious outsider. At the beginning, the adversary outputs both secret keys $\mathsf{sk}_\mathcal{C}$ and $\mathsf{sk}_\mathcal{S}$, and a valid tuple $(T, \mathsf{un}, \mathsf{pw})$ under the specified keys. This models the situations where the adversary somehow obtains the secret keys which might be rotated many times. The challenger then either rotates the keys and updates the enrollment record, or samples a new pair of keys and generates a fresh enrollment record for $(\mathsf{un}, \mathsf{pw})$, using some auxiliary information $\mathsf{aux} = \mathcal{L}(T)$, for some leakage function $\mathcal{L}$. The updated or fresh keys and enrollment record are then sent to the adversary, who must guess how those are produced. We require the probability of the adversary guessing correctly to be negligible.

Definition 5 (Forward Security) *Let* $\mathcal{L}$ *be a leakage function which maps an enrollment record* T *to some auxiliary information* aux*. A cryptographic password hardening service* PH *is* $\mathcal{L}$-forward secure *if for any two-stage* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

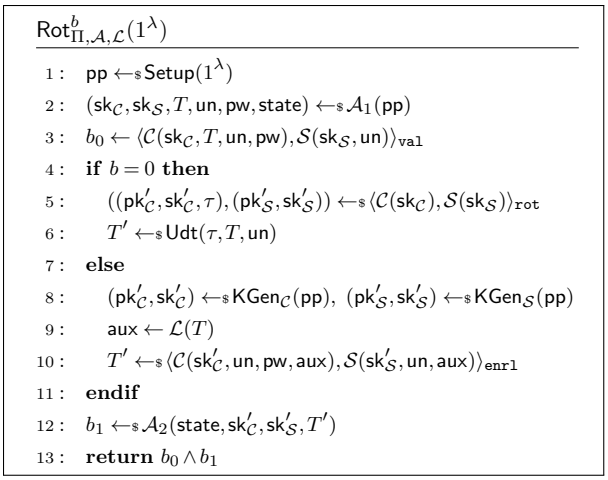

Figure 6: $\mathcal{L}$-Forward Security Experiment

$$\left|\Pr\left[\mathsf{Rot}^0_{\Pi,\mathcal{A},\mathcal{L}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Rot}^1_{\Pi,\mathcal{A},\mathcal{L}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

where the randomness is taken over the random coins of the experiments and the adversary. Figure 6 defines the two experiments.

2.4 Comparison with the Definitions of Schneider et al. [21]

We comprehensively explain the differences between our definitions and those of Schneider *et al.* [21]. We argue that ours either capture the intended security features better, or imply their counterparts.

2.4.1 Partial Obliviousness

The property was introduced in the name "partially hiding", which we believe is an oversight since the primitive was called "partially-oblivious" commitment schemes. The adversary in the original security experiment is stronger such that it generates both the client and server secret keys. However, it is also weaker in other abilities:

- Their embedded-password validation oracle embeds an enrollment record T output by the adversary. In contrast, ours allow the adversary to query on any enrollment record T.
- The client secret key embedded in the oracles is fixed. The adversary cannot instruct the challenger to rotate it into a new one.
- The adversary does not learn the client outputs from the embedded-password oracles. We think this violates the general philosophy of cryptographic definitions where, for most of the time, only the secret keys are assumed to be hidden from the adversary.

2.4.2 Hiding

The property was introduced in the name of "obliviousness", which we believe is an oversight since obliviousness is supposed to be a security property against a malicious server. However, the original security experiment models a malicious client, who is trying to figure out the underlying password pw of a given enrollment record T.

Recall that the whole point of introducing PO-PRF and PO-COM is to prevent against offline dictionary attacks. The idea is that, even given the client secret key $\mathsf{sk}_\mathcal{C}$ and the enrollment record T for some username un, the adversary can only guess the underlying password pw one at a time with the aid of the server. This rate-limits validation queries based on the username un. Assuming there is enough entropy in the password pw, the adversary is unable to recover pw before the limited number of validation query quota is used up. Curiously, the definition of Schneider *et al.* [21] does not model such an attack: In the second stage, after specifying the challenge passwords pw_0 and pw_1, the adversary is no longer given access to the validation oracle. If they allow the adversary to query the validation oracle even once in this stage, the adversary can win trivially by simply querying the oracle with either pw_0 or pw_1.

We fix this issue by requiring the adversary to specify a distribution χ of passwords instead of just two, and allowing it to query the validation oracle as many times as it wants. The resistance against offline dictionary attacks is then modeled by the success probability of the adversary: We require that the adversary must only be able to rule out at most one possible password from each query to the validation oracle. Finally, note that the adversary in the original definition cannot access any rotation oracle.

2.4.3 Binding

In the original definition, the binding property is only guaranteed for honestly generated keys and honestly validated enrollment records. It is not clear what this security guarantee means in the context of password hardening, the main motivating application. We thus make the following changes with a malicious server in mind. First, although it is not necessary for the context of password hardening, we let the adversary provide the client secret key. Second, it does not output the two pairs $(\mathsf{un}_0, \mathsf{pw}_0)$ and $(\mathsf{un}_1, \mathsf{pw}_1)$ right away. Instead, it first outputs

$(\mathsf{un}_0, \mathsf{pw}_0)$, waits until the validation protocol is executed on this pair, and then adaptively outputs the second pair. Third, the server-side code for validating the two pairs $(\mathsf{un}_0, \mathsf{pw}_0)$ and $(\mathsf{un}_1, \mathsf{pw}_1)$ is provided by the adversary. This formulation makes more sense in the context of password hardening. It models a malicious server which is trying to convince the client that an enrollment record is valid for two pairs $(\mathsf{un}_0, \mathsf{pw}_0)$ and $(\mathsf{un}_1, \mathsf{pw}_1)$, whereas at least one of which must be invalid.

3 Phoenix

We propose a conceptually simple, almost generic construction, PHOENIX, based on (partially) homomorphic encryption and pseudorandom functions.

In the enrollment protocol, $\mathcal{S}$ receives a username un. It returns $h_{\mathcal{S}} \leftarrow \mathsf{PRF}_{k_{\mathcal{S}}}(\mathsf{un}, n_{\mathcal{S}})$ for some random nonce $n_{\mathcal{S}}$. $\mathcal{C}$ computes $h_{\mathcal{C}} \leftarrow \mathsf{PRF}_{k_{\mathcal{C}}}(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}})$ for another random nonce $n_{\mathcal{C}}$, and encrypts the product $h_{\mathcal{S}} \cdot h_{\mathcal{C}}$ under the server public key. Then, it stores the ciphertext as the enrollment record of $(\mathsf{un}, \mathsf{pw})$ in its database, and securely deletes the password pw and all intermediate values computed locally or received from the server. $\mathcal{S}$ should also delete all its intermediate values.

To validate a candidate password pw', $\mathcal{C}$ computes the pseudorandom value $h'_{\mathcal{C}} \leftarrow \mathsf{PRF}_{k_{\mathcal{C}}}(\mathsf{un}, \mathsf{pw}', n_{\mathcal{C}})$, and performs a homomorphic operation on the ciphertext such that it now encrypts the product $h_{\mathcal{S}} \cdot h_{\mathcal{C}} / h'_{\mathcal{C}}$. It then sends the resulting ciphertext to $\mathcal{S}$, who attempts to decrypt it. Suppose the candidate password is correct, the term $h_{\mathcal{C}}$ is canceled out, and $\mathcal{S}$ is left with a ciphertext of $h_{\mathcal{S}}$. $\mathcal{S}$ thus verifies whether the message obtained from decryption equals $h_{\mathcal{S}}$, and proves the correctness of the decryption if so. $\mathcal{C}$ is convinced that the candidate password is correct if and only if the proof is valid.

To support key rotation, we need key homomorphism in addition to message homomorphism. We thus instantiate the above generic construction with an encryption scheme which is inspired by ElGamal [8] and Cramer-Shoup [6], and the pseudorandom function $\mathsf{PRF}_k(\cdot) = H(\cdot)^k$ [17], where $k \in \mathbb{Z}_q$ and the hash function H is modeled as a random oracle. We cannot use ElGamal or Cramer-Shoup directly, as the former is only CPA-secure (so it is difficult to simulate the validation oracle in the security reduction) while the latter (or any CCA-secure scheme in general) is not homomorphic. Interestingly, with such an instantiation, an enrollment record is an encryption of $H_{\mathcal{S}}(\mathsf{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}} \cdot H_{\mathcal{C}}(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}})^{k_{\mathcal{C}}}$, from which we can draw connection to PYTHIA [9], in which the record is computed as $e(H_{\mathcal{S}}(\mathsf{un}), H_{\mathcal{C}}(\mathsf{pw}))^{k_{\mathcal{S}}}$. The pairing function $e(\cdot,\cdot)$ is used in PYTHIA mainly for partial blinding by the client, *i.e.*, blinding pw but not un. In our construction, the server only evaluates the PRF on the username un but not the password pw, which perhaps explains why we do not need pairing.

$\mathsf{Setup}(1^\lambda)$	$\mathsf{KGen}_{\mathcal{S}}(\mathsf{pp})$
$\mathsf{crs} \leftarrow\!\!\$\, \Pi.\mathsf{Gen}(1^\lambda),\ g \leftarrow\!\!\$\, \mathbb{G}$	$s, x, y, k_{\mathcal{S}} \leftarrow\!\!\$\, \mathbb{Z}_q$
return (crs, g)	$h \leftarrow g^s$
	$z \leftarrow g^x h^y$
$\mathsf{KGen}_{\mathcal{C}}(\mathsf{pp})$	$\mathsf{pk}_{\mathcal{S}} \leftarrow (h, z)$
$\mathsf{pk}_{\mathcal{C}} \leftarrow \perp,\ \mathsf{sk}_{\mathcal{C}} \leftarrow k_{\mathcal{C}} \leftarrow\!\!\$\, \mathbb{Z}_q$	$\mathsf{sk}_{\mathcal{S}} \leftarrow (s, x, y, k_{\mathcal{S}})$
return $(\mathsf{pk}_{\mathcal{C}}, \mathsf{sk}_{\mathcal{C}})$	**return** $(\mathsf{pk}_{\mathcal{S}}, \mathsf{sk}_{\mathcal{S}})$

Figure 7: Setup Phase of PHOENIX

3.1 Formal Description

Let $\mathbb{G}$ be a (multiplicative) finite cyclic group of order $q = q(\lambda)$. Let $H_{i \in \{\mathcal{C},\mathcal{S}\}} : \{0,1\}^\lambda \times \{0,1\}^* \to \mathbb{G}$ be hash functions to be modeled as random oracles. Let Π be a standard non-interactive zero-knowledge proof of knowledge system for length-2 discrete logarithm representation (instantiated in Figure 11). We construct our cryptographic password hardening service, PHOENIX, as follows.

Setup Phase. Figure 7 shows the setup algorithm as well as the key generation algorithms. The setup algorithm samples a common reference string crs of the proof system Π and a random generator g of the group $\mathbb{G}$, and outputs them as the public parameter pp. The client secret key is a random integer $k_{\mathcal{C}}$. The server secret key consists of random integers $s, x, y, k_{\mathcal{S}}$, while the corresponding public key consists of $h = g^s$ and $z = g^x h^y$.

Enrollment Phase. Figure 8 shows the enrollment protocol. The input auxiliary information aux is either an empty string denoted by ϵ, or a tuple $(n_{\mathcal{S}}, n_{\mathcal{C}})$ of server and client nonces which is purely for proving forward security. In the former usual case, the server and the client sample their nonces $n_{\mathcal{S}}$ and $n_{\mathcal{C}}$ respectively independently and randomly. Next, the server sends the server PRF value $h_{\mathcal{S}} = H_{\mathcal{S}}(\mathsf{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}}$ and the server nonce $n_{\mathcal{S}}$ to the client, who computes the client PRF value $h_{\mathcal{C}} = H_{\mathcal{C}}(\mathsf{un}, n_{\mathcal{S}})^{k_{\mathcal{C}}}$ locally, and encrypts the value $h_{\mathcal{S}} \cdot h_{\mathcal{C}}$ using an ElGamal-like encryption scheme as $(g^r, h^r \cdot h_{\mathcal{S}} \cdot h_{\mathcal{C}}, z^r)$. The element z^r serves as an in-

tegrity tag which is important for proving the hiding property. The client then store the ciphertext and the nonces as the enrollment record.

Validation Phase. Figure 9 shows the validation protocol. The client wishes to validate whether T is a valid enrollment record of the given candidate username un and password pw. Recall that an enrollment record is of the form $T = (g^r, h^r \cdot h_{\mathcal{S}} \cdot h_{\mathcal{C}}, z^r, n_{\mathcal{S}}, n_{\mathcal{C}})$. To prepare for a validation request, the client divides the element $h^r \cdot h_{\mathcal{S}} \cdot h_{\mathcal{C}}$ by the candidate PRF value $H_{\mathcal{C}}(\mathsf{un}, n_{\mathcal{S}})^{k_{\mathcal{C}}}$, and rerandomizes the ciphertext components. It then sends the rerandomized ciphertext and the server nonce to the server. The latter checks if the ciphertext is indeed a valid encryption of $h_{\mathcal{S}} = H_{\mathcal{S}}(\mathsf{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}}$ and, if so, returns a proof of knowledge of this fact. If the proof passes verification, then the client is convinced that the candidate username and password satisfy $h_{\mathcal{S}} = H_{\mathcal{S}}(\mathsf{un}, n_{\mathcal{S}})^{k_{\mathcal{S}}}$ and $h_{\mathcal{C}} = H_{\mathcal{C}}(\mathsf{un}, n_{\mathcal{S}})^{k_{\mathcal{C}}}$, and concludes that the enrollment record T is valid.

Key Rotation Phase. Figure 10 shows the key rotation protocol and the update algorithm. In a nutshell, the protocol and the algorithm work together to perturb the secret keys and the enrollment records randomly yet consistently through homomorphisms. To be concrete, in the key rotation protocol, the server samples random integers $\alpha, \beta, \gamma, \delta$ and η, such that the secret key components of the client and the server are computed as $(s', x', y', k'_{\mathcal{S}}, k'_{\mathcal{C}}) = (\alpha s + \beta, \alpha x + \delta, y + \eta, \alpha s + \gamma, \alpha s)$. The client then updates each of the stored enrollment records T as follows. Recall that an enrollment record $T = (T_1, T_2, T_3, n_{\mathcal{S}}, n_{\mathcal{C}})$ is of the form $(T_1, T_2, T_3) = (g^r, g^{sr} g_{\mathcal{S}}^{k_{\mathcal{S}}} g_{\mathcal{C}}^{k_{\mathcal{C}}}, g^{(x+sy)r})$. Denote $r' := r + v$. For consistency, T_2 is updated as

$$\begin{aligned} T'_2 &= (T_2 \cdot h^v)^{\alpha} \cdot (T_1 \cdot g^v)^{\beta} \cdot g_{\mathcal{S}}^{\gamma} \\ &= g^{\alpha s(r+v)} g_{\mathcal{S}}^{\alpha k_{\mathcal{S}}} g_{\mathcal{C}}^{\alpha k_{\mathcal{C}}} \cdot g^{\beta(r+v)} \cdot g_{\mathcal{S}}^{\gamma} \\ &= g^{(\alpha s+\beta)(r+v)} g_{\mathcal{S}}^{\alpha k_{\mathcal{S}}+\gamma} g_{\mathcal{C}}^{\alpha k_{\mathcal{C}}} \\ &= g^{s'r'} g_{\mathcal{S}}^{k'_{\mathcal{S}}} g_{\mathcal{C}}^{k'_{\mathcal{C}}}. \end{aligned}$$

To update T_3, the client obtains from the server the value $\zeta = \delta + \alpha \cdot \eta \cdot s + \beta \cdot (y + \eta)$, and computes

$$\begin{aligned} T'_3 &= (T_3 \cdot z^v)^{\alpha} \cdot (T_1 \cdot g^v)^{\zeta} \\ &= g^{\alpha(x+sy)(r+v)} \cdot g^{(\delta+\alpha\cdot\eta\cdot s+\beta\cdot(y+\eta))(r+v)} \\ &= g^{((\alpha x+\delta)+(\alpha s+\beta)(y+\eta))(r+v)} \\ &= g^{(x'+s'y')r'}. \end{aligned}$$

The client runs the update algorithm on all of its stored enrollment records.

Correctness. The correctness of PHOENIX follows immediately from the completeness of Π.

3.2 Security Analysis

We give intuitions behind why PHOENIX is partially oblivious, hiding, binding, and forward secure in the random oracle model, assuming the DDH assumption holds in $\mathbb{G}$. We refer the curious readers to Appendix C for the formal security analysis.

Partial Obliviousness. Partial obliviousness means that a compromised server cannot distinguish which password among pw_0^* and pw_1^* was used to generate an enrollment record for some known username un^*. To show why this requirement is satisfied, recall that in the challenge enrollment record $T^* = (T_1^*, T_2^*, T_3^*, n_{\mathcal{S}}^*, n_{\mathcal{C}}^*)$, the only component which is dependent on the password pw_b^* is T_2^*, which is of the form $T_2^* = h^r h_{\mathcal{S}} H_{\mathcal{C}}(\mathsf{un}^*, \mathsf{pw}_b^*, n_{\mathcal{C}}^*)^{k_{\mathcal{C}}}$. Since $H_{\mathcal{C}}$ is modeled as a random oracle, both $H_{\mathcal{C}}(\mathsf{un}^*, \mathsf{pw}_0^*, n_{\mathcal{C}}^*)$ and $H_{\mathcal{C}}(\mathsf{un}^*, \mathsf{pw}_1^*, n_{\mathcal{C}}^*)$ can be programmed to random values independent of the passwords, which perfectly hide the bit b from the server. One subtlety here is the consistency of the simulation of the random oracle, which can be ensured as long as no oracles are queried on inputs containing the random nonce $n_{\mathcal{C}}^*$ before T^* is generated. Fortunately, the latter happens with overwhelming probability as $n_{\mathcal{C}}^*$ is randomly picked by the challenger during the generation of T^*.

Hiding. The hiding property, which defends against dictionary attacks by a compromised client, is the most difficult property to prove. Our proof is inspired by the techniques used to prove the security of password-authenticated key exchange (PAKE) protocols. The main idea is to gradually and unnoticeably replace the challenge enrollment record with a truly random one, such that it hides the password perfectly. During the course, we argue that the only ways for the adversary to notice the changes are either solving the DDH problem or guessing the correct password in a query to the validation oracle. Since DDH is assumed to be hard, we conclude that the adversary cannot perform better than guessing.

Binding. The binding property requires that a malicious cannot convince the client that an enrollment record T^* is valid for two distinct username-password tuples $(\mathsf{un}_0^*, \mathsf{pw}_0^*)$ and $(\mathsf{un}_1^*, \mathsf{pw}_1^*)$. This property follows straightforwardly from the DL assumption, which states that finding the discrete logarithm of g_2 base g_1 is hard for random

Enrollment Protocol

Client $\mathcal{C}(\mathsf{sk}_\mathcal{C}, \mathsf{un}, \mathsf{pw}, \mathsf{aux})$		**Server** $\mathcal{S}(\mathsf{sk}_\mathcal{S}, \mathsf{un}, \mathsf{aux})$
parse $\mathsf{pk}_\mathcal{S}$ **as** $(h = g^s, z = g^x h^y)$, $\mathsf{sk}_\mathcal{C}$ **as** $k_\mathcal{C}$		**parse** $\mathsf{sk}_\mathcal{S}$ **as** $(s, x, y, k_\mathcal{S})$
if $\mathsf{aux} \neq \epsilon$ **then parse** aux **as** $(n_\mathcal{S}, n_\mathcal{C})$		**if** $\mathsf{aux} \neq \epsilon$ **then parse** aux **as** $(n_\mathcal{S}, n_\mathcal{C})$
else $n_\mathcal{C} \leftarrow \{0,1\}^\lambda$		**else** $n_\mathcal{S} \leftarrow \{0,1\}^\lambda$
$r \leftarrow_\$ \mathbb{Z}_q,\ h_\mathcal{C} \leftarrow H_\mathcal{C}(\mathsf{un}, \mathsf{pw}, n_\mathcal{C})^{k_\mathcal{C}}$	$\xleftarrow{h_\mathcal{S}, n_\mathcal{S}}$	$h_\mathcal{S} \leftarrow H_\mathcal{S}(\mathsf{un}, n_\mathcal{S})^{k_\mathcal{S}}$
return $T \leftarrow (g^r, h^r \cdot h_\mathcal{S} \cdot h_\mathcal{C}, z^r, n_\mathcal{S}, n_\mathcal{C})$		

Figure 8: Enrollment Protocol of PHOENIX

Validation Protocol

Client $\mathcal{C}(\mathsf{sk}_\mathcal{C}, T, \mathsf{un}, \mathsf{pw})$		**Server** $\mathcal{S}(\mathsf{sk}_\mathcal{S}, \mathsf{un})$
parse $\mathsf{pk}_\mathcal{S}$ **as** (h, z), $\mathsf{sk}_\mathcal{C}$ **as** $k_\mathcal{C}$		**parse** $\mathsf{sk}_\mathcal{S}$ **as** $(s, x, y, k_\mathcal{S})$
parse T **as** $(T_1, T_2, T_3, n_\mathcal{S}, n_\mathcal{C})$		
$u \leftarrow_\$ \mathbb{Z}_q, b \leftarrow 0$		
$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \leftarrow \begin{bmatrix} T_1 \cdot g^u \\ T_2 \cdot h^u / H_\mathcal{C}(\mathsf{un}, \mathsf{pw}, n_\mathcal{C})^{k_\mathcal{C}} \\ T_3 \cdot z^u \end{bmatrix}$	$\xrightarrow{(c_1, c_2, c_3, n_\mathcal{S})}$	**if** $\begin{bmatrix} c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} c_1^s \cdot H_\mathcal{S}(\mathsf{un}, n_\mathcal{S})^{k_\mathcal{S}} \\ c_1^x (c_2 / H_\mathcal{S}(\mathsf{un}, n_\mathcal{S})^{k_\mathcal{S}})^y \end{bmatrix}$ **then**
$b \leftarrow \Pi.\mathsf{Vf}((g, h, c_1, c_2, H_\mathcal{S}(\mathsf{un}, n_\mathcal{S})), \pi)$	$\xleftarrow{\pi}$	$\pi \leftarrow_\$ \Pi.\mathsf{PoK}\{(s, k_\mathcal{S}) : c_2 = c_1^s \cdot H_\mathcal{S}(\mathsf{un}, n_\mathcal{S})^{k_\mathcal{S}} \ \wedge\ h = g^s\}$
return b		**endif**

Figure 9: Validation Protocol of PHOENIX

(g_1, g_2). To see why, note that if the enrollment record $T^* = (T_1^*, T_2^*, T_3^*, n_\mathcal{S}^*, n_\mathcal{C}^*)$ stores both tuples $(\mathsf{un}_b^*, \mathsf{pw}_b^*)$, $b \in \{0,1\}$, then T_2^* is of the form $T_2^* = h^r H_\mathcal{S}(\mathsf{un}_b^*, n_\mathcal{S}^*)^{k_\mathcal{S}} H_\mathcal{C}(\mathsf{un}_b^*, \mathsf{pw}_b^*, n_\mathcal{C}^*)^{k_\mathcal{C}}$. Then it must be the case that $H_\mathcal{S}(\mathsf{un}_0^*, n_\mathcal{S}^*)^{k_\mathcal{S}} H_\mathcal{C}(\mathsf{un}_0^*, \mathsf{pw}_0^*, n_\mathcal{C}^*)^{k_\mathcal{C}} = H_\mathcal{S}(\mathsf{un}_1^*, n_\mathcal{S}^*)^{k_\mathcal{S}} H_\mathcal{C}(\mathsf{un}_1^*, \mathsf{pw}_1^*, n_\mathcal{C}^*)^{k_\mathcal{C}}$. To exploit this collision, the challenger simulates $H_\mathcal{S}$ and $H_\mathcal{C}$ such that their inputs are mapped to g_1^a and g_2^b respectively for random exponents a and b. Doing so allows it to extract the discrete logarithm from the ratio of the exponents associated with $H_\mathcal{S}$ and $H_\mathcal{C}$ in the expression respectively.

Forward Security. PHOENIX achieves $\mathcal{L}$-forward security with a mild leakage defined by the leakage function $\mathcal{L}$ which, on input $T = (T_1, T_2, T_3, n_\mathcal{S}, n_\mathcal{C})$, merely outputs the nonces $(n_\mathcal{S}, n_\mathcal{C})$. This can be proved by an information-theoretic argument that, all possible combinations of client and server secret keys obtainable from fresh key generation can also be obtained by key rotation. Then, no matter how the new secret keys are generated, the enrollment record T can be updated to be consistent with the new keys, and is indistinguishable to a fresh enrollment record generated using the same nonces.

4 Evaluation

We implemented a prototype using Python3, Falcon as web framework, and Charm for the cryptographic computations. We used NIST P-256 for all TLS public key operations and as the base group for PHOENIX. Information was passed to PHOENIX via HTTP GET parameters and returned as a text/json response (with group elements encoded in base64): therefore, an enrollment interaction would go as follows. The client sends an http request to `/enroll?tweak=john` and would get back a response in the following form: {`hs="rPHu...LcQ==",ns="4qKM...uWQ="`}

We then measured the performance of PHOENIX in comparison with PYTHIA and the scheme by Schneider *et al.* [21] on Amazon EC2 using t2.micro instances with the server running in Frankfurt and clients both on a separate t2.micro instance in Frankfurt and Ireland. At the time of writing, t2.micro instances were equipped with 1 GB of RAM and one core Intel XEON E5-2676.

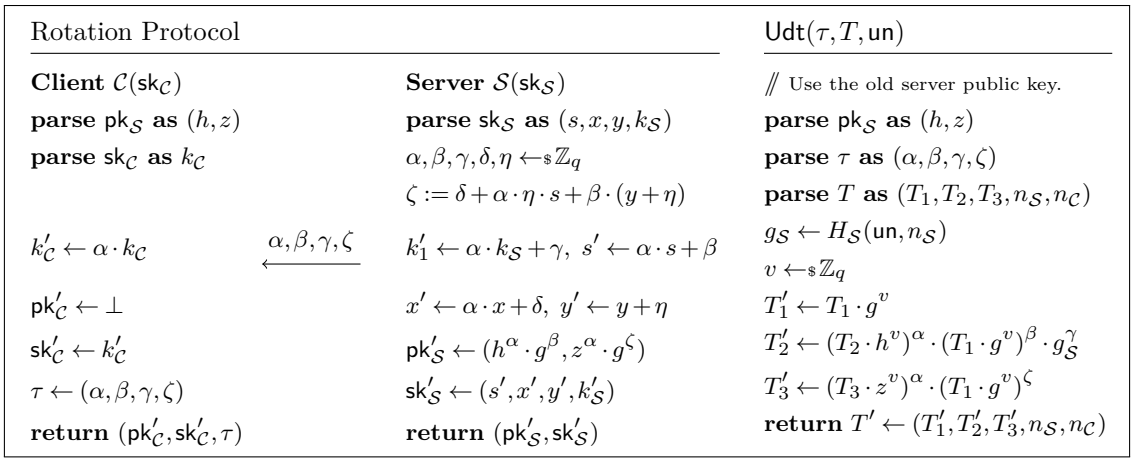

Figure 10: Rotation Protocol of PHOENIX

	HTTP	HTTPS	Frankfurt HTTPS keep-alive	HTTP	HTTPS	Ireland HTTPS keep-alive
RTT (64 bytes)		1.2			23	
PYTHIA eval	17.93	25.28	16.01	62.03	113.79	38.56
Schneider *et al.* enroll	9.80	22.86	8.14	53.72	111.40	30.89
Schneider *et al.* validate	12.30	25.65	10.73	56.32	115.32	33.49
PHOENIX enroll	5.43	17.93	3.89	50.30	107.25	26.52
PHOENIX validate	9.74	22.78	8.06	53.92	113.02	30.73

Table 1: Latency in millisecond (ms)

We used the Nginx web server configured with ECDHE-ECDSA-AES128-GCM-SHA256 for TLS and uWSGI for the Python applications.

Latency. For the latency measurements, a full interaction was executed between the cryptographic service and the consuming web service. The numbers take both server and client-side processing into account. As the client-side computations for PHOENIX are significant compared to the server-side computations, the total latency is significantly larger than the pure latency of the HTTP(S) requests. The latency measurements try to answer the question "How long does the user have to wait for the website to check the password?".

The presented numbers are an average over 5,000 executions of the respective protocol. We measured HTTP and HTTPS setups as well as HTTPS with keep-alive which removes all costs for TCP and TLS handshakes and is therefore close to the inherent latency of the cryptographic scheme.

As shown in Table 1, even in a single datacenter setup, the full TLS handshake takes approximately as much time as the computations of PHOENIX: Reusing a keep-alive connection it takes approximately half the time compared to a fresh HTTPS connection in the same datacenter setting. If the crypto service is hosted by a different datacenter from the web application, network round-trip time quickly dominates the overall execution time of PHOENIX: There is only one round-trip inherently needed for either PHOENIX protocol execution and the difference between the one-datacenter and same-continent setting is almost exactly this one round-trip using keep-alive. In a real world setup for a large website, we expect the web service to keep a connection to the cryptographic service open at any time and the HTTPS with keep-alive measurements is realistic.

Throughput. For throughput measurements, we used the Apache benchmark tool with 10,000 iterations and 400 parallel requests. uWSGI and Nginx were both configured to run with two processes to keep OS overhead on the single core server low.

As shown in Table 2, PHOENIX can process approximately 50 % more requests than the scheme by Schneider *et al.* and about three times as many as PYTHIA. It can even be easily scaled to multiple

	HTTPS keep-alive	HTTPS
static page	$> 10,000$	795.22
parameter	2,607.16	807.50
PYTHIA eval	128.50	125.75
Schneider *et al.* enroll	380.37	278.51
Schneider *et al.* validate	221.75	183.92
PHOENIX enroll	1,557.81	697.66
PHOENIX validate	371.34	275.42

Table 2: Requests per second

cores or even servers if needed.

Current suggestions for state of the art password hashing [23] suggest choosing a work factor of up to one second. Apple uses 10,000 iterations of PBKDF2 for iTunes [12], which takes around 278.80 ms on our Amazon instance. Both computation cost and latency of PHOENIX are considerably below this mark which suggests PHOENIX is highly practical and can even be combined with traditional password hardening in a hybrid approach.

5 Conclusion

We revisit the existing security notions of cryptographic password hardening service and found that some important properties were overlooked or not well defined. While PYTHIA [9] and the subsequent work by Schneider *et al.* [21] highlight the importance of key rotation, none of their security notions take this feature into account. Furthermore, we argue that the security definitions of Schneider *et al.* [21] are weak. We give a stronger definition and show that the scheme of Schneider *et al.* is insecure under our security definition. The attack is simple yet of high practical relevance since it allows an offline password dictionary attack, which is supposedly avoided by the password hardening service.

We propose the PHOENIX password hardening service which greatly improves efficiency while satisfies all desirable security properties. Specifically, it is more efficient than the insecure protocol of Schneider *et al.* and the seminal PYTHIA PRF service. With its efficiency and simplicity, PHOENIX is the first readily deployable password hardening service.

Acknowledgments

This research is based upon work supported by the German research foundation (DFG) through the collaborative research center 1223, by the German Federal Ministry of Education and Research (BMBF) through the project PROMISE, and by the state of Bavaria at the Nuremberg Campus of Technology (NCT). NCT is a research cooperation between the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and the Technische Hochschule Nürnberg Georg Simon Ohm (THN). Sherman Chow is supported by the Early Career Award and General Research Funds (CUHK 14201914) of the Research Grants Council of Hong Kong. We thank Andrei Sabelfeld and the reviewers for for valuable comments that helped to improve our paper.

References

[1] Ali Bagherzandi, Stanislaw Jarecki, Nitesh Saxena, and Yanbin Lu. Password-protected secret sharing. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 11*, pages 433–444, Chicago, Illinois, USA, October 17–21, 2011. ACM Press.

[2] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 356–374, San Francisco, CA, USA, March 19–21, 2008. Springer, Heidelberg, Germany.

[3] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.

[4] Tom Berson, Drew Dean, Matthew K. Franklin, Diana K. Smetters, and Mike Spreitzer. Cryptology as a network service. In *NDSS 2001*, San Diego, CA, USA, February 7–9, 2001. The Internet Society.

[5] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical Report TR260, Institute for theoretical computer science, ETH Zürich, 1997.

[6] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 13–25, Santa Barbara, CA, USA, August 23–27, 1998. Springer, Heidelberg, Germany.

[7] Mario Di Raimondo and Rosario Gennaro. Provably secure threshold password-authenticated key exchange. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 507–523, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany.

[8] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

[9] Adam Everspaugh, Rahul Chaterjee, Samuel Scott, Ari Juels, and Thomas Ristenpart. The pythia prf service. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 547–562, Washington, D.C., 2015. USENIX Association.

[10] Marc Fischlin and Dominique Schröder. Security of blind signatures under aborts. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 297–316, Irvine, CA, USA, March 18–20, 2009. Springer, Heidelberg, Germany.

[11] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 303–324, Cambridge, MA, USA, February 10–12, 2005. Springer, Heidelberg, Germany.

[12] Apple Inc. ios security. `https://www.apple.com/business/docs/iOS_Security_Guide.pdf`, 2016. [Online; accessed 4-June-2017].

[13] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 150–164, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Heidelberg, Germany.

[14] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 669–684, Berlin, Germany, November 4–8, 2013. ACM Press.

[15] Philip D. MacKenzie, Thomas Shrimpton, and Markus Jakobsson. Threshold password-authenticated key exchange. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 385–400, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany.

[16] Allec Muffet. Facebook: Password hashing and authentication. `https://video.adm.ntnu.no/pres/54b660049af94`, 2015. Video.

[17] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 327–346, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

[18] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press.

[19] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[20] New York Post. Verizon wants $1b discount on yahoo deal after reports of hacking, spying, 2016. [Online; accessed 4-June-2017].

[21] Jonas Schneider, Nils Fleischhacker, Dominique Schröder, and Michael Backes. Efficient cryptographic password hardening services from partially oblivious commitments. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1192–1203, Vienna, Austria, October 24–28, 2016. ACM Press.

[22] Dominique Schröder and Dominique Unruh. Security of blind signatures revisited. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 662–679, Darmstadt, Germany, May 21–23, 2012. Springer, Heidelberg, Germany.

[23] John Steven and Jim Manico. Owasp password storage cheat sheet. `https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet`, 2016.

[24] Wikipedia. List of data breaches — wikipedia, the free encyclopedia, 2016. [Online; accessed 4-June-2017].

A Related Work

Many primitives are related to partially-oblivious pseudorandom functions [9], such as delegatable PRFs [14] and fully oblivious PRFs [18, 11]. They do not allow partial obliviousness [9].

One-more unpredictability formalized for partially oblivious PRFs [9] draws some similarities to one-more unforgeability of blind signature schemes [13, 19, 10, 22]. This similarity inspires the subsequent analysis [21] that the "one-more" type assumptions are needed for proving the security of PYTHIA [9].

One should not confuse the resistance against offline dictionary attack with a similar property achieved by threshold password-authenticated key-exchange (t-PAKE) [15]. We only consider protocols between two parties, namely, a client and the server. On the other hand, to authenticate using t-PAKE, a client has to interact with a threshold number of available servers. There are other schemes [15, 1, 7] which support blinding, but they fail to achieve *partial* blinding (and hence rate-limiting).

One may also consider our primitive to be similar to other proof-of-knowledge protocols such as P-Signatures [2] since both share a mechanism to verify if two commitments are committing to the same value. However, they are different in general. In particular, ours does not involve any signature.

B Preliminaries

Non-Interactive Zero-Knowledge Proof of Knowledge (NIZKPoK). $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Vf})$ is an *adaptive non-interactive zero-knowledge (NIZK) proof system* for a language $L \in \mathsf{NP}$ with the witness relation $\mathcal{R}$ if it satisfies the following properties:

Completeness: For all x, w such that $\mathcal{R}(x,w) = 1$, and common reference strings $\mathsf{crs} \in \mathsf{Gen}(1^\lambda)$, we have $\mathsf{Vf}(\mathsf{crs}, x, \mathsf{Prove}(\mathsf{crs}, x, w)) = 1$.

Soundness: For all adversaries $\mathcal{A}$,

$$\Pr[x \notin L \ \wedge\ \mathsf{Vf}(\mathsf{crs}, x, \pi) \to 1 : \mathsf{crs} \leftarrow_\$ \mathsf{Gen}(1^\lambda); (x,\pi) \leftarrow_\$ \mathcal{A}(\mathsf{crs})] = \epsilon(\lambda).$$

Zero-Knowledge: There exists PPT simulator $\mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Prove}})$ such that, for all PPT adversaries $\mathcal{A}$,

$$|\Pr[\mathcal{A}^{\mathsf{Prove}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) \to 1 : \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)] - \Pr[\mathcal{A}^{\mathcal{S}'(\mathsf{crs},\mathsf{td},\cdot,\cdot)}(\mathsf{crs}) \to 1 : (\mathsf{crs},\mathsf{td}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^\lambda)]| = \epsilon(\lambda)$$

where $\mathcal{S}'(\mathsf{crs}, \mathsf{td}, x, w) = \mathcal{S}^{\mathsf{Prove}}(\mathsf{crs}, \mathsf{td}, x)$.

Furthermore, Π is a *proof of knowledge (PoK) system* if, for all PPT provers P^*, there exists a PPT algorithm E_{P^*} such that

$$|\Pr[\mathsf{Vf}(\mathsf{crs}, x, \pi) = 1 \ \wedge\ (x,w) \notin \mathcal{R} : \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda); (x,\pi) \leftarrow P^*(\mathsf{crs}), w \leftarrow E_{P^*}(\mathsf{crs}, x, \pi)]| = \epsilon(\lambda)$$

For ease of reading, we denote by $\mathsf{PoK}\{w : \mathcal{R}(x,w) = 1\}$ the execution of $\mathsf{Prove}(\mathsf{crs}, x, w)$.

Discrete Logarithm (DL) Assumption. Let $\mathbb{G}$ be a finite cyclic group of order $q = q(\lambda)$. Let g be a generator of $\mathbb{G}$, and h be a group element. The discrete logarithm problem asks to find an integer $x \in \mathbb{Z}_q$ such that $h = g^x$. The discrete logarithm assumption states that, for any PPT algorithm $\mathcal{A}$, the probability of $\mathcal{A}$ solving a random instance of the discrete logarithm problem is negligible.

Decisional Diffie-Hellman (DDH) Assumption. Let $\mathbb{G}$ be a finite cyclic group of order $q = q(\lambda)$. Let g be a generator of $\mathbb{G}$, and $a, b, c \in \mathbb{Z}_q$. The decisional Diffie-Hellman problem asks to distinguish the tuple $(g, g^a, g^b, g^{a\cdot b})$ from (g, g^a, g^b, g^c). The decisional Diffie-Hellman assumption states that, for any PPT algorithm $\mathcal{A}$, the probability of $\mathcal{A}$ solving a random instance of the decisional Diffie-Hellman problem is negligible.

C Formal Security Analysis

We will show that PHOENIX is partially oblivious, hiding, binding, and forward secure, relying mainly on the DDH assumption.

Note that the instantiation of Π in Figure 11 is a well-known extension of the Schnorr proofs [5], which is complete, sound, and zero-knowledge, assuming the DL assumption holds in $\mathbb{G}$ (implied by the DDH assumption) and the two hash functions are modeled as random oracles. Thus, in the following, we will assume Π is sound and zero-knowledge.

For conciseness, consider an extended DDH problem, which asks to distinguish whether $c_i \leftarrow_\$ \mathbb{Z}_q$ for $i \in [t]$ or $c_i = a \cdot b_i$ for $i \in [t]$, when given a tuple $(g, g^a, g^{b_i}, g^{c_i})_{i=1}^t$ for some $t = \mathsf{poly}(\lambda)$. By standard hybrid argument, it can be shown that if DDH is hard in $\mathbb{G}$, then so does the extended DDH.

Theorem C.1 (Partial Oblivious) *Suppose the DDH assumption holds in $\mathbb{G}$, and $H_{\mathcal{C}}$ is modeled as a random oracle, then* PHOENIX *is partially oblivious.*

Proof: The idea of the proof is to replace the pseudorandom values $H_{\mathcal{C}}(\mathsf{un}^*, \mathsf{pw}_b^*, n_{\mathcal{C}}^*)^{kc}$ for $b \in \{0,1\}$ by truly random values. Then, we can reverse the role of pw_0 and pw_1. Formally, we prove by defining a

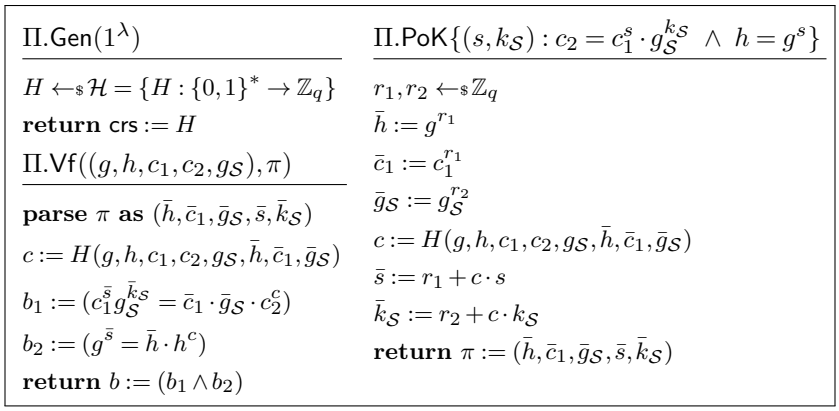

Figure 11: Instantiation of Π

sequence of hybrid experiments for $b \in \{0,1\}$, each differs slightly from the previous:

$\mathsf{EXP}_{b,0}$: is identical to $\mathsf{Obl}^b_{\Pi,\mathcal{A}}$.

$\mathsf{EXP}_{b,1}$: The challenger simulates the random oracle $H_\mathcal{C}$ as follows. On query $H_\mathcal{C}(\mathsf{un}, \mathsf{pw}, n_\mathcal{C})$, it samples $a \leftarrow_\$ \mathbb{Z}_q$ and returns g^a. This experiment is functionally equivalent to $\mathsf{Exp}_{b,0}$.

$\mathsf{EXP}_{b,2}$: When executing $\langle \mathcal{C}(\mathsf{sk}_\mathcal{C}, \mathsf{un}^*, \mathsf{pw}^*_b, \epsilon), \mathcal{A}_2(\mathsf{st})\rangle_{\mathtt{enrl}}$, since $\mathsf{aux} = \epsilon$, the challenger picks the client nonce $n^*_\mathcal{C}$ randomly and programs the random oracle $H_\mathcal{C}$ on $(\mathsf{un}^*, \mathsf{pw}^*_b, n^*_\mathcal{C})$ and $(\mathsf{un}^*, \mathsf{pw}^*_{1-b}, n^*_\mathcal{C})$. If any oracle (including the random oracle) is queried on input containing $n^*_\mathcal{C}$ before, the challenger aborts. This happens with probability $O(2^{-\lambda})$ for each oracle query. Thus, this experiment is computationally indistinguishable to $\mathsf{Exp}_{b,1}$.

$\mathsf{EXP}_{b,3}$: The challenger is given an extended DH-tuple $(g, g^{k_\mathcal{C}}, g^\gamma, g^\theta, g^\delta, g^\xi)$ with $\delta = k_\mathcal{C}\gamma$ and $\xi = k_\mathcal{C}\theta$. Since the challenger does not know $k_\mathcal{C}$, it computes the pseudorandom values $H_\mathcal{C}(\mathsf{un}, \mathsf{pw}, n_\mathcal{C})^{k_\mathcal{C}}$ differently. Let a be such that $H_\mathcal{C}(\mathsf{un}, \mathsf{pw}, n_\mathcal{C})$ is programmed to g^a. The challenger computes $H_\mathcal{C}(\mathsf{un}, \mathsf{pw}, n_\mathcal{C})^{k_\mathcal{C}}$ as $(g^{k_\mathcal{C}})^a$. Since no oracle is queried on input containing $n^*_\mathcal{C}$ before the challenge is requested, $H_\mathcal{C}(\mathsf{un}^*, \mathsf{pw}^*_b, n^*_\mathcal{C})$ is not yet programmed. Upon receiving the challenge request $(\mathsf{un}^*, \mathsf{pw}^*_0, \mathsf{pw}^*_1)$ from $\mathcal{A}$, it programs $H_\mathcal{C}(\mathsf{un}^*, \mathsf{pw}^*_b, n^*_\mathcal{C}) := g^\gamma$ and $H_\mathcal{C}(\mathsf{un}^*, \mathsf{pw}^*_b, n^*_\mathcal{C})^{k_\mathcal{C}} := g^\delta$. Additionally, it programs $H_\mathcal{C}(\mathsf{un}^*, \mathsf{pw}^*_{1-b}, n^*_\mathcal{C}) := g^\theta$ and $H_\mathcal{C}(\mathsf{un}^*, \mathsf{pw}^*_{1-b}, n^*_\mathcal{C})^{k_\mathcal{C}} := g^\xi$. This experiment is functionally equivalent to $\mathsf{EXP}_{b,2}$.

$\mathsf{EXP}_{b,4}$: The challenger is given a random tuple $(g, g^{k_\mathcal{C}}, g^\gamma, g^\theta, g^\delta, g^\xi)$ with $\delta, \xi \leftarrow_\$ \mathbb{Z}_q$. It simulates $H_\mathcal{C}$ as in $\mathsf{EXP}_{b,3}$. This experiment is computationally indistinguishable from $\mathsf{EXP}_{b,3}$, by the (extended) DDH assumption.

Observe that $\mathsf{EXP}_{0,4}$ and $\mathsf{EXP}_{1,4}$ are functionally equivalent. Thus, we have $\mathsf{Obl}^0_{\Pi,\mathcal{A}}$ being computationally indistinguishable from $\mathsf{Obl}^1_{\Pi,\mathcal{A}}$. □

Theorem C.2 (Hiding) *Let $q > 2^\lambda$. Suppose that the DDH assumption holds in $\mathbb{G}$, and $H_\mathcal{S}$ is modeled as a random oracle, then* PHOENIX *is hiding.*

Proof: The idea of the proof is to gradually switch the challenge enrollment record to an entirely random one using hybrid argument. After arriving at that hybrid experiment, the information that can be obtained by the adversary from the oracles can also be obtained by guessing the password. Thus, no adversary can perform better than the one which performs an online dictionary attack. We prove formally by defining a sequence of hybrid experiments, each differs slightly from the previous:

EXP_0: is identical to $\mathsf{Hiding}_{\Pi,\mathcal{A}}$.

EXP_1: The proofs are now simulated using the simulator $\mathcal{S}$ of the proof system Π. This experiment is computationally indistinguishable from EXP_0 by the computational zero-knowledge property of Π.

EXP_2: The challenger simulates the random oracle $H_\mathcal{S}$ as follows. When $\mathcal{A}$ queries $H_\mathcal{S}(\mathsf{un}, n_\mathcal{S})$, it samples $\gamma \leftarrow_\$ \mathbb{Z}_q$ and programs $H_\mathcal{S}(\mathsf{un}, n_\mathcal{S}) := g^\gamma$. It further computes $H_\mathcal{S}(\mathsf{un}, n_\mathcal{S})^{k_\mathcal{S}} = (g^{k_\mathcal{S}})^\gamma$. Notice that the knowledge of $k_\mathcal{S}$ is no longer required by the challenger. Furthermore, when executing $\langle \mathcal{C}(\mathsf{sk}_\mathcal{C}, \mathsf{un}^*, \mathsf{pw}^*, \epsilon), \mathcal{S}(\mathsf{sk}_\mathcal{S}, \mathsf{un}^*, \epsilon)\rangle_{\mathtt{enrl}}$, since $\mathsf{aux} = \epsilon$, the challenger picks fresh client and server nonces $n^*_\mathcal{C}$ and $n^*_\mathcal{S}$ respectively randomly and programs the random oracle $H_\mathcal{S}$ on $(\mathsf{un}^*, n^*_\mathcal{S})$. If any oracle (including the random oracle) is queried on input containing $n^*_\mathcal{S}$ before, the challenger aborts. This happens with probability $O(2^{-\lambda})$ for each oracle query. Thus, this experiment is computationally indistinguishable to Exp_1.

EXP_3: The challenger is given a DH-tuple $(g, g^\delta, g^{k_\mathcal{S}}, g^\eta)$ with $\eta = \delta \cdot k_\mathcal{S}$. It sets $\mathsf{pp} := g$ and computes $\mathsf{pk}_\mathcal{S}$ honestly. Eventually, $\mathcal{A}$ requests to receive an enrollment record for un^*. The challenger programs $H_\mathcal{S}(\mathsf{un}^*, n^*_\mathcal{S}) := g^\delta$ and replaces $H_\mathcal{S}(\mathsf{un}^*, n^*_\mathcal{S})^{k_\mathcal{S}}$ by g^η. This experiment differs from EXP_2 only if $\mathcal{A}$ queries an oracle for inputs containing $n^*_\mathcal{S}$ before requesting the challenge, which by our assumption will never happen.

EXP_4: The challenger is given a random tuple $(g, g^\delta, g^{k_\mathcal{S}}, g^\eta)$ with $\eta \leftarrow_\$ \mathbb{Z}_q$. It simulates $H_\mathcal{S}$ as in EXP_3. This experiment is computationally indistinguishable from EXP_3 by the DDH assumption.

EXP_5: In the validation oracle the server ignores the first condition $c_2 = c_1^s \cdot H_\mathcal{S}(\mathsf{un}, n_\mathcal{S})^{k_\mathcal{S}}$, and only checks the second condition $c_3 = c_1^x (c_2 / H_\mathcal{S}(\mathsf{un}, n_\mathcal{S})^{k_\mathcal{S}})^y$. At this point, note that s is not used anywhere. We show by a Cramer-Shoup-like argument [6] that this experiment is statistically indistinguishable from EXP_3.

Note that the only information about x and y available to an unbounded distinguisher is the relations which are linearly dependent to $\log_g z = x + sy$. Therefore, in the view of the distinguisher, the values of x and y are not uniquely determined, and can only be guessed correctly with a probability of at most $1/q$. Suppose that the two experiments can be distinguished with a probability higher than $1/q$. In such an event, $\mathcal{A}$ must have sent $(c_1, c_2, c_3, n_\mathcal{S})$ as the first message in an interaction with the validation oracle on username un, such that the tuple satisfies only the second condition but not the first one. Let $h_\mathcal{S} := H_\mathcal{S}(\mathsf{un}, n_\mathcal{S})^{k_\mathcal{S}}$. Since the first condition is not satisfied, it holds that $c_2 \neq c_1^s \cdot h_\mathcal{S}$. Let $s' \neq s$ be such that $c_2 = c_1^{s'} \cdot h_\mathcal{S}$. Then, by the second condition, we have $c_3 = c_1^x (c_2/h_\mathcal{S})^y = c_1^{x+s'y}$. In other words, $\log_{c_1} c_3 = x + s'y$. Since $s' \neq s$, the relations $\log_g z = x + sy$ and $\log_{c_1} c_3 = x + s'y$ are linearly independent, meaning that the distinguisher is able to figure out the values of x and y. However, this contradicts to the fact that this cannot happen with probability higher than $1/q$.

EXP_6: The challenger is given a DH-tuple (g, g^r, g^s, g^u) with $u = rs$. It sets $\mathsf{pk}_\mathcal{S} := (h, z)$ where $h := g^s$ and $z := g^x (g^s)^y$. The challenge enrollment record is replaced by $(T_1^*, T_2^*, T_3^*) = (c_1^*, c_2^* \cdot H_\mathcal{C}(\mathsf{un}, \mathsf{pw}, n_\mathcal{C})^{k_\mathcal{C}}, c_3^*)$, where $(c_1^*, c_2^*, c_3^*) := (g^r, g^{u+\eta}, g^{rx+uy})$ (since $H_\mathcal{S}(\mathsf{un}^*, n^*_\mathcal{S})^{k_\mathcal{S}}$ has been programmed to g^η). Note that c_3^* can be precomputed before answering any oracle queries. This experiment is functionally equivalent to EXP_5.

EXP_7: The challenger is given a random tuple (g, g^r, g^s, g^u) with $u \leftarrow_\$ \mathbb{Z}_q$. It simulates the challenge enrollment record as in EXP_6. This experiment is computationally indistinguishable from EXP_6 by the DDH assumption.

EXP_8: The challenger samples $r, s \leftarrow_\$ \mathbb{Z}_q$ (so that it knows s again) and $u \leftarrow_\$ \mathbb{Z}_q \setminus \{rs\}$ instead. This experiment is different from EXP_7 with probability only $1/q$, which is negligible.

EXP_9: This is the most technical transition. In this and the next (which is also the last) experiment, we assume an unbounded challenger and only use information-theoretic arguments. Suppose $\mathcal{A}$ queries the validation oracle on un and sends $(c_1, c_2, c_3, n^*_\mathcal{S})$ to the server. We split into two cases. First, $(c_1, c_2, c_3) = (c_1^* \cdot g^v, c_2^* \cdot h^v, c_3^* \cdot z^v)$ for some $v \in \mathbb{Z}_q$ (which is checkable by an unbounded challenger). In this case, we call the adversary successful, and the challenger outputs a simulated proof without checking the second condition (it must be satisfied). Otherwise, $(c_1, c_2, c_3) \neq (c_1^* \cdot g^v, c_2^* \cdot h^v, c_3^* \cdot z^v)$ for all $v \in \mathbb{Z}_q$. In this case, the challenger outputs $\perp$ without checking the second condition. We claim that this experiment is indistinguishable from EXP_8 by the following information-theoretic argument.

Throughout the experiment, $\mathcal{A}$ learns the relations $z = g^{x+sy}$, $c_2^* = g^{u+\eta}$, and $c_3^* = g^{rx+uy}$. If $\mathcal{A}$ is powerful, it might know $\log z = x + sy$, $\log c_2^* = u + \eta$, and $\log c_3^* = rx + uy$. Substituting the first and second into the third, we have $\log c_3^* - r \log z = (\log c_2^* - rs)y - \eta y$. Note that this is a quadratic equation with two variables (y, η) which has exponentially many solutions. η also counts as a variable since $H_\mathcal{S}(\mathsf{un}, n^*_\mathcal{S})^{k_\mathcal{S}}$ are never revealed to $\mathcal{A}$ for all un, and in particular for $\mathsf{un} = \mathsf{un}^*$. Suppose that $\mathcal{A}$ queries the validation oracle on $(\mathsf{un}, n_\mathcal{C}, n_\mathcal{S})$ where $(\mathsf{un}, n_\mathcal{S}) = (\mathsf{un}^*, n^*_\mathcal{S})$, and (c_1, c_2, c_3) received from the $\mathcal{A}$ satisfies the second condition. We have $c_3 = c_1^x c_2^y g^{-\eta \cdot y}$. By substituting $\log z = x + sy$, we have $\log c_3 - \log c_1 \log z = (\log c_2 - \log c_1 \cdot s)y - \eta y$. For this relation to be satisfied, $\mathcal{A}$ must either guess the tuple (y, η) correctly, which happens with negligible probability, or keep the coefficients unchanged. For the latter case, we have $\log c_2^* - rs = \log c_2 - \log c_1 \cdot s$ and $\log c_3^* - r \log z = \log c_3 - \log c_1 \log z$. Let $\log c_1 = r + v$ for some $v \in \mathbb{Z}_q$, or equivalently, $c_1 = c_1^* \cdot g^v$. We obtain $c_2 = c_2^* \cdot (g^s)^v = c_2^* \cdot h^v$ and $c_3 = c_3^* \cdot z^v$.

EXP_{10}: The challenger replaces T_1^*, T_2^* and T_3^* by random group elements in $\mathbb{G}$, and hence the challenge enrollment record is independent of pw^*. Internally, it stores $(c_1^*, c_2^*, c_3^*) := (T_1^*, T_2^* / H_\mathcal{C}(\mathsf{un}, \mathsf{pw}, n_\mathcal{C})^{k_\mathcal{C}}, T_3^*)$ so as to answer queries to the validation oracle. This experiment is functionally equivalent to EXP_9.

In EXP_{10}, the view of $\mathcal{A}$ is independent of pw^*

unless it is successful in one of the Q queries to the validation oracle on un^*. Among all successful adversaries, $\mathcal{A}$ cannot do better than guessing pw^* and hence (c_1^*, c_2^*, c_3^*) correctly, and re-randomizes the latter. Note that the probability of guessing the correct pw^* is upper-bounded by $\sum_{i=1}^{Q+1} p_i$. To conclude, the probability that $\mathcal{A}$ wins in EXP_0, that is $\mathsf{Hiding}_{\Pi,\mathcal{A}}$, is upper-bounded by $\sum_{i=1}^{Q+1} p_i + \epsilon(\lambda)$. □

Theorem C.3 (Binding) *Suppose that the DL assumption holds in $\mathbb{G}$, and $H_{\mathcal{S}}$ and $H_{\mathcal{C}}$ are modeled as random oracles, then* PHOENIX *is binding.*

Proof: Our idea is to program the random oracles so that they map inputs to random group elements, with their discrete logarithms known to the simulator. Thus, if the adversary outputs two valid tuples for the same enrollment record, the simulator can solve a system of linear equations of the exponents. It is then able to recover the discrete logarithm of a group element which is used as a generator for simulating the random oracles. Formally, we prove by reduction.

Suppose a PPT adversary $\mathcal{A}$ breaks binding with non-negligible probability, we construct a PPT solver $\mathcal{B}$ of the discrete logarithm problem. Let $\mathcal{B}$ be a simulator which receives a discrete logarithm problem instance (g_1, g_2). It generates crs honestly and sends $\mathsf{pp} := (\mathsf{crs}, g_1)$ to $\mathcal{A}$. $\mathcal{B}$ maintains dictionaries D_1 and D_2 mapping $(\mathsf{un}, n_{\mathcal{S}})$ and $(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}})$ respectively to random exponents. When $\mathcal{A}$ queries the random oracle $H_{\mathcal{S}}$ on $(\mathsf{un}, n_{\mathcal{S}})$, it checks whether $H_{\mathcal{S}}(\mathsf{un}, n_{\mathcal{S}})$ is programmed. If so, it retrieves and returns $H_{\mathcal{S}}(\mathsf{un}, n_{\mathcal{S}})$. Otherwise, it samples a random exponent $a \leftarrow \mathbb{Z}_q$, records $D_1[\mathsf{un}, n_{\mathcal{S}}] := a$, and programs $H_{\mathcal{S}}(\mathsf{un}, n_{\mathcal{S}}) := g_1^a$. $\mathcal{B}$ simulates $H_{\mathcal{C}}$ similarly. When $\mathcal{A}$ queries the random oracle $H_{\mathcal{C}}$ on $(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}})$, it checks whether $H_{\mathcal{C}}(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}})$ is programmed. If so, it retrieves and returns $H_{\mathcal{C}}(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}})$. Otherwise, it samples a random exponent $b \leftarrow \mathbb{Z}_q$, records $D_2[\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}}] := b$, and programs $H_{\mathcal{C}}(\mathsf{un}, \mathsf{pw}, n_{\mathcal{C}}) := g_2^b$.

Assuming $\mathcal{A}$ is successful, it outputs $(\mathsf{sk}_{\mathcal{C}}, T^*, \mathsf{un}_0^*, \mathsf{pw}_0^*, \mathsf{state})$ such that $\langle \mathcal{C}(\mathsf{sk}_{\mathcal{C}}, T^*, \mathsf{un}_0^*, \mathsf{pw}_0^*), \mathcal{A}_2(\mathsf{state}) \rangle_{\mathtt{val}}$ outputs 1 at the client side. Parse $\mathsf{sk}_{\mathcal{C}} = k_{\mathcal{C}}$, and $T^* = (T_1^*, T_2^*, T_3^*, n_{\mathcal{S}}^*, n_{\mathcal{C}}^*)$. Let a_0 and b_0 be such that $H_{\mathcal{S}}(\mathsf{un}_0^*, n_{\mathcal{S}}^*) = g_1^{a_0}$ and $H_{\mathcal{C}}(\mathsf{un}_0^*, \mathsf{pw}_0^*, n_{\mathcal{C}}^*) = g_2^{b_0}$. This means that $\mathcal{A}$ is able to produce a proof π_0 of the knowledge of $(s, k_{\mathcal{S},0})$ such that $T_2^* = (T_1^*)^s \cdot H_{\mathcal{S}}(\mathsf{un}_0^*, n_{\mathcal{S}}^*)^{k_{\mathcal{S},0}} \cdot H_{\mathcal{C}}(\mathsf{un}_0^*, \mathsf{pw}_0^*, n_{\mathcal{C}}^*)^{k_{\mathcal{C}}} = (T_1^*)^s \cdot g_1^{a_0 \cdot k_{\mathcal{S},0}} \cdot g_2^{b_0 \cdot k_{\mathcal{C}}}$ and $h = g_1^s$. Using the extractor of Π, $\mathcal{B}$ extracts $(s, k_{\mathcal{S},0})$.

Next, $\mathcal{A}$ outputs $(\mathsf{un}_1^*, \mathsf{pw}_1^*, \mathsf{state})$ such that $\langle \mathcal{C}(\mathsf{sk}_{\mathcal{C}}, T^*, \mathsf{un}_1^*, \mathsf{pw}_1^*), \mathcal{A}(\mathsf{state}) \rangle_{\mathtt{val}}$ outputs 1 at the client side. Let a_1 and b_1 be such that $H_{\mathcal{S}}(\mathsf{un}_1^*, n_{\mathcal{S}}^*) = g^{a_1}$ and $H_{\mathcal{C}}(\mathsf{un}_1^*, \mathsf{pw}_1^*, n_{\mathcal{C}}^*) = h^{b_1}$. This means that $\mathcal{A}$ is able to produce a proof π_1 of the knowledge of $(s, k_{\mathcal{S},1})$ such that $T_2^* = (T_1^*)^s \cdot H_{\mathcal{S}}(\mathsf{un}_1^*, n_{\mathcal{S}}^*)^{k_{\mathcal{S},1}} \cdot H_{\mathcal{C}}(\mathsf{un}_1^*, \mathsf{pw}_1^*, n_{\mathcal{C}}^*)^{k_{\mathcal{C}}} = (T_1^*)^s \cdot g_1^{a_1 \cdot k_{\mathcal{S},1}} \cdot h^{b_1 \cdot k_{\mathcal{C}}}$ and $h = g_1^s$. Using the extractor of Π, $\mathcal{B}$ extracts $(s, k_{\mathcal{S},1})$.

Through simple arithmetic, we obtain the relation $g_1^{a_0 \cdot k_{\mathcal{S},0}} \cdot g_2^{b_0 \cdot k_{\mathcal{C}}} = g_1^{a_1 \cdot k_{\mathcal{S},1}} \cdot g_2^{b_1 \cdot k_{\mathcal{C}}}$. That is, $\log_{g_1} g_2 = (a_1 \cdot k_{\mathcal{S},1} - a_0 \cdot k_{\mathcal{S},0}) / (k_{\mathcal{C}} \cdot (b_0 - b_1))$. Since $(\mathsf{un}_0^*, \mathsf{pw}_0^*) \neq (\mathsf{un}_1^*, \mathsf{pw}_1^*)$, b_0 and b_1 are sampled independently at random. Thus the above expression is well defined with overwhelming probability. $\mathcal{B}$ thus outputs $\frac{a_1 \cdot k_{\mathcal{S},1} - a_0 \cdot k_{\mathcal{S},0}}{k_{\mathcal{C}} \cdot (b_0 - b_1)}$ and solves the discrete logarithm problem with overwhelming probability. □

Theorem C.4 (Forward Security) *Let $\mathcal{L}$ be a leakage function such that $\mathcal{L}(T) := (n_{\mathcal{S}}, n_{\mathcal{C}})$ for $T = (T_1, T_2, T_3, n_{\mathcal{S}}, n_{\mathcal{C}})$.* PHOENIX *is $\mathcal{L}$-forward secure.*

Proof: We prove by showing that each pair of client and server secret keys output by the key generation algorithms can also be obtained via rotation from any old pair of secret keys, and vice versa.

Consider the $\mathsf{Rot}^b_{\Pi,\mathcal{A},\mathcal{L}}$ experiment. Let $(s, x, y, k_{\mathcal{S}}, k_{\mathcal{C}}) \in \mathbb{Z}_q^5$ be the client and server secret key components chosen by $\mathcal{A}$. There is a one-to-one correspondence between each $(s', x', y', k'_{\mathcal{S}}, k'_{\mathcal{C}}) \in \mathbb{Z}_q^5$ and each $(\alpha, \beta, \gamma, \delta, \eta) \in \mathbb{Z}_q^5$, given equivalently by

$$\begin{cases} s' &= \alpha \cdot s + \beta \\ x' &= \alpha \cdot x + \delta \\ y' &= y + \eta \\ k'_{\mathcal{S}} &= \alpha \cdot k_{\mathcal{S}} + \gamma \\ k'_{\mathcal{C}} &= \alpha \cdot k_{\mathcal{C}} \end{cases} \text{ and } \begin{cases} \alpha &= k'_{\mathcal{C}} / k_{\mathcal{C}} \\ \beta &= s' - \alpha \cdot s \\ \gamma &= k'_{\mathcal{S}} - \alpha \cdot k_{\mathcal{S}} \\ \delta &= x' - \alpha \cdot x \\ \eta &= y' - y \end{cases}.$$

Thus, the distribution of $(s', x', y', k'_{\mathcal{S}}, k'_{\mathcal{C}})$ which is sampled uniformly from $\mathbb{Z}_q^5$ and that which is computed from a uniformly random tuple $(\alpha, \beta, \gamma, \delta, \eta)$ are identical.

Next, let $T = (T_1, T_2, T_3)$ given by $\mathcal{A}$ be of the form $(g^r, g^{sr} g_{\mathcal{S}}^{k_{\mathcal{S}}} g_{\mathcal{C}}^{k_{\mathcal{C}}}, g^{(x+sy)r})$. The new record T' is of the form $T' = (g^{r'}, g^{s'r'} g_{\mathcal{S}}^{k'_{\mathcal{S}}} g_{\mathcal{C}}^{k'_{\mathcal{C}}}, g^{(x'+s'y')r'})$, where if $b = 0$ then $r' = r + v$ for a uniformly random $v \leftarrow_{\$} \mathbb{Z}_q$, and if $b = 1$ then $r' \leftarrow_{\$} \mathbb{Z}_q$ is sampled uniformly at random. The two cases give identical distributions. □