Simple Password-Hardened Encryption Services

Russell W. F. Lai[1], Christoph Egger[1], Manuel Reinert[2],
Sherman S. M. Chow[3], Matteo Maffei[4], and Dominique Schröder[1]

[1]Friedrich-Alexander University Erlangen-Nuremberg
[2]Saarland University
[3]Chinese University of Hong Kong
[4]Vienna University of Technology

Abstract

Passwords and access control remain the popular choice for protecting sensitive data stored online, despite their well-known vulnerability to brute-force attacks. A natural solution is to use encryption. Although standard practices of using encryption somewhat alleviate the problem, decryption is often needed for utility, and keeping the decryption key within reach is obviously dangerous.

To address this seemingly unavoidable problem in data security, we propose password-hardened encryption (PHE). With the help of an external crypto server, a service provider can recover the user data encrypted by PHE only when an end user supplied a correct password. PHE inherits the security features of password-hardening (Usenix Security '15), adding protection for the user data. In particular, the crypto server does not learn any information about any user data. More importantly, both the crypto server and the service provider can rotate their secret keys, a proactive security mechanism mandated by the Payment Card Industry Data Security Standard (PCI DSS).

We build an extremely simple password-hardened encryption scheme. Compared with the state-of-the-art password-hardening scheme (Usenix Security '17), our scheme only uses minimal number-theoretic operations and is, therefore, 30% - 50% more efficient. In fact, our extensive experimental evaluation demonstrates that our scheme can handle more than 525 encryption and (successful) decryption requests per second per core, which shows that it is lightweight and readily deployable with large-scale systems. Regarding security, our scheme also achieves a stronger soundness property, which puts less trust on the good behavior of the crypto server.

1 Introduction

Online services store huge amounts of sensitive user data in their databases, such as email and physical addresses, personal interests, *etc.* Pragmatically, accesses to this data is restricted to authorized users by an access control mechanism instead of by encryption and decryption, for a very simple reason that (the users of) the online services eventually need to use them. Nevertheless, some information is required to be stored in an encrypted form, such as credit card information, as mandated by the payment card industry data security standard (PCI DSS) [1]. Note that any form of encryption is useless if an attacker gains access to anything which possesses the decryption capabilities or leads to the decryption. For example, an attacker who gets access to a password database can first launch an offline dictionary attack to obtain user passwords, then logs in as these users and "legitimately" requests the online service provider to perform decryption. Even worse, an insider or a persistent attacker who obtains the decryption key can download the entire database and perform decryption offline. It is clear that as long as an online service provider has the full capability of decrypting the database, an attacker fully compromising it is just as powerful and can launch catastrophic attacks.

1.1 Password-Hardening Services

To defend against such a powerful attacker, an appealing approach is to use external crypto services to provide an extra layer of protection. This is a central idea in password-hardening (PH) services [2, 3]. In the context of PH, an online service provider who is providing services to end users is itself a client of a crypto server providing PH services. Hereinafter, we call the online service provider as the *server* and the crypto server as the *rate-limiter*[1]. When an end user registers with the server, the latter cooperates with the rate-limiter to jointly create a record which encrypts the password of the end user. Later, when this end user logs in with a candidate password, the server cooperates with the rate-limiter again to

[1]Lai *et al.* [3] call them the client and the server respectively.

check if the candidate password is identical to the one encrypted in the corresponding record.

The cooperation requirement above implies PH performs a double encryption of the passwords. What make PH interesting is its set of four fundamental guarantees tailored to practical deployment. First, the server (or the rate-limiter) alone is unable to check whether a candidate password is correct. This means the best strategy for any attacker who has fully compromised the server is to launch online (instead of offline) attacks. Second, the rate-limiter can track the number of unsuccessful login attempts of each end user, and rate-limit password validation requests, and hence online attacks, on a per-user basis. The third guarantee is that the rate-limiter learns no information about the passwords, meaning that PH is not just "transferring" the problem to the rate-limiter. Lastly, if either the server or the rate-limiter is compromised, or if the secret keys are in use for quite some time, the parties can jointly execute a key-rotation mechanism to refresh their secret keys. Furthermore, the key-rotation is seamless to the end users and requires arguably minimal help from the rate-limiter. Specifically, the server can locally update the records of its end users without interacting with the rate-limiter or the end users. This proactive mechanism provides *forward security*.

These strong security guarantees of PH make it very difficult for an attacker to get access to the passwords of the end users, even if the server is fully compromised. However, the protection of PH is confined to just the password itself. An attacker who fully compromises the server can simply decrypt any encrypted database and retrieve all other related data in it.

1.2 Password-Hardened Encryption

The problem of PH services stems from its limitation of functionality. In an abstract sense, PH can only "encrypt" a special message: the password. Decryption is not possible; one can just test whether a given message is encrypted. It is thus not suitable for encrypting general messages. In other words, PH only provides *authentication*. To solve this problem, we propose password-hardened encryption (PHE) services, which is an extension of PH services that goes beyond authentication and uses the passwords to secure general data in addition to the passwords. PHE aims to ensure that any attacker who can compromise the storage of these encrypted data cannot decrypt directly.

The formulation of PHE is similar to that of PH described above, with the following key differences. When an end user registers, the server and the rate-limiter jointly create a record which not only encrypts the user password but also a secret message. The message can be a freshly generated key for a symmetric key encryption scheme (*e.g.*, AES). The server then encrypts any sensitive information belonging to this end user with this key, and discards the key after encryption. Later, when the end user logs in, the server and the rate-limiter jointly validate the given candidate password. If and only if the password is correct, the server can then recover the key and proceed to decrypt the sensitive user information. Figure 1 depicts the basic workflow of a PHE scheme.

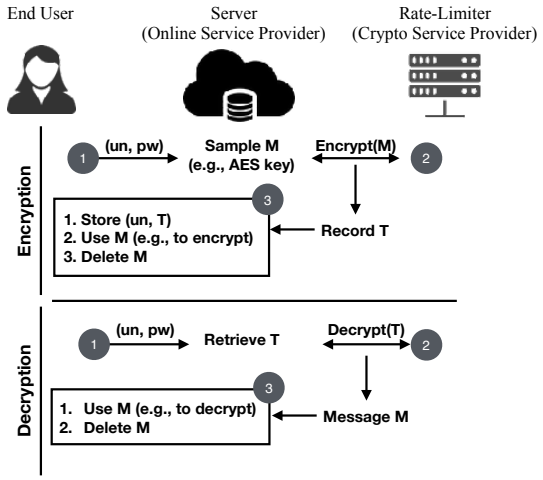

Figure 1: General Workflow of PHE

PHE inherits all four fundamental security guarantees provided by PH, with the protection of passwords is extended to additional secret messages as well. In particular, PHE inherits the key rotation capability. This makes PHE an appealing approach, for example, to conform to PCI DSS which requires credit card information to be encrypted by a mechanism supporting key rotation.

With per-user secret messages, each user can now enjoy the benefit of encrypting their respective data using an independent key. Data leakage is thus limited even if some of the keys are compromised. More importantly, if the server decides to rotate not only its own secret key but also some of the (data-)keys, the rotation is not as costly as re-encrypting the whole database.

In a nutshell, PHE is a one-package data-security solution for online service providers who employ password-based authentication and store sensitive user data.

1.2.1 General Applicability of PHE

PHE can be applied to any scenarios where a password-based authentication system is employed to protect user data, as a cryptographic replacement to access-control-based protection. For example, it can be used by online retail stores and e-commerce providers to encrypt

credit card numbers and especially the CVV (card verification values). It can also be used as a more secure password vault, where the user password serves as a master password for encrypting other (high-entropy) passwords (with the aid of the rate-limiter).

1.3 Our Contributions

Our contributions can be summarized as follows:

- We introduce and formalize the notion of PHE in order to protect arbitrary user data while retaining the functionality and security features of the underlying PH. The definitional framework encompasses dedicated cryptographic games as well as a soundness property that is stronger than the one commonly adopted in PH services, inasmuch as it puts less trust on the good behavior of the rate-limiter.

- We propose a remarkably simple PHE construction. Its novelty lies in the fact that it reduces the number of number-theoretic operations (in particular, dispenses from the implicit use of ElGamal encryption) in previous PH services, despite providing stronger security guarantees.

- Our PHE instantiation is between 30% and 50% more efficient than previous PH (without E) constructions. Our extensive experimental evaluation demonstrated that our PHE scheme is highly efficient ($\sim$ 10ms per request) and scales well to high-throughput scenarios.

- We prove the security of our construction in the random oracle model under the decisional Diffie-Hellman (DDH) assumption.

1.4 Technical Overview

1.4.1 A Simpler and More Efficient Construction

To appreciate the technical contribution brought by our PHE construction, we first consider a natural attempt which builds PHE by using PH as a blackbox. Such a generic construction will likely require the use of zero-knowledge proof systems for a complex language dependent on the PH scheme. Since our aim is to build a practical scheme which is plausible for deployment, we decide to modify the construction of the PH scheme Phoenix [3] in a non-black-box way to become a PHE scheme. More interestingly, it turns out that a major component in the construction of Phoenix – a variant of the Cramer-Shoup encryption scheme [4] – is unnecessary. With this observation, we design an extremely simple PHE scheme (which also gives a much simpler PH scheme) as follows.

To encrypt the message M under the password pw, the server and the rate-limiter sample random nonces $n_{\mathcal{S}}$ and $n_{\mathcal{R}}$ respectively, and jointly compute

$$(H^x_{\mathcal{R},0}H^y_{\mathcal{S},0},\ \ H^x_{\mathcal{R},1}H^y_{\mathcal{S},1}M^y)$$

where $H_{\mathcal{R},b} = \mathsf{H}_{\mathcal{R}}(n_{\mathcal{R}}, \mathsf{pw}, b)$ and $H_{\mathcal{S},b} = \mathsf{H}_{\mathcal{S}}(n_{\mathcal{S}}, \mathsf{pw}, b)$ are (multiplicative) group elements output by hash functions $\mathsf{H}_{\mathcal{R}}$ and $\mathsf{H}_{\mathcal{S}}$, $b \in \{0,1\}$, and x and y are the secret keys of the rate-limiter and the server respectively.

To decrypt with the password pw, the server computes $H^y_{\mathcal{S},0}$ to recover the hash value $H^x_{\mathcal{R},0}$, and sends the latter to the rate-limiter. Upon verifying the correctness of $H^x_{\mathcal{R},0}$, the rate-limiter returns $H^x_{\mathcal{R},1}$. The server then computes the value $H^y_{\mathcal{S},1}$. Together with $H^x_{\mathcal{R},1}$, the server can then recover M.

1.4.2 Stronger Soundness using Efficient Proofs

We observe that in the existing definition of PH [3], in the case where the rate-limiter rejects in the validation phase, it is indistinguishable to the server whether the rate-limiter refuses to entertain the validation request (even when the password is correct) or the password indeed does not match the record. To address this issue, we define the (strong) soundness property, which requires the rate-limiter to explain not only the reasons for acceptance, but also for rejections.

In any real-world instantiation with strong soundness, compromised/cheating rate-limiters which (selectively) prevent legitimate logins (using correct passwords) can be detected. It further means that external parties can serve as rate-limiters with minimal trust requirements.

To achieve our newly defined soundness, additional zero-knowledge proofs need to be generated by the rate-limiter during both encryption and decryption. This does not impact efficiency in any significant way, as confirmed by our experimental evaluation.

1.4.3 Strengthened yet Simplified Definitions

From the viewpoint of extending the definition of PH to that of PHE, we made the following contributions other than the stronger soundness requirement. Firstly, all security experiments are modified to reflect attacks against not only the passwords but also the secret data to be encrypted. Furthermore, most of the syntax and security experiments are more refined and simplified when comparing to their PH counterparts [3]. For example, in the definition of PH [3], the username of an end user serves as a common input to both the server and the rate-limiter in the enrollment and validation protocols. This input is actually unnecessary (in the security definition nor in the construction) and is not present in our definition.

In short, apart from adding encryption and decryption functionalities, we also make several improvements which can also be applicable to PH schemes, in terms of both definition and construction.

1.5 Related Work

We first briefly recap PH schemes, then overview other cryptographic primitives which offer related security guarantees but not those fundamental to PHE / PH.

1.5.1 Password-Hardening Services

Everspaugh et al. [2] introduced the notion of PH services to replace salted hashes for login validation. Key-rotation is also identified as an important property to "heal the system" after compromise [2].

While Everspaugh et al. [2] formally defined partially-oblivious pseudorandom functions (PO-PRF) services, and informally suggested PH as an application, the subsequent work by Schneider *et al.* [5] attempted to give a formal definition (of a closely related notion called partially-oblivious commitments) and a scheme provably secure under the said definition. Unfortunately, the definition of Schneider *et al.* [5] was shown by Lai *et al.* [3] to be flawed, as they discovered a devastating attack to the scheme of Schneider *et al.* [5] which extracts user passwords. To capture such attacks Lai *et al.* [3] gave a new security definition. They also proposed a scheme Phoenix which is secure under the new definition.

PHE services extend the security of PH as defined by Lai *et al.* [3] to messages, such that encrypted messages can only be decrypted with the correct password and the help of the external rate-limiter.

Finally, we stressed again that our PHE not only performs much better than the possible approach of applying generic zero-knowledge proof to "glue" PH with an encryption, but also leads to an implicit PH scheme which is even more efficient than the state-of-the-art [3].

1.5.2 Password-Protected Secret Sharing

The main goal of password-protected secret sharing (PPSS) or password-authenticated key-exchange (PAKE) is also to protect a secret message (of an end-user, with the help of possibly more than one server) in such a way that it can only be recovered using the correct password. Unlike the "game-based style" definition used in this work and in PH, the security of state-of-the-art PPSS/PAKE schemes [6, 7] is usually proven in "simulation style" under the UC framework [8].

PPSS in the public-key model implies threshold PAKE [9] so we focus on PPSS. While it seems that PHE can be constructed from PPSS by having the server hold one of the shares and the rate-limiter hold the other, the resulting scheme lacks important features of PHE.

Per-user rate-limiting. While global rate-limiting is trivial, note that PHE schemes additionally allow (and require) the rate-limiter to count the number of unsuccessful login attempts of each user, and refuse to provide decryption services to the server for a certain user (indirectly) if the latter has attempted too many unsuccessful logins. Existing PPSS schemes do not, nor can be easily extended to, support per-user rate-limiting.

Key-rotation. Most PPSS schemes do not support key-rotation. The only existing scheme with key-rotation [10] is very inefficient: It requires "a few hundred exponentiations" per number of shares [10].

1.5.3 Distributed Password Verification

Distributed password verification (DPV) protocols [11] also requires the online service provider to seek help from external crypto servers for verifying user passwords. Moreover, both notions explicitly feature key-rotation mechanisms. Yet, unlike PH, DPV does not explicitly support per-user rate limiting, nor can the existing construction [11] be modified to support it. Unlike PHE, DPV does not provide encryption functionality.

1.5.4 Other Related Work

Hidden credential retrieval (HCR) [12] also considers having a crypto service to unlock credentials for users who hold low-entropy passwords. Not protected by other mechanisms, the crypto service in HCR can launch an inevitable offline dictionary attack to recover the user credential. HCR does not support key rotation either.

Password-based key-derivation or encryption [13–15] encrypts messages directly using keys derived from passwords. As typical passwords have low entropy, salt values are also used. Yet, it is still vulnerable to brute-force attacks by an attacker who obtained the salts database.

2 Password-Hardened Encryption (PHE)

We formalize password-hardened encryption, an extension of password-hardening, for encrypting messages which can only be decrypted by the user password, the secret keys of both the server and the rate-limiter.

2.1 Definition of PHE

Let 1^λ be a λ-bit unary string of 1 which represents the security parameter. Let $\mathcal{P}$ and $\mathcal{M}$ be the password space and message space respectively. Let $\mathcal{S}$ and $\mathcal{R}$ refer to

the server and the rate-limiter respectively. We denote by $(u,v) \leftarrow\!\!\$\, P^{\ell}\langle \mathcal{S}(x), \mathcal{R}(y)\rangle$ the protocol P executed by the parties $\mathcal{S}$ and $\mathcal{R}$ with common input ℓ, local inputs x and y, and local outputs u and v respectively. We denote the empty string with ε.

A *password-hardened encryption* (PHE) scheme consists of the efficient algorithms and protocols $(\mathsf{Setup}, \mathsf{KGen}_{\mathcal{S}}, \mathsf{KGen}_{\mathcal{R}}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Rotate}, \mathsf{Update})$, which we define as follows:

Setup and Key Generation. The following algorithms initialize our PHE system.

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda})$. The *setup algorithm* generates the public parameters pp.

$(\mathsf{pk}_{\mathcal{S}}, \mathsf{sk}_{\mathcal{S}}) \leftarrow \mathsf{KGen}_{\mathcal{S}}(\mathsf{pp})$. The server runs $\mathsf{KGen}_{\mathcal{S}}(\mathsf{pp})$ to generate a key-pair $(\mathsf{pk}_{\mathcal{S}}, \mathsf{sk}_{\mathcal{S}})$.

$(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}) \leftarrow \mathsf{KGen}_{\mathcal{R}}(\mathsf{pp})$. The rate-limiter runs $\mathsf{KGen}_{\mathcal{R}}(\mathsf{pp})$ to generate a key-pair $(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$.

We assume that all parties take pp, $\mathsf{pk}_{\mathcal{S}}$, and $\mathsf{pk}_{\mathcal{R}}$ as inputs in all algorithms and protocols.

Encryption. When an end user registers for an account with password $\mathsf{pw} \in \mathcal{P}$ and a secret message $M \in \mathcal{M}$ (*e.g.*, an AES key, which can also be chosen by the server on behalf of the end user), the server engages in the (labeled) encryption protocol with the rate-limiter $\mathcal{R}$ to compute a record T with label ℓ':

$((\ell', T), \varepsilon) \leftarrow \mathsf{Encrypt}^{\ell}\langle \mathcal{S}(\mathsf{sk}_{\mathcal{S}}, \mathsf{pw}, M), \mathcal{R}(\mathsf{sk}_{\mathcal{R}})\rangle$.

The server $\mathcal{S}$ inputs a secret key $\mathsf{sk}_{\mathcal{S}}$, a password $\mathsf{pw} \in \mathcal{P}$, a message $M \in \mathcal{M}$. The rate-limiter $\mathcal{R}$ takes as inputs a secret key $\mathsf{sk}_{\mathcal{R}}$. Both parties take a common input label $\ell = (\ell_{\mathcal{S}}, \ell_{\mathcal{R}})$. When the protocol concludes, $\mathcal{S}$ outputs a record T with a label $\ell' = (\ell'_{\mathcal{S}}, \ell'_{\mathcal{R}})$. $\mathcal{R}$ outputs nothing, denoted by the empty string ε.

We assume the convention that $\ell' = \ell$ or $\ell = \varepsilon$. The first condition is an exception which only appears in the definition of forward-security, while the second holds in all other situations, including normal executions in real-world applications. In this case ℓ' is sampled during the protocol execution. The label ℓ' consists of $\ell'_{\mathcal{S}}$ and $\ell'_{\mathcal{R}}$, which can be interpreted as the session identifiers or the pseudonyms of the end user assigned by the server and the rate-limiter respectively.

Decryption. When an end user logs in to the service provided by the server with a candidate password $\mathsf{pw} \in \mathcal{P}$, the server retrieves the corresponding encryption record T and label ℓ for the user, and engages in the (labeled) decryption protocol with the rate-limiter:

$((f, M), \varepsilon) \leftarrow \mathsf{Decrypt}^{\ell}\langle \mathcal{S}(\mathsf{sk}_{\mathcal{S}}, \mathsf{pw}, T), \mathcal{R}(\mathsf{sk}_{\mathcal{R}})\rangle$.

The server $\mathcal{S}$ inputs its secret key $\mathsf{sk}_{\mathcal{S}}$, the candidate password pw, and the retrieved record T. The rate-limiter $\mathcal{R}$ inputs its secret key $\mathsf{sk}_{\mathcal{R}}$. Both parties take a common input (non-empty) label ℓ[2]. The server outputs a flag f and a message M. The flag f is either $\perp$ to indicate failure (the rate-limiter aborts), 0 if the record or the password is invalid, or 1 for a successful login. The rate-limiter outputs nothing, *i.e.*, the empty string ε.

Key Rotation and Record Update. The server $\mathcal{S}$ and the rate-limiter $\mathcal{R}$ may decide to rotate their keys and update the records, which can be due to a regular routine or a compromise at either side. The process consists of two steps, performed without involving any end user.

First, $\mathcal{S}$ and $\mathcal{R}$ engage in a key rotation protocol to rotate their keys and compute an update token.

$((\mathsf{pk}'_{\mathcal{S}}, \mathsf{sk}'_{\mathcal{S}}, \tau), (\mathsf{pk}'_{\mathcal{R}}, \mathsf{sk}'_{\mathcal{R}})) \leftarrow \mathsf{Rotate}\langle \mathcal{S}(\mathsf{sk}_{\mathcal{S}}), \mathcal{R}(\mathsf{sk}_{\mathcal{R}})\rangle$. Their input is the respective secret key $\mathsf{sk}_{\mathcal{S}}$ and $\mathsf{sk}_{\mathcal{R}}$. When Rotate concludes, $\mathcal{S}$ outputs a rotated key-pair $(\mathsf{pk}'_{\mathcal{S}}, \mathsf{sk}'_{\mathcal{S}})$, and an update token τ. $\mathcal{R}$ outputs a rotated key-pair $(\mathsf{pk}'_{\mathcal{R}}, \mathsf{sk}'_{\mathcal{R}})$.

With the token, $\mathcal{S}$ then *locally* runs an update algorithm on each record T with label ℓ.

$T' \leftarrow \mathsf{Update}^{\ell}(\tau, T)$. On input a label ℓ, an update token τ, and a record T (which encrypts some message M with label ℓ), the update algorithm outputs a new record T' (also encrypting M with label ℓ).

One may consider a general treatment of update which allows changing the encrypted message M. For simplicity, we assume that M remains unchanged.[3]

Correctness. A PHE is correct whenever all honestly generated records can be successfully decrypted to recover the encrypted message with the correct password. Moreover, if a record passes decryption with respect to some secret keys, then the updated record also passes decryption with respect to the rotated keys. Since correctness is subsumed by soundness and forward security, we omit the formal definition.

2.2 Security of PHE

PHE is secure against persistent attackers. Intuitively key-rotation can be seen as structuring the PHE protocol execution into separate rounds. In each round, the attacker can compromise either the rate-limiter or the server and use whatever he learned in the next round without gaining additional advantage (as formalized in Forward Security).

[2] Equivalently, one can think of $\ell_{\mathcal{R}}$ where $\ell = (\ell_{\mathcal{S}}, \ell_{\mathcal{R}})$ as part of the first message sent from $\mathcal{S}$ to $\mathcal{R}$ during the execution of the protocol.

[3] In some scenarios, updating the messages in a certain meaningful way should require the consent of the user (*i.e.*, the involvement of the user to supply the password), or expect the accompanying system supports some advanced functionalities (*e.g.*, when M is used as the secret key of AES, it is only useful if AES supports "efficient re-encryption").

$\mathsf{Hid}^b_{\mathsf{PHE},\mathcal{A}}(1^\lambda)$

1 : $\mathsf{pp} \leftarrow_\$ \mathsf{Setup}(1^\lambda)$, $(\mathsf{pk}_\mathcal{R}, \mathsf{sk}_\mathcal{R}) \leftarrow_\$ \mathsf{KGen}_\mathcal{R}(\mathsf{pp})$
2 : $\mathbb{O} := \{P\langle \cdot, \mathcal{R}(\mathsf{sk}_\mathcal{R}, \ldots)\rangle :$
3 : $\quad P \in \{\mathsf{Encrypt}^\varepsilon, \mathsf{Decrypt}^\ell, \mathsf{Rotate} : \ell \in \{0,1\}^*\}\}$
4 : // All rate-limiter outputs are given to adversary,
5 : $\quad$ // except for $\mathsf{sk}'_\mathcal{R}$ from $\mathsf{Rotate}\langle \cdot, \mathcal{R}(\mathsf{sk}_\mathcal{R})\rangle$.
6 : // $\mathsf{Rotate}\langle \cdot, \mathcal{R}(\mathsf{sk}_\mathcal{R})\rangle$ updates $\mathsf{sk}_\mathcal{R}$ embedded in all oracles to $\mathsf{sk}'_\mathcal{R}$.
7 : $(\mathsf{sk}^*_\mathcal{S}, \chi, M^*_0, M^*_1, \mathsf{st}) \leftarrow_\$ \mathcal{A}_1^\mathbb{O}(\mathsf{pp}, \mathsf{pk}_\mathcal{R})$
8 : $\mathsf{pw}^* \leftarrow_\$ \chi$
9 : $((\ell^*, T^*), \varepsilon) \leftarrow_\$ \mathsf{Encrypt}^\varepsilon\langle \mathcal{S}(\mathsf{sk}^*_\mathcal{S}, \mathsf{pw}^*, M^*_b), \mathcal{R}(\mathsf{sk}_\mathcal{R})\rangle$
10 : $b' \leftarrow_\$ \mathcal{A}_2^\mathbb{O}(\mathsf{st}, \ell^*, T^*)$
11 : **return** b'

Figure 2: Message Hiding Experiment

Both our definitions and our construction assume a secure channel when executing honest interactions between server and rate-limiter. This assumption is also made implicitly in the original definition of PH [3]. Practically this implies using a TLS connection between rate-limiter and server, and updating long-term keys and certificates during key-rotation.

We formalize the security properties of PHE, extending those from password-hardening [3]. This obviously makes a secure PHE scheme also a secure PH scheme.

Message Hiding (Figure 2). Strengthening the (password-)hiding property of PH, the encrypted message corresponding to a record should also remain hidden even if the the server (and its secret key) is compromised. Specifically, message hiding requires that an adversary cannot distinguish whether a record T^* is encrypting M^*_0 or M^*_1, even if these messages as well as the distribution of the password is chosen by the adversary. However, since by functionality the message can be recovered by engaging in the decryption protocol with the rate-limiter using the correct password, the highest possible security level that we can hope for is upper-bounded by the entropy of the password. Our formalization covers this by parameterizing the winning condition of the adversary using the distribution of the passwords.

Formally, we model message hiding as an experiment $\mathsf{Hid}^b_{\mathsf{PHE},\mathcal{A}}(1^\lambda)$ participated by a 2-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ acting as the malicious server and a challenger acting as the honest rate-limiter. The adversary $\mathcal{A}_1$ gets access to the encryption, decryption, and key update oracles on chosen inputs. Eventually, $\mathcal{A}_1$ outputs a server secret key $\mathsf{sk}^*_\mathcal{S}$, a password distribution χ, two messages M^*_0 and M^*_1, and a state st.

The challenger picks a random password pw^* from the distribution χ, and encapsulates M^*_b into a record T^* with label ℓ^* honestly using $\mathsf{sk}^*_\mathcal{S}$ and pw^* by locally emulating the encryption protocol. Note that the communication transcript of the emulation is not given to $\mathcal{A}$. Intuitively this is justified because the server was honest while the record was created and we assume a secure channel.

Finally, $\mathcal{A}_2$ gets ℓ^* and T^*, and must guess whether M^*_0 or M^*_1 is encrypted by outputting a guess b', which is also output by the experiment.

Definition 1 (Message Hiding) *A PHE* PHE *is* message hiding *if, for any* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr\left[\mathsf{Hid}^0_{\mathsf{PHE},\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Hid}^1_{\mathsf{PHE},\mathcal{A}}(1^\lambda) = 1\right]\right| \leq 2\sum_{i=1}^{Q} p_i + \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of the experiment, p_i is the probability of the i-th most probable event in the distribution χ specified by the adversary, and Q is the number of times that $\mathcal{A}_2$ queries $\mathsf{Decrypt}^\ell\langle \cdot, \mathcal{R}(\mathsf{sk}_\mathcal{R})\rangle$ with input label $\ell = (\cdot, \ell^*_\mathcal{R})$[4].

Partial Obliviousness (Figure 3). Our formalization of partial obliviousness follows the recent definition for PH [3] closely, but is adapted to our PHE setting. This property hides the password and the encrypted message against a malicious rate-limiter, *e.g.*, during the execution of the encryption and decryption protocols. It is partial in the sense that it does not guarantee the anonymity of the end user. In particular, it might be possible for the rate-limiter to link executions of the encryption and decryption protocols triggered by the same end user.

Formally, we model partial obliviousness as an experiment $\mathsf{Obl}^b_{\mathsf{PHE},\mathcal{A}}(1^\lambda)$ participated by a 3-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ acting as the malicious rate-limiter, and a "challenger" acting as the honest server. Initially, $\mathcal{A}_1$ can interact through the oracles $\mathbb{O}$ (denoted by $\mathcal{A}_1^\mathbb{O}$) with the challenger in the protocols for encryption, decryption, and key-rotation on inputs of its choice. The server outputs of the protocols are given to $\mathcal{A}$, with the obvious exception of the rotated secret key from the rotation protocol. Eventually, $\mathcal{A}_1$ outputs two password-message pairs $(\mathsf{pw}^*_0, M^*_0, \mathsf{pw}^*_1, M^*_1)$, with some state information st to be passed to $\mathcal{A}_2$. The password pw^*_b and message M^*_b, where b is specified by the experiment, is called the "challenge password" and "challenge message" respectively.

[4]The constant 2 in the upper bound is due to the specific style and proof technique which we do not think is inherent: We will eventually show that, for our construction, $\mathsf{Hid}^b_{\mathsf{PHE},\mathcal{A}}$ for both $b \in \{0,1\}$ are indistinguishable to a hybrid experiment except with probability $\sum_{i=1}^{Q} p_i + \mathsf{negl}(\lambda)$. Taking the union bound yields the constant 2.

$\mathsf{Obl}^b_{\mathsf{PHE},\mathcal{A}}(1^\lambda)$

1 : $\mathsf{pp} \leftarrow\!\!\$\, \mathsf{Setup}(1^\lambda)$, $(\mathsf{pk}_\mathcal{S}, \mathsf{sk}_\mathcal{S}) \leftarrow\!\!\$\, \mathsf{KGen}_\mathcal{S}(\mathsf{pp})$
2 : $\mathbb{O} := \{P\langle \mathcal{S}(\mathsf{sk}_\mathcal{S}, \ldots), \cdot\rangle :$
3 : $\quad P \in \{\mathsf{Encrypt}^\varepsilon, \mathsf{Decrypt}^\ell, \mathsf{Rotate} : \ell \in \{0,1\}^*\}\}$
4 : $(\mathsf{pw}_0^*, M_0^*, \mathsf{pw}_1^*, M_1^*, \mathsf{st}) \leftarrow\!\!\$\, \mathcal{A}_1^{\mathbb{O}}(\mathsf{pp}, \mathsf{pk}_\mathcal{S})$
5 : // All server outputs are given to $\mathcal{A}$, except for $\mathsf{sk}'_\mathcal{S}$ from $\mathsf{Rotate}\langle \mathcal{S}(\mathsf{sk}_\mathcal{S}), \cdot\rangle$.
6 : // $\mathsf{Rotate}\langle \mathcal{S}(\mathsf{sk}_\mathcal{S}), \cdot\rangle$ updates $\mathsf{sk}_\mathcal{S}$ embedded in all oracles to $\mathsf{sk}'_\mathcal{S}$.
7 : $((\ell^*, T^*), \mathsf{st}) \leftarrow\!\!\$\, \mathsf{Encrypt}^\varepsilon\langle \mathcal{S}(\mathsf{sk}_\mathcal{S}, \mathsf{pw}_b^*, M_b^*), \mathcal{A}_2(\mathsf{st})\rangle$
8 : // The server output (f, m) from $\mathsf{Decrypt}^\ell\langle \mathcal{S}(\mathsf{sk}_\mathcal{S}, \ldots), \cdot\rangle$ is withheld from $\mathcal{A}$
9 : $\quad$ // if $(\ell, \mathsf{pw}) = ((\ell^*_\mathcal{S}, \cdot), \mathsf{pw}_0^*)$ or $((\ell^*_\mathcal{S}, \cdot), \mathsf{pw}_1^*)$.
10 : $b' \leftarrow\!\!\$\, \mathcal{A}_3^{\mathbb{O}}(\mathsf{st}, \ell^*, T^*)$
11 : **return** b'

Figure 3: Partial Obliviousness Experiment

The challenger, acting as the server, then engages in the encryption protocol using the empty label, the challenge password, and the challenge message with the adversary $\mathcal{A}_2$ acting as the rate-limiter. Upon termination, the challenger outputs a record T^* with label ℓ^* and sends them to $\mathcal{A}_3$. The adversary $\mathcal{A}_2$ outputs a state state which will also be passed to $\mathcal{A}_3$.

After the generation of the challenge record T^*, $\mathcal{A}_3$ can still interact with the challenger through the oracles, except that the decryption oracle will no loner return the decryption result to $\mathcal{A}$, if it is queried on inputs containing $(\ell^*, \mathsf{pw}_0^*)$ or $(\ell^*, \mathsf{pw}_1^*)$. This prevents $\mathcal{A}$ from winning trivially. Eventually, $\mathcal{A}_3$ outputs a guess b' of which password-message pair is chosen as the challenge. The experiment then simply outputs the value b'.

Definition 2 (Partial Obliviousness) *A PHE* PHE *is* partially oblivious *if, for any three-stage* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

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

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

Soundness (Figure 4 and Figure 5). Soundness (Figure 4) ensures that if a record and its encrypted message are generated by an honest server and a (possibly malicious) rate-limiter, then the message can be recovered by engaging in the decryption protocol using the correct password (unless the rate-limiter aborts). On the other hand, decrypting using an incorrect password is guaranteed to yield $f = 0$ (unless the rate-limiter aborts). This property arguably suffices for practical applications.

To make the rate-limiter even more accountable, the strong soundness property guarantees all properties of

$\mathsf{Soundness}_{\mathsf{PHE},\mathcal{A}}(1^\lambda)$

1 : $(\mathsf{pk}_\mathcal{R}, \mathsf{sk}_\mathcal{S}, \mathsf{pw}, \mathsf{pw}', M, \mathsf{st}) \leftarrow\!\!\$\, \mathcal{A}_1(1^\lambda)$
2 : $((\ell, T), \mathsf{st}) \leftarrow\!\!\$\, \mathsf{Encrypt}^\varepsilon\langle \mathcal{S}(\mathsf{sk}_\mathcal{S}, \mathsf{pw}, M), \mathcal{A}_2(\mathsf{st})\rangle$
3 : $((f, M'), \mathsf{st}) \leftarrow\!\!\$\, \mathsf{Decrypt}^\ell\langle \mathcal{S}(\mathsf{sk}_\mathcal{S}, \mathsf{pw}', T), \mathcal{A}_3(\mathsf{st})\rangle$
4 : $b_0 \leftarrow (f \neq \perp)$
5 : $b_1 \leftarrow (\mathsf{pw} = \mathsf{pw}' \wedge (f \neq 1 \vee M \neq M'))$
6 : $b_2 \leftarrow (\mathsf{pw} \neq \mathsf{pw}' \wedge f \neq 0)$
7 : **return** $b_0 \wedge (b_1 \vee b_2)$

Figure 4: Soundness Experiment

$\mathsf{StrongSoundness}_{\mathsf{PHE},\mathcal{A}}(1^\lambda)$

1 : $(\mathsf{pk}_\mathcal{R}, \mathsf{sk}_\mathcal{S}, \ell, \ell', \mathsf{pw}, \mathsf{pw}', T, \mathsf{st}) \leftarrow\!\!\$\, \mathcal{A}'(1^\lambda)$
2 : $((f, M), \mathsf{st}) \leftarrow\!\!\$\, \mathsf{Decrypt}^\ell\langle \mathcal{S}(\mathsf{sk}_\mathcal{S}, T, \mathsf{pw}), \mathcal{A}_2(\mathsf{st})\rangle$
3 : $((f', M'), \mathsf{st}) \leftarrow\!\!\$\, \mathsf{Decrypt}^{\ell'}\langle \mathcal{S}(\mathsf{sk}_\mathcal{S}, T, \mathsf{pw}'), \mathcal{A}_3(\mathsf{st})\rangle$
4 : $b_0 \leftarrow (\perp \notin \{f, f'\})$ // Rate-limiter does not abort.
5 : $b_1 \leftarrow ((\ell, \mathsf{pw}) = (\ell', \mathsf{pw}') \wedge (f, M) \neq (f', M'))$
6 : // Same labels and passwords, different behaviors
7 : $b_2 \leftarrow ((\ell, \mathsf{pw}) \neq (\ell', \mathsf{pw}') \wedge f = f' = 1)$
8 : // Record is valid under different label-password pairs
9 : **return** $b_0 \wedge (b_1 \vee b_2)$

Figure 5: Strong Soundness Experiment

soundness, with some additional ones (Figure 5). These additional requirements are similar to those in the binding property of PH. Specifically, we additionally require that, even for a maliciously generated record, it is infeasible for the malicious rate-limiter to behave inconsistently without getting caught (assuming that it does not abort). The inconsistent behaviors include: 1) convince the server to output differently when decrypting the same record using the same password and the same label; 2) convince the server that the record is valid when decrypting with different label-password pairs.

Definition 3 ((Strong) Soundness) *A PHE* PHE *is* sound *if, for any* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

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

Furthermore, it is strongly sound if it also holds that

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

The probabilities are taken over the random coins of the experiments.

Forward Security (Figure 6). The key rotation phase should *heal* the system in the sense that it renders the old

$\mathsf{FwdSec}^{b}_{\mathsf{PHE},\mathcal{A}}(1^\lambda)$

1 : $\mathsf{pp} \leftarrow_{\mathrm{s}} \mathsf{Setup}(1^\lambda)$
2 : $(\mathsf{sk}_{\mathcal{S}}, \mathsf{sk}_{\mathcal{R}}, \{(\ell_i, \mathsf{pw}_i, T_i)\}_{i=1}^{n}, \mathsf{st}) \leftarrow_{\mathrm{s}} \mathcal{A}_1(\mathsf{pp})$
3 : / for some $n = \mathsf{poly}(\lambda)$
4 : $\forall i \in [n],\ ((f_i, M_i), \varepsilon) \leftarrow \mathsf{Decrypt}^{\ell_i}\langle \mathcal{S}(\mathsf{sk}_{\mathcal{S}}, \mathsf{pw}_i, T_i), \mathcal{R}(\mathsf{sk}_{\mathcal{R}})\rangle$
5 : **if** $b = 0$ **then**
6 : $\quad((\mathsf{pk}'_{\mathcal{S}}, \mathsf{sk}'_{\mathcal{S}}, \tau), (\mathsf{pk}'_{\mathcal{R}}, \mathsf{sk}'_{\mathcal{R}})) \leftarrow_{\mathrm{s}} \mathsf{Rotate}\langle \mathcal{S}(\mathsf{sk}_{\mathcal{S}}), \mathcal{R}(\mathsf{sk}_{\mathcal{R}})\rangle$
7 : $\quad\forall i \in [n], T'_i \leftarrow_{\mathrm{s}} \mathsf{Update}^{\ell_i}(\tau, T_i)$
8 : **else**
9 : $\quad(\mathsf{pk}'_{\mathcal{S}}, \mathsf{sk}'_{\mathcal{S}}) \leftarrow_{\mathrm{s}} \mathsf{KGen}_{\mathcal{S}}(\mathsf{pp}),\ (\mathsf{pk}'_{\mathcal{R}}, \mathsf{sk}'_{\mathcal{R}}) \leftarrow_{\mathrm{s}} \mathsf{KGen}_{\mathcal{R}}(\mathsf{pp})$
10 : $\quad\forall i \in [n], ((\ell'_i, T'_i), \varepsilon) \leftarrow_{\mathrm{s}} \mathsf{Encrypt}^{\ell_i}\langle \mathcal{S}(\mathsf{sk}'_{\mathcal{S}}, \mathsf{pw}_i, M_i), \mathcal{R}(\mathsf{sk}'_{\mathcal{R}})\rangle$
11 : $\quad$/ By the assumed convention, $\ell'_i = \ell_i\ \forall i \in [n]$
12 : **endif**
13 : $b' \leftarrow_{\mathrm{s}} \mathcal{A}_2(\mathsf{st}, \mathsf{sk}'_{\mathcal{S}}, \mathsf{sk}'_{\mathcal{R}}, T'_1, \ldots, T'_n)$
14 : **return** $((\forall i \in [n],\ f_i = 1) \wedge b')$

Figure 6: Forward Security Experiment

secret keys of the server and the rate-limiter useless to the adversary. The old secret keys should not help the adversary in recovering information from an updated record. On the other hand, the rotated keys and updated records should function the same as freshly generated keys and records respectively.

In order not to consider all possible sequences of corruption of the server and the rate-limiter in all security properties, we adopt the approach in the original PH definition [3] to define a strong notion of forward security. This property ensures that even for maliciously generated secret keys for both the server and the rate-limiter, and maliciously generated records, the rotated keys and updated records are indistinguishable to freshly generated keys and records respectively, except for the information that is preserved for ensuring functionality, *e.g.*, the encrypted messages and the labels.

Unlike the original definition [3], our definition allows the adversary to generate multiple records. This definition seems not to be equivalent to the single-record variant, as an adversary against the single-record variant cannot simulate a challenger of the multi-record variant without knowing the update token chosen by the challenger of the single-record variant.

Definition 4 (Forward Security) *A PHE* PHE *is* forward secure *if for any two-stage* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function* $\mathsf{negl}(\lambda)$ *with*

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

where the probability is taken over the random coins of the experiments.

3 Our Construction

Since PHE is an extension of PH with encryption functionality, it is natural to construct a PHE scheme from an existing PH scheme (*e.g.*, [2, 3]). Recall that in a PH scheme, when a new end user registers, the server and the rate-limiter engage in an enrollment protocol and jointly create a record which "encrypts" the password of the end user. Later, when the end user logs in with a candidate password, the server and the rate-limiter can jointly verify whether the candidate password is valid.

3.1 Why Generic Construction Fails

Our first attempts are to construct PHE generically from PH or PO-PRF. Below, we discuss why these approaches are unsatisfactory.

3.1.1 Generic Construction from PH

At first glance, a PHE scheme might be built on top of a PH scheme, by additionally encrypting the message in the enrollment protocol, in such a way that it can be decrypted if and only if a valid candidate password is provided. Below, we sketch a plausible construction.

Suppose there exist a PH scheme and a public-key encryption (PKE) scheme which are both key-rotatable. In the encryption phase, the server and the rate-limiter engage in the enrollment protocol of the PH scheme. The server additionally encrypts the message using PKE to the rate-limiter. Later, in the decryption phase, the server and the rate-limiter engage in the validation protocol of the PH scheme. The server additionally requests the rate-limiter to decrypt a possibly blinded / rerandomized version of the ciphertext.

One can immediately notice that the above construction suffers from a mix-and-match attack: The server can request decryption of arbitrary combinations of enrollment records and ciphertexts in the decryption phase. One way to avoid this issue is to let the rate-limiter sign the record-ciphertext pairs as they are created, and let the server prove in zero-knowledge that a decryption request is on a record-ciphertext pair which is blinded / rerandomized from another pair for which it possesses a signature. However, such an approach seems inefficient since the server likely needs to prove complex statements involving PH protocol execution, PKE encryption, and signature verification.

3.1.2 Generic Construction from PO-PRF Services

We also investigate the possibility of building PHE generically from PO-PRF. Similar to the construction of

symmetric-key encryption from PRFs, where a ciphertext C which encrypts message M using key k is computed as $C = (\mathsf{PRF}(k,r) \oplus M, r)$, one idea is to encrypt a message by the output of the PO-PRF as a one-time pad.

When instantiated with PYTHIA, the only known construction of PO-PRF, a PRF value is a group element $e(H_1(\mathsf{un}), H_2(\mathsf{pw}))^k$ in the target group of a cryptographic bilinear map e. A ciphertext of M would thus be $C = e(H_1(\mathsf{un}), H_2(\mathsf{pw}))^k \cdot M$. The problem with this approach is that, after key-rotation (from k to rk where r is a random field element), the corresponding ciphertext becomes $C' = C^r = e(H_1(\mathsf{un}), H_2(\mathsf{pw}))^{rk} \cdot M^r$, which encrypts M^r instead of M.

Another idea is to use the output of a PO-PRF as the secret key of a key-homomorphic encryption (KHE) scheme. However, recall that PRF values of PYTHIA are target group elements, and hence the companion KHE scheme must have target group elements as secret keys. Assuming the decryption algorithm of the KHE scheme only uses generic group operations, it seems rather difficult to "protect" the secret key, *i.e.*, one may infer the secret key from the ciphertext and its corresponding decryption result by "undoing" the generic group operations involved in decryption. Additional machinery such as another bilinear map might be needed. In other words, this approach needs a cryptographic trilinear map of which no known efficient construction exists.

3.2 Non-Blackbox Approach: Intuition

We adopt an alternative approach which upgrades the PH scheme Phoenix by Lai *et al.* [3] in a non-black-box way into an efficient PHE scheme. The transform is based on the observation below: In the validation protocol of Phoenix, the server first sends to the rate-limiter a PKE ciphertext encrypting a pseudorandom value generated by the rate-limiter. The latter decrypts the ciphertext and checks whether the pseudorandom value is well-formed, or equivalently whether the candidate password is valid. If so, it proves the well-formedness in zero-knowledge to the server. The rate-limiter essentially provides an "equality check service" to the server. With this observation, the idea is to turn such a service to a "conditional decryption service" where decryption is performed if the equality check is satisfied.

However, we can do even better. Observe that the use of PKE in Phoenix is actually not necessary: It does not offer protection against a malicious rate-limiter since the latter knows the decryption key anyway. It also does not offer protection against a malicious server, since the (password-)hiding property relies on the fact that the server must guess the correct password to derive (a ciphertext of) the pseudorandom value. We believe that the use of PKE in Phoenix is inherited from the scheme [5] the authors were trying to fix.

In the following, we construct an extremely simple PHE scheme by taking the core idea of Phoenix, stripping off the PKE operations, and adding a (symmetric-key) encryption mechanism for messages. The only drawback of removing the PKE operations seems to be that we now explicitly require that the communication between the server and the rate-limiter is done through a secure channel, which was implicitly assumed in Phoenix[5].

Along with the simplification and the upgrade, we also let the rate-limiter generate a proof even if the pseudorandom value given by the server, or equivalently the given candidate password, is invalid (which was missing in Phoenix). With these modifications the scheme satisfies the strong soundness definition (which subsumes binding), making the rate-limiter more accountable.

3.3 Description of Construction

Let $\mathbb{G}$ be a finite multiplicative cyclic group of order q with identity element I. Let $\Pi.(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Vf})$ be a non-interactive zero-knowledge proof of knowledge (NIZKPoK) scheme for discrete logarithm representations in $\mathbb{G}$ (*e.g.*, the generalized Schnorr protocol). Let $\mathsf{H}_{\mathcal{S}}, \mathsf{H}_{\mathcal{R}} : \{0,1\}^* \to \mathbb{G}$ be hash functions (to be modeled as random oracles in the security proof). Let the password space and message space to be $\mathcal{P} := \{0,1\}^*$ and $\mathcal{M} := \mathbb{G}$ respectively. Our construction is as follows.

Setup and Key Generation (Figure 7). The setup procedure generates a common reference string crs (which defines $\mathsf{H}_{\mathcal{S}}$ and $\mathsf{H}_{\mathcal{R}}$) and a generator G of the group $\mathbb{G}$. The server and the rate-limiter generate their keys using $\mathsf{KGen}_{\mathcal{S}}$ and $\mathsf{KGen}_{\mathcal{R}}$ respectively and individually. The server secret key consists of an integer $y \in \mathbb{Z}_q$. The rate-limiter secret key is $x \in \mathbb{Z}_q$ and the public key is $X = G^x$.

Encryption (Figure 8). When a new end user registers for a new account with the server, the server engages in an encryption protocol with the rate-limiter. The server inputs its secret key, the password pw, and the message M, while the rate-limiter inputs its secret key. (As mentioned in the discussion of the definitions, the input label ℓ is always an empty string in real-world usage.)

The protocol is as follows. In the usual case where ℓ is empty, the server and the rate-limiter sample random nonces $n_{\mathcal{S}}$ and $n_{\mathcal{R}}$ respectively. These nonces serve as session identifiers or pseudonyms of the registering end user. Otherwise, if ℓ is non-empty, the parties simply parse it as the tuple $(n_{\mathcal{R}}, n_{\mathcal{S}})$.

[5] In the hiding experiment, the communication transcript of the enrollment protocol for creating the challenge record is not given to the adversary. Their security proof indeed makes use of this fact.

$\mathsf{Setup}\ (1^\lambda)$	$\mathsf{KGen}_{\mathcal{R}}(\mathsf{pp})$
$\mathsf{crs} \leftarrow_{\$} \Pi.\mathsf{Gen}(1^\lambda)$	$x \leftarrow_{\$} \mathbb{Z}_q$
$G \leftarrow_{\$} \mathbb{G}$	$X \leftarrow G^x$
return (crs, G)	$\mathsf{pk}_{\mathcal{R}} \leftarrow X$
	$\mathsf{sk}_{\mathcal{R}} \leftarrow x$
$\mathsf{KGen}_{\mathcal{S}}(\mathsf{pp})$	**return** $(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$
$\mathsf{pk}_{\mathcal{S}} \leftarrow \varepsilon$	
$\mathsf{sk}_{\mathcal{S}} \leftarrow y \leftarrow_{\$} \mathbb{Z}_q$	
return $(\mathsf{pk}_{\mathcal{S}}, \mathsf{sk}_{\mathcal{S}})$	

Figure 7: Setup and Key Generation of PHE

Next, the parties jointly create the ciphertext $(H^x_{\mathcal{R},0}H^y_{\mathcal{S},0}, H^x_{\mathcal{R},1}H^y_{\mathcal{S},1}M^y)$[6], where $H_{\mathcal{S},b} = \mathsf{H}_{\mathcal{S}}(\mathsf{pw}, n_{\mathcal{S}}, b)$ and $H_{\mathcal{R},b} = \mathsf{H}_{\mathcal{R}}(n_{\mathcal{R}}, b)$ for $b \in \{0,1\}$[7]. To do so, the rate-limiter sends the tuple $(H^x_{\mathcal{R},0}, H^x_{\mathcal{R},1})$ along with the rate-limiter nonce $n_{\mathcal{R}}$ to the server. The latter completes the ciphertext by multiplying the tuple with $(H^y_{\mathcal{S},0}, H^y_{\mathcal{S},1}M^y)$ (component-wise). Finally, the server stores the resulting ciphertext as the record T and the nonces $n_{\mathcal{S}}$ and $n_{\mathcal{R}}$ as the label ℓ' for the registering end user.

Decryption (Figure 9). When an end user logs in with a candidate password pw, the server looks up its corresponding record T and label ℓ, and engages in the decryption protocol with the rate-limiter. The server inputs its secret key $\mathsf{sk}_{\mathcal{S}}$, the label ℓ, the record T, and the password pw. The rate-limiter inputs its secret key $\mathsf{sk}_{\mathcal{R}}$ and the label ℓ. In a slightly different formulation, we can let the server send ℓ with the first message to the rate-limiter.

Recall that the record T is in the form $(T_0, T_1) = (H^x_{\mathcal{R},0}H^y_{\mathcal{S},0}, H^x_{\mathcal{R},1}H^y_{\mathcal{S},1}M^y)$. To begin, the server computes C_0 as $T_0/\mathsf{H}_{\mathcal{S}}(\mathsf{pw}, n_{\mathcal{S}}, 0)^y$, which is equal to $H^x_{\mathcal{R},0}$ if the password pw is correct. It sends C_0 to the rate-limiter, who checks if C_0 is indeed equal to $H^x_{\mathcal{R},0}$. If so it sends $H^x_{\mathcal{R},1}$, and a proof that the computation is done faithfully, back to the server. The latter then verifies the proof, recovers M as $(T_1 H^{-x}_{\mathcal{R},1} H^{-y}_{\mathcal{S},1})^{1/y}$, and outputs the flag $f = 1$ and the message M. Otherwise, the rate-limiter proves that C_0 and $H^x_{\mathcal{R},0}$ are not equal. The server verifies the proof and outputs the flag $f = 0$ (and $M = \varepsilon$).

Key Rotation and Update (Figure 10). When either one of the server and the rate-limiter is compromised, or due to regular routine, they may engage in a key rotation protocol to rotate their (public and) secret keys such that they are distributed identically as freshly generated keys. Then, the server *locally* runs the update algorithm on each record so that it is valid with respect to the new keys. Note that the update is done *without* knowing the passwords and messages corresponding to the records.

In the key rotation protocol, the rate-limiter generates a tuple of random integers (α, β) and sends it to the server[8]. The latter updates its secret key to $y' = \alpha y$. Similarly, the rate-limiter updates its secret key to $x' = \alpha x + \beta$. It also publishes its new public key $X' = G^{x'}$.

To update each encryption record T without knowing the encrypted message and the corresponding password, the server runs the update algorithm on each record T with its label $\ell = (n_{\mathcal{R}}, n_{\mathcal{S}})$. Recall that a record T is in the form $(T_0, T_1) = (H^x_{\mathcal{R},0}H^y_{\mathcal{S},0}, H^x_{\mathcal{R},1}H^y_{\mathcal{S},1}M^y)$. The algorithm simply computes $T' = (T'_0, T'_1)$ as

$$\begin{aligned}(T'_0,\ T'_1) &= (T_0^{\alpha} H^{\beta}_{\mathcal{R},0},\ T_1^{\alpha} H^{\beta}_{\mathcal{R},1}) \\ &= (H^{\alpha x+\beta}_{\mathcal{R},0} H^{\alpha y}_{\mathcal{S},0},\ H^{\alpha x+\beta}_{\mathcal{R},1} H^{\alpha y}_{\mathcal{S},1} M^{\alpha y}) \\ &= (H^{x'}_{\mathcal{R},0} H^{y'}_{\mathcal{S},0},\ H^{x'}_{\mathcal{R},1} H^{y'}_{\mathcal{S},1} M^{y'}).\end{aligned}$$

Correctness. The correctness of the scheme follows immediately from the completeness of the NIZKPoK scheme, and is subsumed by the soundness property.

3.4 Security Analysis

We state our formal results with proof sketches. Full proofs are postponed to Appendix A.

Theorem 1 (Partial Obliviousness) *Assume that DDH is hard in $\mathbb{G}$. Then, in the random oracle model, our construction achieves partial obliviousness.*

Proof 1 (Proof sketch) *The proof is based on the observation that the adversary can only obtain (pseudorandom) hashes of (pw^*_b, M^*_b) but not $(\mathsf{pw}^*_{1-b}, M^*_{1-b})$, since (essentially) the only way to obtain the latter is by querying the decryption oracle on (ℓ, pw) where $\ell = (\ell^*_{\mathcal{S}}, \cdot)$ and $\mathsf{pw} = \mathsf{pw}^*_{1-b}$, which is refused by the oracle.*

Theorem 2 (Message Hiding) *If Π is zero-knowledge and DDH is hard in $\mathbb{G}$, then our construction achieves message hiding in the random oracle model.*

Proof 2 (Proof sketch) *The core of the proof relies on the fact that the adversary must submit a pseudorandom value in order to gain any useful information about the challenge message M^*_b. However, since the pseudorandom value is masked by the (pseudorandom) hash of the challenge password pw^*, the only way to obtain the value is through guessing pw^*.*

[6] The purpose of encrypting M^y instead of M is to "absorb" the effect of key-rotation to y, so that M does not change after key-rotation.

[7] The input b essentially splits $\mathsf{H}_{\mathcal{R}}$ (and $\mathsf{H}_{\mathcal{S}}$) into two independent hash functions, thus saving the need to have a two-integer secret key.

[8] It is also possible to have the server and the rate-limiter jointly generate these values, so that both parties are convinced that the values are truly random. Yet, this is not necessary for proving security in our model.

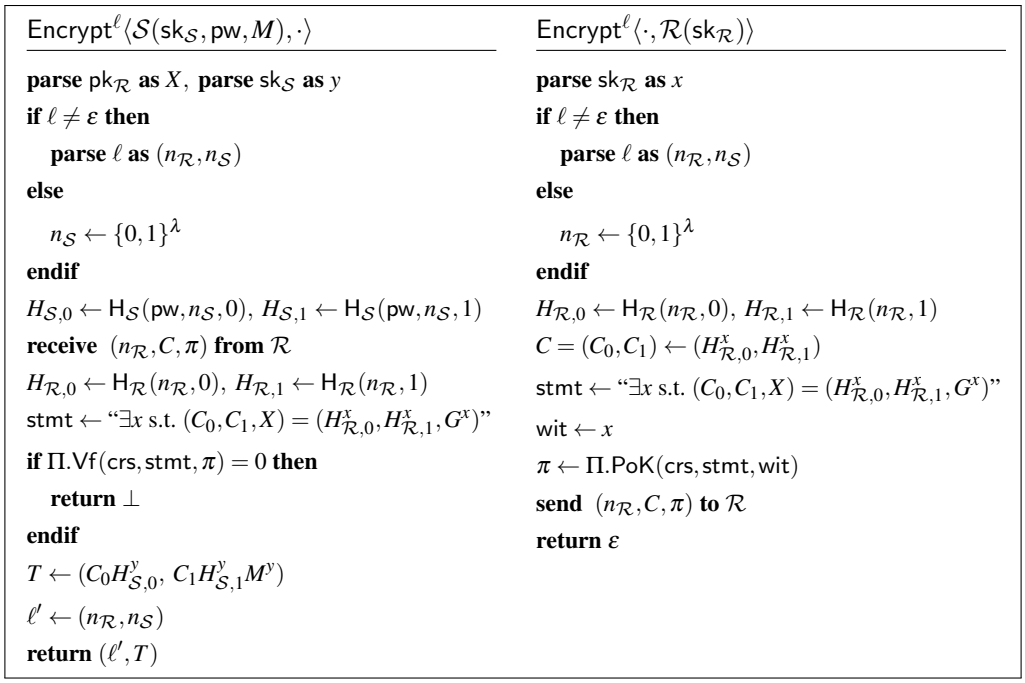

Figure 8: Encryption Protocol of PHE

Theorem 3 (Strong Soundness) *If* Π *is sound and has the proof of knowledge property, then our construction is strongly sound.*

Proof 3 (Proof sketch) *The proof follows almost immediately from the soundness and the proof of knowledge property of* Π*: An adversary against (strong) soundness must convince an honest server to either draw an incorrect conclusion about the validity of a record or a candidate password, or recover a different message which is not encrypted in the record. This means that the adversary is able to produce proofs of contradicting statements, one of which must be false. We can thus use such an adversary as a black-box to break the soundness of* Π*.*

Theorem 4 (Forward security) *Our construction is perfectly forward secure.*

Proof 4 (Proof sketch) *The truth of the claim follows from the fact that, for any tuples* (x,y) *and* (x',y') *in* $\mathbb{Z}_q^2$*, there exists a unique mapping* $(x',y') = (\alpha x + \beta, \alpha y)$ *defined by* (α, β) *in* $\mathbb{Z}_q^2$ *which maps* (x,y) *to* (x',y')*.*

4 Evaluation and Deployment

We report the performance evaluation of our prototype implementation and discuss the possibility of practical deployment. We use `SHA256` for the hash functions and `NIST P-256` for the group $\mathbb{G}$. For the zero-knowledge proofs we use sigma protocols [16] based on Fiat-Shamir [17] for equality and inequality of discrete logarithm representations.

4.1 Evaluation

For a detailed evaluation, we implemented our scheme using the Charm [18] crypto prototyping library and the Falcon Web Framework. Data is passed through GET parameters to the crypto service and the results are communicated back in JSON. We used a dedicated virtual machine on an off-the-shelves server and assigned one up to eight cores to the virtual machine. The host system for the local setup is running nginx and uwsgi on a 10 Core Intel Xeon E5-2640 CPU.

For all studies, we assume an https connection with keep-alive. We consider this realistic for busy sites where a dedicated connection is kept open between the PHE service and the user-facing webserver.

To estimate the resources needed, we evaluated the throughput of the PHE rate-limiter. The measurements are obtained using the Apache benchmark tool. As shown in Figure 11, the PHE crypto service perfectly scales to more cores and can handle more than 525 encryption and (successful) decryption (*i.e.*, registration and login) requests per second (per core). As shown in Table 1, this is a significant improvement even compared to PHOENIX which has no encryption functionality: PHOENIX can process 371 validation requests using similar hardware [3]. Enrollment in PHOENIX is significantly cheaper (1500 requests per second [3]) than en-

$\mathsf{Decrypt}^{\ell}\langle \mathcal{S}(\mathsf{sk}_{\mathcal{S}}, \mathsf{pw}, T), \cdot \rangle$

parse $\mathsf{pk}_{\mathcal{R}}$ **as** X
parse $\mathsf{sk}_{\mathcal{S}}$ **as** y
parse T **as** (T_0, T_1)
parse ℓ **as** $(n_{\mathcal{R}}, n_{\mathcal{S}})$
$H_{\mathcal{R},0} \leftarrow \mathsf{H}_{\mathcal{R}}(n_{\mathcal{R}}, 0)$, $H_{\mathcal{R},1} \leftarrow \mathsf{H}_{\mathcal{R}}(n_{\mathcal{R}}, 1)$
$H_{\mathcal{S},0} \leftarrow \mathsf{H}_{\mathcal{S}}(\mathsf{pw}, n_{\mathcal{S}}, 0)$, $H_{\mathcal{S},1} \leftarrow \mathsf{H}_{\mathcal{S}}(\mathsf{pw}, n_{\mathcal{S}}, 1)$
$C_0 \leftarrow T_0 H_{\mathcal{S},0}^{-y}$
send C_0 **to** $\mathcal{R}$
receive (f, C_1, π) **from** $\mathcal{R}$
if $f = 1$ **then**
  $\mathsf{stmt} \leftarrow$ "$\exists x$ s.t. $(C_0, C_1, X) = (H_{\mathcal{R},0}^{x}, H_{\mathcal{R},1}^{x}, G^{x})$"
  $M \leftarrow (T_1 C_1^{-1} H_{\mathcal{S},1}^{-y})^{1/y}$
elseif $f = 0 \wedge C_1 \neq I$ **then**
  $\mathsf{stmt} \leftarrow$ "$\exists(\alpha, \beta)$ s.t. $(C_1, I) = (C_0^{\alpha} H_{\mathcal{R},0}^{\beta}, X^{\alpha} G^{\beta})$"
  $M \leftarrow \varepsilon$
endif
if $\Pi.\mathsf{Vf}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1$ **then**
  return (f, M)
endif
return $(\perp, \varepsilon)$

$\mathsf{Decrypt}^{\ell}\langle \cdot, \mathcal{R}(\mathsf{sk}_{\mathcal{R}}) \rangle$

parse $\mathsf{sk}_{\mathcal{R}}$ **as** x
parse ℓ **as** $(n_{\mathcal{R}}, n_{\mathcal{S}})$
receive C_0 **from** $\mathcal{S}$
$H_{\mathcal{R},0} \leftarrow \mathsf{H}_{\mathcal{R}}(n_{\mathcal{R}}, 0)$, $H_{\mathcal{R},1} \leftarrow \mathsf{H}_{\mathcal{R}}(n_{\mathcal{R}}, 1)$
if $C_0 = H_{\mathcal{R},0}^{x}$ **then**
  $f \leftarrow 1$, $C_1 \leftarrow H_{\mathcal{R},1}^{x}$
  $\mathsf{stmt} \leftarrow$ "$\exists x$ s.t. $(C_0, C_1, X) = (H_{\mathcal{R},0}^{x}, H_{\mathcal{R},1}^{x}, G^{x})$"
  $\mathsf{wit} \leftarrow x$
else
  $f \leftarrow 0$, $r \leftarrow_{\$} \mathbb{Z}_q$, $C_1 \leftarrow C_0^{r} H_{\mathcal{R},0}^{-rx}$
  $\mathsf{stmt} \leftarrow$ "$\exists(\alpha, \beta)$ s.t. $(C_1, I) = (C_0^{\alpha} H_{\mathcal{R},0}^{\beta}, X^{\alpha} G^{\beta})$"
  $\mathsf{wit} \leftarrow (\alpha, \beta) = (r, -rx)$
endif
$\pi \leftarrow \Pi.\mathsf{PoK}(\mathsf{crs}, \mathsf{stmt}, \mathsf{wit})$
send (f, C_1, π) **to** $\mathcal{S}$
return ε

Figure 9: Decryption Protocol of PHE

	HTTPS keep-alive
static page	$> 10{,}000$
parameter	2,607.16
PYTHIA eval	128.50
Schneider *et al.* enroll	380.37
Schneider *et al.* validate	221.75
PHOENIX enroll	1,557.81
PHOENIX validate	371.34
PHE encrypt	525.04
PHE decrypt	524.21

Table 1: Rate-Limiter Requests per Second

cryption in our scheme, as the former does not involve any zero-knowledge proofs. PYTHIA is even slower due to the pairing-based construction and achieves 129 enrollment or validation requests per second [3].

Finally, we measure the throughput of the server. Since the server needs to perform twice the amount of exponentiations than the rate-limiter does, it is expected that the throughput of the server is roughly half that of the rate-limiter. This expectation is indeed confirmed by the evaluation Figure 12, in which the server is utilizing the same set of machines as were used for the rate-limiter-side evaluation. Specifically, the server can process about 250 requests per core per second. Although no measurement of the server throughput is available for Phoenix [3], we expect our scheme comes on top since fewer exponentiations (*e.g.*, encryption and rerandomization in Phoenix) are required. On the other hand, since the server in Pythia does nothing but equality checks, its computation cost should be negligible.

Considering current recommendations for best practice [19] on password hashing we note that algorithms like scrypt or Argon2 [20] are usually configured to limit login throughput to tens of requests per second which is significantly slower than the overhead introduced by PHE. It might be advisable to instantiate $H_{\mathcal{S}}$ with such a state-of-the-art hashing function for maximum protection. When doing so the overhead of PHE becomes tiny.

4.2 Scalability

Regarding the scalability of PHE, we make two remarks. First, note that the state kept by the rate-limiter for each server is small: It consists of one counter per end user

$\mathsf{Rotate}\langle\mathcal{S}(\mathsf{sk}_\mathcal{S}),\mathcal{R}(\mathsf{sk}_\mathcal{R})\rangle$			$\mathsf{Update}^\ell(\tau,T)$
Client $\mathcal{S}$		**Server** $\mathcal{R}$	**parse** $\mathsf{pk}_\mathcal{R}$ **as** X
parse $\mathsf{sk}_\mathcal{S}$ **as** y		**parse** $\mathsf{sk}_\mathcal{R}$ **as** x	**parse** ℓ **as** $(n_\mathcal{R}, n_\mathcal{S})$
		$\alpha,\beta \leftarrow_\$ \mathbb{Z}_q$	**parse** τ **as** (α,β)
	$\xleftarrow{\quad(\alpha,\beta)\quad}$		**parse** T **as** (T_0,T_1)
			$H_{\mathcal{R},0} \leftarrow \mathsf{H}_\mathcal{R}(n_\mathcal{R},0)$
$y' \leftarrow \alpha y$		$x' \leftarrow \alpha x+\beta$	$H_{\mathcal{R},1} \leftarrow \mathsf{H}_\mathcal{R}(n_\mathcal{R},1)$
$\tau \leftarrow (\alpha,\beta)$			$T_0' \leftarrow T_0^\alpha H_{\mathcal{R},0}^\beta$
$\mathsf{pk}'_\mathcal{S} \leftarrow \varepsilon$		$\mathsf{pk}'_\mathcal{R} \leftarrow G^{x'}$	$T_1' \leftarrow T_1^\alpha H_{\mathcal{R},1}^\beta$
$\mathsf{sk}'_\mathcal{S} \leftarrow y'$		$\mathsf{sk}'_\mathcal{R} \leftarrow x'$	**return** $T' \leftarrow (T_0',T_1')$
return $(\mathsf{pk}'_\mathcal{S},\mathsf{sk}'_\mathcal{S},\tau)$		**return** $(\mathsf{pk}'_\mathcal{R},\mathsf{sk}'_\mathcal{R})$	

Figure 10: Key-Rotation Protocol of PHE

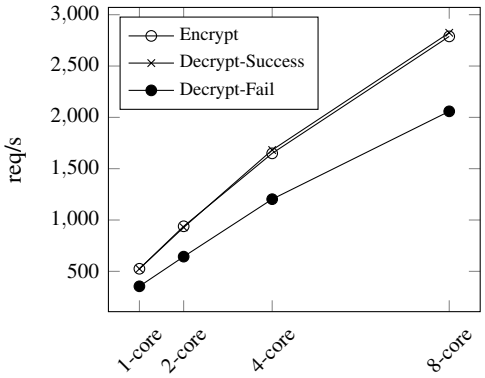

Figure 11: Rate-Limiter throughput in req/s

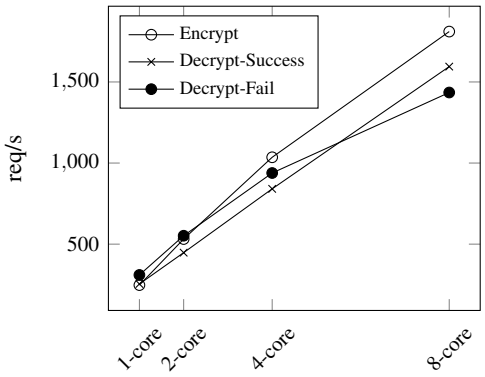

Figure 12: Server throughput in req/s

of the server, solely for rate-limiting purposes. Second, instances of the encryption, decryption, and key rotation protocols (for the same or different servers) are independent. Thus, it is expected that the throughput of the rate-limiter scales linearly with the number of cores, except for the inevitable overhead for threading.

4.3 Possibility of Deployment

We envision a practical deployment of the system due to a mutual benefit of all parties – end users, online service providers, and crypto service providers.

End Users. As the end users are registered for the services provided by the online service providers, we assume that the latter are trusted to a certain degree. Although a new party, namely the rate-limiter, is introduced for the transition from an existing, say access-control-based, data security solution to the more secure PHE solution, the end users need not trust any additional parties due to the obliviousness property against rate-limiters. In fact, the transition to PHE even reduces trust required to the online service providers, since the latter can no longer decrypt user data by themselves.

Online Service Providers. By providing a better security solution to the end users, an online service provider can improve its image which potentially popularizes its services. The risk of financial losses due to data leakage is also reduced, since attackers would now need to fully compromise both the online service provider and the rate-limiter to decrypt user data. This is particularly important for small companies whose developers are not specialized in security. Assuming that the rate-limiters are developed and maintained by security experts, it is a reasonable assumption that these rate-limiters are much harder to attack.

Crypto Service Providers. Crypto service providers have financial incentives to run and maintain rate-limiters, assuming online service providers and end users are willing to invest in better security. The PHE solu-

tion also introduces a better division of labor: Security experts can focus on developing and maintaining rate-limiters which are specialized in security, while online service providers can focus on providing (non-security) services they used to provide.

4.4 Conversion of Existing Systems

An existing system can be converted gradually in at least two ways. As an end user logs in, the server can retrieve the record from the existing system (*e.g.*, salted hash), and create a new record encrypting a random message M using PHE.

To convert the system in a single batch conversion step, assuming the existing system stores passwords in the form of salted hashes $(n_S, \mathsf{H}(n_S, \mathsf{pw}))$, the server samples a random message M, further hashes each record to compute $(n_S, \mathsf{H}^y_{S,0}\mathsf{H}^y_{S,1}) = (n_S, \mathsf{H}(\mathsf{H}(n_S, \mathsf{pw}), 0)^y, \mathsf{H}(\mathsf{H}(n_S, \mathsf{pw}), 1)^y M^y)$ (modeling H as a random oracle and interpreting its output as a group element), and communicate with the rate-limiter to complete the PHE record.

Either way, the random message M is used as a symmetric key (*e.g.*, for AES) to encrypt the existing (plaintext) profile of the end user, and is discarded after encryption. Note that for both approaches, the entire transformation happens at the back-end and does not require special actions from the end user.

5 Conclusion

We have proposed and constructed password-hardened encryption (PHE) services, an extension to password-hardening (PH) services, which not only protects passwords but also user data stored by an online service provider, even if the latter is fully compromised. This is achieved with the aid of an external yet minimally trusted rate-limiter. PHE inherits all useful properties of PH, namely obliviousness, hiding and forward security, and features a stronger soundness property which makes the rate-limiter more accountable. Forward security, or the ability to rotate secret keys, is particularly important in the data security context and is explicitly required by standards such as the PCI DSS [1].

Our construction is obtained by taking the core idea behind a recent PH scheme Phoenix [3], greatly simplifying it, and augmenting it with encryption functionality. The result is an extremely simple and efficient PHE scheme, which can be readily deployed in existing online services without affecting the end users at all or changing the database infrastructure significantly. The scheme incurs an even milder overhead than existing PH schemes, and scales well to a large number of end users and servers.

This work opens up a number of research directions. First, it would be interesting to explore cryptographic techniques to achieve rate limiting while preserving end user anonymity and / or do so in a distributed manner with more than one rate-limiter. The second is to consider a stronger attacker model, in which the attacker can partly observe the messages exchanged between the end user and the server in the decryption phase. In this setting, it is inevitable for the end user to also perform cryptographic operations, which in turns allows stronger security guarantees. The third is to revisit other cryptographic primitives in the password-hardened paradigm. Given the seamless nature of such paradigm (in the view of the end users), it is more likely for the cryptographic primitives to be deployed. Finally, new constructions, perhaps based on other (*e.g.*, lattice-based) complexity assumptions or without using the random oracle, and more efficient instantiations are always welcome.

References

[1] P. S. S. Council, "Requirements and security assessment procedures." PCI DSS v3.2, 2016.

[2] A. Everspaugh, R. Chaterjee, S. Scott, A. Juels, and T. Ristenpart, "The pythia prf service," in *24th USENIX Security Symposium (USENIX Security 15)*, (Washington, D.C.), pp. 547–562, USENIX Association, 2015.

[3] R. W. F. Lai, C. Egger, D. Schröder, and S. S. M. Chow, "Phoenix: Rebirth of a cryptographic password-hardening service," in *26th USENIX Security Symposium (USENIX Security 17)*, (Vancouver, BC), pp. 899–916, USENIX Association, 2017.

[4] R. Cramer and V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack," in *CRYPTO'98* (H. Krawczyk, ed.), vol. 1462 of *LNCS*, (Santa Barbara, CA, USA), pp. 13–25, Springer, Heidelberg, Germany, Aug. 23–27, 1998.

[5] J. Schneider, N. Fleischhacker, D. Schröder, and M. Backes, "Efficient cryptographic password hardening services from partially oblivious commitments," in *ACM CCS 16* (E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, eds.), (Vienna, Austria), pp. 1192–1203, ACM Press, Oct. 24–28, 2016.

[6] S. Jarecki, A. Kiayias, and H. Krawczyk, "Round-optimal password-protected secret sharing and T-PAKE in the password-only model," in *ASIACRYPT 2014, Part II* (P. Sarkar and T. Iwata,

eds.), vol. 8874 of *LNCS*, (Kaoshiung, Taiwan, R.O.C.), pp. 233–253, Springer, Heidelberg, Germany, Dec. 7–11, 2014.

[7] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu, "TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF," in *ACNS 17* (D. Gollmann, A. Miyaji, and H. Kikuchi, eds.), vol. 10355 of *LNCS*, (Kanazawa, Japan), pp. 39–58, Springer, Heidelberg, Germany, July 10–12, 2017.

[8] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *42nd FOCS*, (Las Vegas, NV, USA), pp. 136–145, IEEE Computer Society Press, Oct. 14–17, 2001.

[9] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu, "Password-protected secret sharing," in *ACM CCS 11* (Y. Chen, G. Danezis, and V. Shmatikov, eds.), (Chicago, Illinois, USA), pp. 433–444, ACM Press, Oct. 17–21, 2011.

[10] J. Camenisch, R. R. Enderlein, and G. Neven, "Two-server password-authenticated secret sharing UC-secure against transient corruptions," in *PKC 2015* (J. Katz, ed.), vol. 9020 of *LNCS*, (Gaithersburg, MD, USA), pp. 283–307, Springer, Heidelberg, Germany, Mar. 30 – Apr. 1, 2015.

[11] J. Camenisch, A. Lehmann, and G. Neven, "Optimal distributed password verification," in *ACM CCS 15* (I. Ray, N. Li, and C. Kruegel:, eds.), (Denver, CO, USA), pp. 182–194, ACM Press, Oct. 12–16, 2015.

[12] X. Boyen, "Hidden credential retrieval from a reusable password," in *ASIACCS 09* (W. Li, W. Susilo, U. K. Tupakula, R. Safavi-Naini, and V. Varadharajan, eds.), (Sydney, Australia), pp. 228–238, ACM Press, Mar. 10–12, 2009.

[13] J. Kelsey, B. Schneier, C. Hall, and D. Wagner, "Secure applications of low-entropy keys," in *ISW'97* (E. Okamoto, G. I. Davida, and M. Mambo, eds.), vol. 1396 of *LNCS*, (Tatsunokuchi, Japan), pp. 121–134, Springer, Heidelberg, Germany, Sept. 17–19, 1998.

[14] B. Kaliski, *RFC 2298: PKCS #5: Password-Based Cryptography Specification Version 2.0*. Internet Activities Board, Sept. 2000.

[15] K. M. Moriarty, B. Kaliski, and A. Rusch, *RFC 8018: PKCS #5: Password-Based Cryptography Specification Version 2.1*. Internet Activities Board, Jan. 2017.

[16] J. Camenisch and V. Shoup, "Practical verifiable encryption and decryption of discrete logarithms," in *CRYPTO 2003* (D. Boneh, ed.), vol. 2729 of *LNCS*, (Santa Barbara, CA, USA), pp. 126–144, Springer, Heidelberg, Germany, Aug. 17–21, 2003.

[17] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO'86* (A. M. Odlyzko, ed.), vol. 263 of *LNCS*, (Santa Barbara, CA, USA), pp. 186–194, Springer, Heidelberg, Germany, Aug. 1987.

[18] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013.

[19] J. Steven and J. Manico, "Owasp password storage cheat sheet." https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet, 2016.

[20] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: New generation of memory-hard functions for password hashing and other applications," in *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pp. 292–302, IEEE, 2016.

[21] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," in *38th FOCS*, (Miami Beach, Florida), pp. 458–467, IEEE Computer Society Press, Oct. 19–22, 1997.

A Formal Security Proofs

Partial Obliviousness. If the DDH assumption holds in $\mathbb{G}$, and $\mathsf{H}_{\mathcal{S}}$ is modeled as a random oracle, then PHE is partially oblivious. We prove by defining a sequence of hybrid experiments for $b \in \{0,1\}$, each differs slightly from the previous:

$\mathsf{Exp}_{b,0}$: is identical to $\mathsf{Obl}^{b}_{\mathsf{PHE},\mathcal{A}}$.

$\mathsf{Exp}_{b,1}$: The challenger simulates the random oracle $\mathsf{H}_{\mathcal{S}}$ as follows. If $\mathcal{A}$ queries $\mathsf{H}_{\mathcal{S}}$ directly on an input X, the challenger samples a random integer $a \leftarrow_{\$} \mathbb{Z}_q$ and programs $\mathsf{H}_{\mathcal{S}}(m) := G^a$. However, if $\mathsf{H}_{\mathcal{S}}$ is invoked by the challenger when executing the encryption protocol on the password pw and the message M (and the empty label ε), it samples $n_{\mathcal{S}} \leftarrow_{\$} \{0,1\}^{\lambda}$ and programs $\mathsf{H}_{\mathcal{S}}$ such that $\mathsf{H}_{\mathcal{S}}(\mathsf{pw}, n_{\mathcal{S}}, 0) = G^{a_0}$ and $\mathsf{H}_{\mathcal{S}}(\mathsf{pw}, n_{\mathcal{S}}, 1)M = G^{a_1}$ for random integers $a_0, a_1 \leftarrow_{\$} \mathbb{Z}_q$, assuming $\mathsf{H}_{\mathcal{S}}$ has not been programmed on $(\mathsf{pw}, n_{\mathcal{S}}, 0)$ and $(\mathsf{pw}, n_{\mathcal{S}}, 1)$. The latter assumption holds except with negligible probability as

$n_{\mathcal{S}}$ is uniformly random. If the above assumption holds, this experiment is functionally equivalent to $\mathsf{Exp}_{b,0}$.

$\mathsf{Exp}_{b,2}$: The challenger replaces the values $\mathsf{H}_{\mathcal{S}}(\mathsf{pw}, n_{\mathcal{S}}, 0)^y$ and $(\mathsf{H}_{\mathcal{S}}(\mathsf{pw}, n_{\mathcal{S}}, 1)M)^y$ by random values. This experiment is computationally indistinguishable to $\mathsf{Exp}_{b,1}$ by the DDH assumption [21].

In the experiment $\mathsf{Exp}_{b,2}$, the only information about (pw_b^*, M_b^*) available to $\mathcal{A}$ are the uniformly random values $(\mathsf{H}_{\mathcal{S}}(\mathsf{pw}_b^*, n_{\mathcal{S}}^*, 0)^y$ and $(\mathsf{H}_{\mathcal{S}}(\mathsf{pw}_b^*, n_{\mathcal{S}}^*, 1)M_b^*)^y$, where $\ell^* = (n_{\mathcal{R}}^*, n_{\mathcal{S}}^*)$, since the decryption oracle refuses to decrypt ciphertexts with the labels $\ell = (\ell_{\mathcal{S}}^*, \cdot)$ and passwords pw_0^* and pw_1^*. The experiments $\mathsf{Exp}_{0,2}$ and $\mathsf{Exp}_{1,2}$ are thus identical in the view of $\mathcal{A}$.

Message Hiding. If the DDH assumption holds in $\mathbb{G}$, Π is zero knowledge, and $\mathsf{H}_{\mathcal{S}}$ and $\mathsf{H}_{\mathcal{R}}$ are modeled as random oracles, then PHE is message hiding. We prove formally by defining a sequence of hybrid experiments, each differs slightly from the previous:

$\mathsf{Exp}_{b,0}$: is identical to $\mathsf{Hid}^b_{\mathsf{PHE},\mathcal{A}}$.

$\mathsf{Exp}_{b,1}$: The proofs are now simulated using the simulator guaranteed by the zero-knowledge property of Π. This experiment is computationally indistinguishable from $\mathsf{Exp}_{b,0}$ by the zero-knowledge property of Π.

$\mathsf{Exp}_{b,2}$: The challenger simulates the random oracles $\mathsf{H}_{\mathcal{S}}$ and $\mathsf{H}_{\mathcal{R}}$ as follows. When $\mathsf{H}_{\mathcal{S}}$ (resp. $\mathsf{H}_{\mathcal{R}}$) is queried on some input m, the challenger samples $a \leftarrow_{\$} \mathbb{Z}_q$ and programs $\mathsf{H}_{\mathcal{S}}(m) := G^a$ (resp. $\mathsf{H}_{\mathcal{R}}(m) := G^a$). This experiment is functionally equivalent to $\mathsf{Exp}_{b,1}$.

$\mathsf{Exp}_{b,3}$: The challenger replaces the function $\mathsf{H}_{\mathcal{R}}(m)^x$ by a random function. The indistinguishability of $\mathsf{Exp}_{b,3}$ to $\mathsf{Exp}_{b,2}$ follows from the DDH assumption in the random oracle model [21].

$\mathsf{Exp}_{b,4}$: When $\mathcal{A}$ queries the decryption oracle with the label $\ell = (\cdot, \ell_{\mathcal{R}}^*)$, the challenger always rejects, *i.e.*, it outputs a simulated proof that the value C_0 is invalid. In the following, we show that a distinguisher which distinguishes this experiment from $\mathsf{Exp}_{b,3}$ cannot succeed with a probability higher than that of guessing the password pw^*, except with negligible probability. Then, the proof is done since $\mathsf{Exp}_{0,4}$ and $\mathsf{Exp}_{1,4}$ are functionally identical.

After the modification made in $\mathsf{Exp}_{b,3}$, note that the challenger essentially acts as a conditional decryption oracle which, on input $(n_{\mathcal{R}}, C_0)$, checks if the ciphertext is well-formed, *i.e.*, whether $C_0 = \mathsf{H}_{\mathcal{R}}(n_{\mathcal{R}}, 0)^x$ (which is programmed to a random value), and if so outputs $C_1 = \mathsf{H}_{\mathcal{R}}(n_{\mathcal{R}}, 1)^x$ with a simulated proof of correctness. Otherwise, it outputs a simulated proof of the statement that C_0 and $C_0 = \mathsf{H}_{\mathcal{R}}(n_{\mathcal{R}}, 0)^x$ are not equal.

Recall that the challenge record is computed as

$$\begin{bmatrix} T_0^* \\ T_1^* \end{bmatrix} = \begin{bmatrix} \mathsf{H}_{\mathcal{R}}(n_{\mathcal{R}}^*, 0)^x \mathsf{H}_{\mathcal{S}}(\mathsf{pw}^*, n_{\mathcal{S}}^*, 0)^y \\ \mathsf{H}_{\mathcal{R}}(n_{\mathcal{R}}^*, 1)^x \mathsf{H}_{\mathcal{S}}(\mathsf{pw}^*, n_{\mathcal{S}}^*, 1)^y (M_b^*)^y \end{bmatrix}$$

where $\mathsf{H}_{\mathcal{R}}(n_{\mathcal{R}}^*, 0)^x$ and $\mathsf{H}_{\mathcal{R}}(n_{\mathcal{R}}^*, 1)^x$ are all uniformly random values in the view of $\mathcal{A}$. Thus, in the experiment $\mathsf{Exp}_{b,3}$, the only information of (pw^*, M_b^*) available to $\mathcal{A}$, apart from the challenge record, is obtained via interacting with the decryption oracle, which always rejects unless $\mathcal{A}$ guesses the uniformly random value $\mathsf{H}_{\mathcal{R}}(n_{\mathcal{R}}^*, 0)^x$ correctly, which equivalently means guessing the value $\mathsf{H}_{\mathcal{S}}(\mathsf{pw}^*, n_{\mathcal{S}}^*, 0)$ correctly. Since $\mathsf{H}_{\mathcal{S}}$ is a random oracle, it holds except with negligible probability that $\mathcal{A}$ has queried $\mathsf{H}_{\mathcal{S}}$ at the point $(\mathsf{pw}^*, n_{\mathcal{S}}^*, 0)$. Thus, the challenger can extract pw^*.

Soundness. If Π is sound and has the proof of knowledge property, then PHE is strongly sound.

To prove such claim, we observe that if there exists an adversary $\mathcal{A}$ which causes either of the soundness experiments to output 1, then the challenger can extract two proofs for two contracting statements respectively, which breaks the soundness of Π. In the following, we assume the server acted by $\mathcal{A}$ never aborts (otherwise the experiment outputs 0).

Suppose there exists $\mathcal{A}$ such that the experiment $\mathsf{Soundness}_{\mathsf{PHE},\mathcal{A}}$ outputs 1 with non-negligible probability. There are two cases. First, $(\mathsf{pw} = \mathsf{pw}' \wedge (f \neq 1 \vee M \neq M'))$. Second, $(\mathsf{pw} \neq \mathsf{pw}' \wedge f \neq 0)$.

In either case, the challenger receives upon conclusion of the encryption protocol a proof for the statement "$\exists x$ s.t. $(C_0, C_1, X) = (H_{\mathcal{R},0}^x, H_{\mathcal{R},1}^x, G^x)$". Then, in the first case, suppose the first sub-case $f \neq 1$ happens. It means that the challenger receives a proof for the statement "$\exists(\alpha, \beta)$ s.t. $(C_1, I) = (C_0^\alpha H_{\mathcal{R},0}^\beta, X^\alpha G^\beta)$", which equivalently means "$\exists x$ s.t. $C_0 \neq H_{\mathcal{R},0}^x \wedge X = G^x$)". Since the statements are contradictory, either one is false. The challenger can thus be turned into an adversary against the soundness of Π. Similarly, in the second sub-case, $M \neq M'$. This means that the challenger has a proof of "$\exists x$ s.t. $(C_1', X) = (H_{\mathcal{R},1}^x, G^x)$" for some $C_1' \neq C_1$, another contradicting statement.

For the second case, since $\mathsf{pw} \neq \mathsf{pw}'$, the challenger sends C_0' which is not equal to C_0 except with negligible probability to $\mathcal{A}$ in the decryption protocol. The contradicting statement here is then "$\exists x$ s.t. $(C_0', X) = (H_{\mathcal{R},0}^x, G^x)$".

The analysis of the other experiment is similar. We describe it for completeness. Suppose there exists $\mathcal{A}$ such that the experiment $\mathsf{StrongSoundness}_{\mathsf{PHE},\mathcal{A}}$ outputs 1 with non-negligible probability. There are again two cases. First, $((\ell, \mathsf{pw}) = (\ell', \mathsf{pw}') \wedge (f, M) \neq (f', M'))$. Second, $((\ell, \mathsf{pw}) \neq (\ell', \mathsf{pw}') \wedge f = f' = 1)$.

For the first case, since $(\ell, \mathsf{pw}) = (\ell', \mathsf{pw}')$ the same message C_0 is sent from the challenger to $\mathcal{A}$ in the decryption protocols. We then split into two subcases. First, $f = 0$ but $f' = 1$. The contradicting

statements are thus "$\exists x$ s.t. $(C_0, X) = (H^x_{\mathcal{R},0}, G^x)$" and "$\exists x$ s.t. $(C_0, X) \neq (H^x_{\mathcal{R},0}, G^x)$". Second, $f = f' = 1$ but $M \neq M'$. Here, the contradicting statements are "$\exists x$ s.t. $(C_1, X) = (H^x_{\mathcal{R},1}, G^x)$" and "$\exists x$ s.t. $(C'_1, X) = (H^x_{\mathcal{R},1}, G^x)$" for some $C'_1 \neq C_1$.

For the second case, since $(\ell, \mathsf{pw}) \neq (\ell', \mathsf{pw}')$, distinct C_0 and C'_0 are sent instead with high probability. The contradicting statements in this case are "$\exists x$ s.t. $(C_0, X) = (H^x_{\mathcal{R},0}, G^x)$" and "$\exists x$ s.t. $(C'_0, X) = (H^x_{\mathcal{R},0}, G^x)$". This completes the proof.

Forward Security. We show that PHE is perfectly forward secure. To prove such claim, it suffices to show that the secret keys $\mathsf{sk}'_{\mathcal{S}}$ and $\mathsf{sk}'_{\mathcal{R}}$ output from the rotation protocol is identically distributed as fresh secret keys. The public keys and the records are uniquely determined by the secret keys.

For any client and server secret keys x and y, there is a one-to-one correspondence between each fresh key pairs $(x', y') \in \mathbb{Z}_q^2$ and each tuple of randomness $(\alpha, \beta) \in \mathbb{Z}_q^2$ chosen in the rotation protocol, given by

$$\begin{cases} x' &= \alpha x + \beta \\ y' &= \alpha y \end{cases} \equiv \begin{cases} \alpha &= y'/y \\ \beta &= x' - \alpha x \end{cases}.$$

Thus, the distribution of (x', y') which is sampled uniformly from $\mathbb{Z}_q^2$ and that which is computed from a uniformly random tuple (α, β) are identical.