Hybrid Password Hardening Encryption

Zixuan Ding and Ding Wang

***Abstract*—More and more sensitive data is made online, and even when the data is encrypted on the server, adversaries who compromise the server can obtain the decryption key and thus decrypt the data, because the key is generally stored on the same server as the data. Password Hardening (PH) encryption introduces an additional security layer by incorporating an external PH server to restrict unauthorized decryption. However, leading PH encryption schemes (at USENIX SEC'18 and ACM CCS'20) suffer from substantial encryption/decryption inefficiency, making them unsuitable for large-scale data processing. Additionally, these schemes still have privacy shortcomings, as the external PH server can infer user habits by learning authentication results. *For the first time*, we propose a brand-new PH encryption scheme named HPHE and a hash-based puncturable pseudorandom function, which together form a hybrid PH encryption architecture. The architecture is extensible to other PH schemes and avoids key reuse by deriving high-entropy keys to achieve one-data-one-key. Compared with non-hybrid original PH schemes, HPHE achieves at least a 61% improvement in the efficiency of interactive PH encryption/decryption. In one-data-one-key scenarios, HPHE achieves approximately 450 times higher encryption/decryption efficiency than original PHE (USENIX SEC'18) by replacing multi-round interaction with key derivation. Additionally, HPHE achieves irrecoverable secure deletion and access restrictions by puncturing keys. *For the first time*, the novel construction of our HPHE achieves the *Hiding* of password verification results in PH. In addition, we formally define the *Privacy* security attributes in PH encryption and show that HPHE satisfies the strongest security guarantees. This work extends PH encryption to efficient large-scale data processing and more comprehensive privacy protection.**

***Index Terms*—Cryptographic protocols, encryption, passwords, privacy.**

I. Introduction

IN PRACTICE, web service providers store user account information, including sensitive data such as credit card numbers and email addresses. However, frequent data breaches (e.g., the 200 million Twitter leak in January 2023 [16], the 26 billion MOAB leak in January 2024 [31], the 10 billion Rockyou leak in July 2024 [22], and the 2.7 billion global IoT record and WiFi password leak in February 2025 [9]) raise serious concerns about the security of server-side storage.

Received 17 June 2025; revised 27 November 2025; accepted 22 December 2025. Date of publication 31 December 2025; date of current version 18 March 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 62222208 and in part by the Tianjin Natural Science Foundation Project under Grant 25JCJQJC00230. The associate editor coordinating the review of this article and approving it for publication was Dr. Yue Zhang. *(Corresponding author: Ding Wang.)*

The authors are with the College of Cryptology and Cyber Science, Nankai University, Tianjin 300350, China, also with the Key Laboratory of Data and Intelligent System Security (Ministry of Education), Nankai University, Tianjin 300350, China, also with the Tianjin Key Laboratory of Network and Data Security Technology, Nankai University, Tianjin 300350, China, and also with the Beijing Key Laboratory of Post-Quantum Cryptography, Beijing 100083, China (e-mail: dingzixuan@mail.nankai.edu.cn; wangding@nankai.edu.cn).

Digital Object Identifier 10.1109/TIFS.2025.3650025

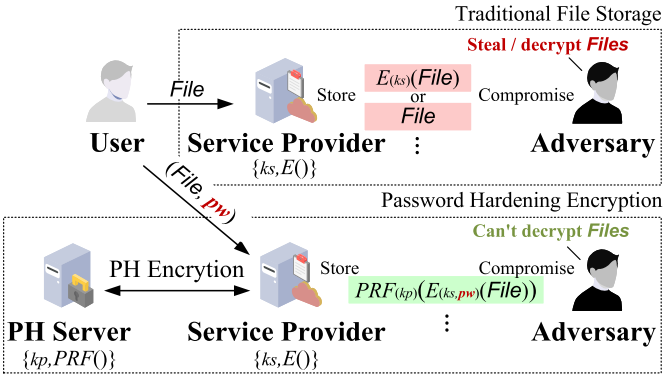


Fig. 1. Password Hardening (PH) encryption architecture diagram. In traditional file storage services, service providers store files in plaintext or encrypted form. An adversary can obtain files after compromising the service provider. PH encryption introduces a PH server, which interacts with the service provider during the enrollment/encryption phase to encrypt files. The encryption involves three-party secrets, and neither collusion between two parties nor data compromise by an adversary can decrypt the files.

Worse still, encryption alone does not prevent internal adversaries from exfiltrating the database and performing offline decryption once the necessary key material is obtained [2], [26], [39]. Therefore, if the service provider retains the full secret key material, adversaries who compromise the service provider may also decrypt it [38].

A practical method for securing user data is to implement additional encryption services as a security layer, where an auxiliary server assists in encryption/decryption without learning the plaintext, an idea coined Password Hardening (PH) service and PH Encryption (PHE). In this work, we refer to the external server as the PH server, which interacts with the service provider to perform password hardening for encryption and decryption. Lai et al. propose the first PHE scheme at USENIX SEC'18 [23]. During the enrollment/encryption phase, the service provider interacts with the PH server to generate a password-bound encryption record. For decryption, the parties interact to recover the plaintext from the record. The architecture of PHE is shown in Figure 1.

Why is PHE needed? Challenges with user-managed encryption: managing multiple encryption keys is burdensome; potential key reuse across multiple files; adversaries compromising the service provider can brute-force ciphertexts encrypted with low-entropy keys; and users bear the computational overhead. Concerns with service providers solely handling file encryption: inability to resist insider theft; and compromised data can be cracked offline. In contrast, PHE uses user passwords as low-entropy secrets and hardens them via the PH server, enabling interaction-based decryption that

mitigates offline guessing and preventing any single compromised entity from decrypting data.

The mandatory interaction requirement in PHE introduces four additional security properties compared to simple encryption/decryption solely performed by the service provider: 1) Neither the service provider nor the PH server can independently verify the correctness of the password or decrypt the ciphertext. An adversary compromising the service provider can only launch online guessing attacks; 2) The PH server monitors the frequency of online login attempts, thereby limiting online guessing attacks; 3) Knowledge of the password, encrypted message, and verification results is not disclosed to any party; 4) The service provider and PH server collaborate on key rotation, mitigating the risk of key or record leakage.

Although PHE strengthens the security of online encryption, its efficiency is constrained by interaction latency and computational overhead. Additionally, due to the protocol's requirements for computational correctness, existing PHE schemes restrict the plaintext length for each encryption/decryption operation. A viable approach is to employ PHE for encrypting separate symmetric keys [23]. However, in practice, encrypting large volumes of data with a limited, repeated key can result in key reuse and pose challenges in ciphertext management. Moreover, current PHE schemes still have shortcomings. For instance, in PHE [23] and APHE [14], the PH server learns the password verification result. The interaction messages in (t, m)-PHE [6] are three times those of other PHE schemes, and (t, m)-PHE assumes that at least t out of m PH servers remain online and responsive.

A. Motivations

1) Privacy: To maximize privacy protection, it is essential to minimize information disclosure to third parties. Therefore, apart from the additional encryption services provided by the PH server, the PH server is not expected to access user-related information. In PHE [23] and APHE [14], the PH server verifies the legitimacy of requests by comparing $T_0 H_{S,0}^{-y} = H_{R,0}^{x}$ based on the stored random number n_R, where $H_{S,0} = H_R(n_R, pw, 0)$, pw is the password, thereby learning the results of identity verification. Additionally, since the PH server limits access frequency based on fixed parameters, it can learn users' login habits and analyze their profiles [42]. Therefore, we aim to strengthen privacy by minimizing the information leaked during PH-server interactions, particularly authentication results and access-pattern information.

2) Efficiency: Efficiency is a critical factor in encryption/decryption processes. In scenarios involving concurrent multiprocesses or large-scale data encryption/decryption, high latency or numerous interaction rounds can adversely affect the deployment. In (t, m)-PHE [6], decryption involves 3 rounds and $6p$ interactions, where p represents the number of PH servers involved. Even when $p = 1$, the overhead of (t, m)-PHE is still much higher than that of other PHE schemes. Additionally, due to the requirement for non-interactive zero-knowledge proofs in PHE [23] and APHE [14], their encryption and decryption latency in applications is 500 to 1,000 times that of symmetric encryption on equivalent hardware and data volume. Thus, we aim to implement PHE with fewer interaction rounds and lower overhead to make the scheme suitable for large-scale encryption and decryption.

3) Ciphertext Security and Management: Ciphertext independence is crucial for the overall security of encryption schemes, as it prevents adversaries from deducing any information about the plaintext, even with access to multiple ciphertexts [42]. Ciphertext manageability (e.g., secure sharing and revocation/deletion) is an additional requirement. Achieving these goals at scale is challenging for PHE, since existing PHE schemes incur substantial latency and interaction overhead for each encryption. The efficiency and interaction limitations of current PHE schemes make simultaneous encryption of multiple data items impractical. One solution is to use PHE to encrypt independent keys, but reusing the same key for multiple items introduces the risk of key reuse, compromising both security and manageability. A more effective approach is to assign different keys to each data item. Practical PHE needs secure key distribution and robust ciphertext management.

B. Contributions

Our main contribution is a hybrid (two-tier hierarchical) PHE architecture and a novel PHE scheme, named HPHE. The two-tier collaborative framework integrates PHE and hash-based Puncturable Pseudorandom Functions (PPRF). In the upper tier, PHE encrypts a high-entropy root key, the key's confidentiality relies on the joint security of the service provider and the PH server, thereby resisting offline brute-force attacks. The lower tier uses PPRF to derive multiple unique subkeys from the root key, which are further applied in symmetric encryption for large-scale data processing.

Our hybrid PHE architecture enhances security and privacy, and we formally define and prove the privacy for PHE. The proposed hash-based PPRF can be applied to any PHE scheme, and compared to the original PHE, the hybrid PHE improves encryption/decryption efficiency by 45 to 457 times. Furthermore, the hybrid PHE enables one-data-one-key, supports secure deletion of ciphertext, and allows access control when sharing. We validated the feasibility and efficiency of the scheme through practical deployment tests, showing that the throughput of the PH server is about 10 times greater than that of the most efficient existing PHE scheme.

In summary, our contributions are three-fold:

- **Hybrid password hardening encryption.** *For the first time*, we propose a hybrid PHE architecture and a brand-new PHE, called Hybrid Password Hardening Encryption (HPHE). HPHE fully hides user secrets from the third-party PH server, eliminating the need for the server to store preset parameters and preventing it from learning password-verification results. Our proposed hybrid encryption framework, integrated with PPRF, extends any PHE into a secure symmetric encryption system, where each data item has a unique key, along with comprehensive ciphertext and key management. We formally prove that the proposed HPHE meets the highest current security standards.
- **Hash-based puncturable pseudorandom functions.** *For the first time*, we construct a hash-based fast PPRF

that can be directly deployed using existing hash functions (e.g., SHA-512 is used to construct a PPRF that generates 256-bit keys, with a cost of approximately 33 *ms* to derive 1,024 keys). The purpose of introducing the PPRF is to integrate it with PHE, enabling the derivation of high-entropy keys from a root key for symmetric encryption/decryption. This solution addresses the original PHE's limitations in encryption/decryption efficiency, data volume, and the inability to manage ciphertext or control access permissions. We provide a security definition for the proposed PPRF and formally prove its security.

- **Privacy.** *For the first time*, we formally define the *privacy* attributes of PHE, focusing on concealing user privacy from external PH servers. We show that HPHE satisfies this notion and additionally hides password-verification results from the PH server. We formally prove HPHE's privacy.
- **Insight.** Original PHE schemes are limited in terms of both data volume and efficiency. Although our proposed PHE significantly improves efficiency by reducing encryption/decryption overhead to approximately 0.34% and 0.42% of the existing optimal PHE scheme [14], the amount of data processed in a single encryption/decryption operation is still constrained by computational limitations (e.g., based on NIST-P256, the encryption/decryption limit is 256 bits). The proposed hybrid PHE architecture is so far the optimal solution available and is applicable to most existing PHE schemes. In the case of large-scale data encryption/decryption, the proposed hybrid encryption efficiency is approximately 45 to 457 times higher than existing PHE schemes.

C. Related Work

At USENIX SEC'15, Everspaugh et al. [12] introduced the first Password Hardening Service (PHS) named Pythia, which utilizes a verifiable PO-PRF. Pythia offers flexible and easily deployable solutions for password cryptography. The construction $F_{k_w}(t,m) = e(H_1(t), H_2(m))^{k_w}$ in Pythia aligns with the left-or-right constrained PRF [5]. The server holds the client key k_w, which is derived from its root key msk, with key rotations passively managed by the server. The server computes both the existing k_w and the updated client key k'_w, then provides the client with an update token $\Delta_w = k'_w/k_w$. We acknowledge that a server breach exposes the client key. Furthermore, the client cannot secure the record through key rotation when compromised, as the adversary can update the PRF value $F_{k_w}(t,m)$ using the update token Δ_w.

At CCS'16, Schneider et al. [35] demonstrated that Pythia [12] depends on a robust interactive assumption. Schneider et al propose a PHS scheme named Partially Oblivious Commitment (PO-COM), and derive three security goals: hiding, binding, and obliviousness, where PO-COM is a variant of the Pedersen commitment scheme [29]. In the enrollment record T, the client stores $T_3 = g^{sxy} \cdot m^{csk}$, where m is the password, csk is client key, y is a nonce, s and x are the derived key and secret key of the server, respectively. g^{sxy} is calculated in server-side. In the validation phase, the attack proceeds as follows. First, a compromised client forges $v = {m_2}^{r \cdot csk}$ and sends the validation request to the server, where the password m_2 can be generated randomly. Then, the client verifies the password using the returned parameter A_2, where $A_2 = T_1^{z_2}(T_3/{m_2}^{csk})^{h'_2} = T_1^{z_2}(g^{sxy} \cdot m^{csk}/{m_2}^{csk})^{h'_2}$, m is the real password. If the guessed $m_2 = m$, then $T_3/{m_2}^{csk} = g^{sxy} = c_3/v$, where c_3 and v are available in the validation phase.

At USENIX SEC'17, Lai et al. [24] pointed out that the server's failure to validate the client's request causes the offline guessing attack in PO-COM [35]. Lai et al. introduced an enhanced scheme called Phoenix [24], which relies on the standard decisional Diffie-Hellman assumption and stronger security definitions. The server stores an additional username un and auxiliary data $aux = (n_S, n_C)$ for retrieval in the validation phase. However, the explicit username poses privacy concerns. The intuitive definition of forward secrecy in pythia [12] states that key rotation should render old keys invalid, Lai et al. formalize this concept into a security experiment. In the key rotation phase, the client computes $k'_C = \alpha \cdot k_C$ to update the old key k_C, with the update factor α provided by the server. However, backward secrecy in key rotation remains unachievable. Jia et al. [19] identified two weaknesses in Phoenix [24]: the client cannot verify the server during enrollment, and both the username and verification results are exposed to the server.

At USENIX SEC'18, Lai et al. proposed the PHE [23] service. PHE extends PHS, and they share similar architectures. The difference is that PHE encrypts messages based on passwords, rather than verifying passwords. PHE [23] does not follow the encryption variant of Phoenix [24]. To encrypt a message M, the service provider and the rate-limiter select random numbers n_S and n_R, respectively, and interact to compute $(H^x_{R,0}H^y_{S,0}, H^x_{R,1}H^y_{S,1}M^y)$, where $H_{R,b} = H_R(n_R, b)$ and $H_{S,b} = H_R(n_R, pw, b)$ are hash functions that output group elements, x and y are the secret keys of the rate-limiter and the service provider, respectively. The service provider stores $(H^x_{R,0}H^y_{S,0}, H^x_{R,1}H^y_{S,1}M^y)$ as the enrollment record. To decrypt, the service provider calculates $H^y_{S,0}$ to recover $H^x_{R,0}$ and sends $H^x_{R,0}$ to the rate-limiter. After verifying the password, the rate-limiter returns $H^y_{R,1}$. The service provider recovers M.

Lai et al. [23] introduces a new security requirement named (strong) soundness: "1) The rate-limiter responds identically to the same record and password-label pair; 2) The rate-limiter does not convince the service provider that different password-label pairs correspond to the same result." To prevent malicious behavior of a compromised rate-limiter, PHE's rate-limiter additionally stores random values (n_S, n_R) to verify the request in decryption interactions, and returns a zero-knowledge proof of the verification result. However, external servers would be curious about user information. The rate-limiter can infer the verification result by verifying $T_0 H^{-y}_{S,0} = H^x_{R,0}$ and record the user's behaviors.

We argue that a malicious rate-limiter deceiving the server by marking a correct password as incorrect is equivalent to a denial of service, which cannot be avoided in a single-server architecture. It is important to note that a malicious rate-limiter should not be able to convince the service provider that an incorrect password is correct, as this poses a greater threat. If successful, any adversary could use an incorrect password to legitimately access an account. In Section II-C, we define (strong) soundness as "the PH server cannot convince the service provider that it decrypted the message with an incorrect password." We also introduce *privacy* as a new security property, highlighting that the external server should not access the verification results.

Ha et al. proposed an Anonymous PHE (APHE) scheme [14] based on PHE [23] that achieves cross-epoch anonymity by utilizing the trusted execution environment (an enclave [3]) provided by Intel SGX. Both the server and the rate-limiter require additional deployment of the enclave, where the enclave generates secret values to protect requests. The secret values change across different epochs to achieve unlinkability. The encryption and decryption structure of APHE is the same as PHE. Furthermore, we discover that, like PHE, APHE's C_0 leaks the result of the user's password verification to the rate-limiter. Even across different epochs, the rate-limiter can still recover the same $C_0 = H(x_K, n, 0)$ and the random number n, which satisfies cross-epoch linkability. Therefore, the anonymity claimed by APHE is difficult to achieve.

At CCS'20, Brost et al. introduced a multi-server threshold scheme (t, m)-PHE [6] based on the Shamir secret sharing [36] and ElGamal encryption scheme [11] to address Single Point of Failure (SPOF). (t, m)-PHE weakens individual external servers, the rate-limiting servers directly store shadow keys in the enrollment phase. (t, m)-PHE [6] transforms the encryption of M in PHE [23] from the form of $H_R^x(n_R, 1) \cdot H_S^y(pw.n_S, 1) \cdot M^y$ to $H_1(pw, n) \cdot H_1(n)^{\bar{s}_0} \cdot M$, where n is a random number shared by rate limiters, and $\bar{s}_0$ is secret-shared among m rate-limiters via Shamir secret sharing.

The security model of (t, m)-PHE [6] is limited to semi-adaptive corruption: the adversary must declare the set of corrupt parties before the key rotations. Brost et al. contend that an effective and secure fully adaptively construction does not exist. In the encryption (enrollment) phase, the client interacts with at least t (threshold) servers for 3 messages. In the decryption (verification) stage, the aforementioned interaction extends to 3 rounds. Throughout the interaction, the client iteratively undergoes Shamir key recovery and navigates authentication with rate-limiting servers. However, the threshold strategy employed in (t, m)-PHE suffers from scalability limitations and an inability to detect malicious behavior. Consequently, the presence of overhead and cycles could pose significant obstacles to deployment.

In PHE-based cryptosystems, besides the algorithm (i.e., PHE schemes), there are the secret keys (i.e., the passwords). The contrary, the security of passwords has gained significant attention and a number of studies have focused on enhancing the security [4], [28], [34]. To help users create robust passwords, the password strength meter [8] provides real-time feedback on the strength of passwords. There are also secure password storage [37] and password managers [30]. Different from smart cards which represent user possession and biometric information which represents user attributes, passwords represent knowledge and can be updated and are less prone to loss [32].

TABLE I
TERMINOLOGY CORRESPONDENCE TABLE FOR ROLES

Scheme	Client (User-Side)	PH Server (Auxiliary Server)
Pythia [12]	Client (stores T)	Server (stores ck)
PO-COM [35]	Client (stores T)	Server (stores sk)
Phoenix [24]	Client (stores T)	Server (stores un and sk)
PHE [23]	Service Provider (stores T)	Rate-limiter (stores nonces and rk)
APHE [14]	Service Provider/Server (deploys SGX enclave, stores T)	Rate-limiter (stores nonces and rk)
(t, m)-PHE [6]	Client (performs Shamir key recovery)	Rate-limiting Servers (store shadow keys)
HPHE (This Work)	Service Provider (Client) (generates master key, stores T)	PH Server (stores sk)

T: Enrollment record. ck: Client key. sk: Server key. rk: The key of rate-limiter.

In PHE schemes, the "PH Server" (regardless of its name: Server, Rate-limiter, Rate-limiting Servers) shares identical core responsibilities: (1) Enforcing access frequency limits to resist online password guessing; (2) Participating in cryptographic computations (e.g., key derivation, request verification) to assist the client in encryption/decryption. The client/user-side role also has consistent duties: managing user credentials, storing data/records, and initiating interactions with the PH server. Table I eliminates potential confusion by explicitly mapping these naming differences to their unified roles.

II. OVERVIEW

A. Preliminaries

Ideally, encrypting different files with different keys increases the complexity for attacks compared to reusing the same key. Additionally, encrypting with low-entropy and reused passwords weakens the security and practicality of password-based encryption. To address the challenges in password encryption systems, this work utilizes Puncturable Pseudorandom Functions (PPRF) [5], [21] to dynamically manage and derive high-entropy keys within PHE system.

PPRF is a pseudorandom function that supports a puncturing operation. This operation transforms the PRF key k into a new key k^*, where k^* is punctured on a set $S \subseteq \{0,1\}^{m(\lambda)}$ with the following properties.

For $x^* \notin S$, $F_k(x^*) = F_{k^*}(x^*)$. For $x \in S$ and punctured key $k^* = puncture(k, S)$, the value $F_{k^*}(x)$ is computationally indistinguishable from random.

1) Security of Puncturable Pseudorandom Functions: The selective security game is as follows:

- *Setup*: The challenger $\mathcal{C}$ randomly selects a PRF key k.
- *Query* 1: (Selective) The adversary $\mathcal{A}$ first commits x^*. $\mathcal{A}$ submits a polynomial number of evaluation requests. For each request $x_i \neq x^*$, $\mathcal{C}$ provides $\mathcal{A}$ with $F_k(x_i)$.

- *Challenge*: $\mathcal{A}$ chooses challenge x^*. $\mathcal{C}$ computes punctured key $k^* = \mathsf{Puncture}(k, \{x^*\})$ and samples $b \leftarrow \{0,1\}$. If $b = 0$, send $(k^*, F_k(x^*))$; if $b = 1$, send (k^*, y) where $y \leftarrow \{0,1\}^\lambda$.
- *Query* 2: $\mathcal{A}$ makes more evaluation queries on $x_i \neq x^*$.
- *Guess*: $\mathcal{A}$ outputs b'. A wins if $b' = b$ and x^* was not queried in *Query* 1 or 2.

Definition 1 (Selective PPRF Security): Let E be the set of evaluation requests. The adversary $\mathcal{A}$ wins if $b' = b$ and $x^* \notin E$. The advantage is defined as $Adv_{\mathcal{A}}^{PPRF}(\lambda)$. PPRF is selectively secure if for all PPT adversaries, $Adv_{\mathcal{A}}^{PPRF}(\lambda) \leq negl(\lambda)$, where $negl(\lambda)$ is a negligible function.

The full proof of Theorem 1, including hybrid arguments, reduction to H_t's PRF security, and explicit truncation analysis via the Leftover Hash Lemma, is provided in Section IV. For the *storage and security implications of punctured keys*, please see the Appendix A of the full version available at https://wangdingg.weebly.com/publications.html.

B. Description of Construction

The proposed scheme is denoted as Π, involving the service provider (also referred to as "client" $\mathcal{C}$) and external PH server ("server" $\mathcal{S}$). Let $\mathbb{G}$ be a finite multiplicative cyclic group of order q. Let $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ and $H_t : \{0,1\}^* \rightarrow \{0,1\}^{2\lambda}$ be hash functions. Let $k \in \{0,1\}^\lambda$ be the root encryption key. The main cryptographic algorithms are as follows:

1) Setup Phase: In the setup phase, the generation algorithm $\Pi.\mathsf{Setup}(1^\lambda)$ produces and publishes the public parameters $\mathsf{pp} = \{F_{(.)}(.), g, H(.)\}$, where $F_{(.)}(.)$ is a generation function of the PPRF constructed by H_t. The PH server uses the key generation algorithm $\Pi.\mathsf{KGen}(1^\lambda)$ to generate a public-private key pair and publishes the public key.

2) Enrollment/Encryption Phase: In the enrollment/encryption phase, the user inputs account information, including username un and password pw. The service provider and PH server interact to perform this phase. The service provider generates a long key k for deriving encryption keys. For the input key k and multiple files, the function $F_k(.)$ distributes sufficient keys. Ultimately, $\Pi. < \mathcal{C}(un, pw, k, k_c, PK_s), \mathcal{S}(k_s, PK_s) >_{erl}$ returns the enrollment record T and the encrypted files.

3) Decryption Phase: In the decryption phase, the user inputs candidate username un' and password pw'. The service provider and PH server interact to perform this phase. If the candidate information is correct, that is, $un = un'$ and $pw = pw'$, $\Pi. < \mathcal{C}(un', pw', k_c, PK_s, T), \mathcal{S}(k_s, PK_s) >_{dec}$ returns the root key k. At the same time, the PH server limits online access frequency based on the parameter E to counter online password guessing.

4) Key Puncture & Update Phase: This phase supplements the decryption phase, allowing users to perform key puncture to achieve secure deletion or limit decryption permissions. When a legitimate user decrypts the root key k, she can obliterate the derived keys related to the target files, share, or update the punctured keys.

5) Key Rotation Phase: The key rotation consists of a client key rotation protocol $\Pi. < \mathcal{C}(k_c, k_c', T) >_{Crot}$ and a server key rotation protocol $\Pi. < \mathcal{C}(k_c, PK_s, T), \mathcal{S}(k_s, PK_s, k_s', PK_s') >_{Srot}$. On executing the client key rotation, the protocol returns new key k_c' and new record T' to the client. For the server key rotation, the protocol returns new key pair (k_s', PK_s') to the server, PK_s' and new record T' to the client.

C. Security of PHE

First, we describe the adversary. An adversary of PHE or PHS has the following capabilities and limitations: (1) She cannot compromise both the service provider and the PH server simultaneously;[1] (2) She can obtain stored enrollment records or server's secret keys after compromising a device, which could potentially lead to unauthorized access to critical system resources and sensitive user information; (3) She may recover recently sent or received messages after compromising a device (as parties communicating via TLS cache session keys and messages [27], [33]), thereby gaining insights into the communication patterns and potentially sensitive data exchanged; (4) She holds dictionaries for online and offline password guessing, enabling her to launch persistent attacks on the password-based security measures; (5) Due to the specific nature of the server deployment, she cannot physically take over the server entirely to interfere with key rotation, thus ensuring a certain level of security in the long-term key management of the system.

Throughout the PHS iteration, researchers are constantly improving and updating security requirements and definitions. In Phoenix, Lai et al. [24] add forward security to the original hiding, binding, and (partially) oblivious [12]. Lai et al. [23] extend the security properties of PHS to adapt to PHE and define *Message Hiding*, *Partial Obliviousness*, *(Strong) Soundness*, and *Forward Security*. In this section, we first introduce the extended security requirements proposed in this work, then provide proof in Section IV that the proposed protocol meets these requirements.

1) Message Hiding: Given the service provider's secret key k_c and the enrollment record T, the adversary cannot distinguish the encrypted message corresponding to record T. Specifically, *message hiding* requires that the adversary cannot distinguish whether the record is encrypting m_0 or m_1, even if the message is chosen by the adversary and the adversary possesses a dictionary containing the password.

The corresponding game is named PH-Hi and is played in two phases: in the first phase, the honest service provider and PH server act as challengers $\mathcal{R}_c$ and $\mathcal{R}_s$ respectively, while in the second phase, the compromised service provider is considered the adversary $\mathcal{A}$. PH-Hi is defined as follows:

- Setup 1: $\Pi.\mathsf{Setup}(1^\lambda)$ generates public parameters pp based on the secret parameter λ. The PH server challenger $\mathcal{R}_s$ generates a key pair (k_s, PK_s) using

[1]This is because any password-related cryptographic protocol will be insecure when both devices are compromised, i.e., susceptible to offline password guessing attacks, as pointed out in [18] and [43]. This is called the plain case in password-related cryptographic protocols (including PHS and PHE). Just like in existing password-based authentication schemes(e.g., [1], [10]) where only two parties are involved (i.e. user and authentication server), passwords are always susceptible to offline password guessing attacks once the server is corrupted [15].

$\Pi.\mathsf{KGen}(1^\lambda)$. The service provider challenger generates a secret key k_c and sends it to the adversary $\mathcal{A}$.

- Setup 2: $\mathcal{A}$ can access the oracle $\mathbb{O} \leftarrow \{\Pi. < \mathcal{A}, \mathcal{A} >_X, \Pi. < \mathcal{A}, \mathcal{R}_s >_X, X \in \{erl, dec, S\,rot, C\,rot\}\}$ and generates a username un, a password distribution $\mathbb{D}$, and two different messages m_0 and m_1. Parameters $\{un, \mathbb{D}, (m_0, m_1)\}$ are sent to the challenger $\mathcal{R}_c$.
- Challenge: The service provider challenger $\mathcal{R}_c$ randomly selects a bit $b \leftarrow \{0, 1\}$ and a password $pw \leftarrow \mathbb{D}$, then performs the enrollment protocol $\Pi. < \mathcal{R}_c(un, pw, m_b, k_c, PK_s), \mathcal{R}_s(k_s, PK_s) >_{erl}$ to generate the record T. The record T is sent to the adversary $\mathcal{A}$.
- Output: $\mathcal{A}$ outputs the guessed b', $b' = b$ means $\mathcal{A}$ wins the game. The advantage of $\mathcal{A}$'s winning is defined as

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

Definition 2 (Message Hiding): Given a PHE protocol in $\mathbb{O}$ and a PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that $Adv^{\mathcal{A},\mathbb{O}}_{PH-Hi}(\lambda) \leq C' \cdot q^{s'} + negl(\lambda)$, where C' and s' are the linear regression parameters of the password distribution $\mathbb{D}$ [7], q is the number of decryption executions, protocol can be considered as message hiding.

2) Partial Oblivious: This property hides both the password and the encrypted message, protecting against an adversary that compromises the PH server. The 'partial' nature comes from the need to resist online password guessing, the PH server limits access frequency based on a fixed parameter (e.g., tweak t in PO-COM [35] and Pythia [12], un in Phoenix [24]). In this work, fresh random numbers are incorporated into decryption requests to ensure that usernames remain confidential.

The corresponding game is called PH-Ob and is played in two phases: in the first phase, the honest service provider and PH server act as challengers $\mathcal{R}_c$ and $\mathcal{R}_s$ respectively, while in the second phase, the compromised PH server is considered the adversary $\mathcal{A}$. PH-Ob is defined as follows:

- Setup 1: $\Pi.\mathsf{Setup}(1^\lambda)$ generates public parameters pp based on the secret parameter λ. The PH server challenger $\mathcal{R}_s$ generates a key pair (k_s, PK_s) using $\Pi.\mathsf{KGen}(1^\lambda)$. The service provider challenger generates a secret key k_c. The key pair (k_s, PK_s) is sent to the adversary $\mathcal{A}$.
- Setup 2: $\mathcal{A}$ can access the oracle $\mathbb{O} \leftarrow \{\Pi. < \mathcal{A}, \mathcal{A} >_X, \Pi. < \mathcal{A}, \mathcal{R}_c >_X, X \in \{erl, dec, S\,rot, C\,rot\}\}$ and generates a username un, two password-message pairs (pw_0, m_0) and (pw_1, m_1). Parameters $\{un, (pw_0, m_0), (pw_1, m_1)\}$ are sent to the service provider challenger $\mathcal{R}_c$.
- Challenge: The service provider challenger $\mathcal{R}_c$ randomly selects $b \leftarrow \{0, 1\}$, then performs the enrollment protocol $\Pi. < \mathcal{R}_c(un, pw_b, m_b, k_c, PK_s), \mathcal{R}_s(k_s, PK_s) >_{erl}$ to generate the record T. The record T is sent to $\mathcal{A}$.
- Output: $\mathcal{A}$ outputs the guessed b', $b' = b$ means $\mathcal{A}$ wins the game. The advantage of $\mathcal{A}$'s winning is defined as

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

Definition 3 (Partial Obliviousness): If for a PHE protocol in $\mathbb{O}'$ and PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that $Adv^{\mathcal{A},\mathbb{O}'}_{PH-Ob}(\lambda) \leq negl(\lambda)$, then this PHE protocol can be considered as partial oblivious.

3) Soundness: Soundness ensures the binding of an enrollment record, encrypted messages, and a password. The records are generated by an honest server and a PH server, and the encrypted messages can only be recovered by participating in the decryption protocol with the correct password. Further, we define *strong soundness*, which not only includes the aforementioned properties but also detects the behavior of a malicious PH server. After the service provider and the PH server interact during enrollment, a malicious/compromised PH server cannot convince the service provider that it decrypted the message with an incorrect password. Additionally, a malicious/compromised PH server cannot convince the service provider that the same password and enrollment record correspond to different decryption results.

The corresponding game is called PH-Sd and is played between the honest service provider and the malicious PH server. The provider and the PH server are regarded as the challenger $\mathcal{R}$ and the adversary $\mathcal{A}$, respectively. The game description is as follows:

- Setup: $\Pi.\mathsf{Setup}(1^\lambda)$ generates public parameters pp based on the secret parameter λ. The adversary $\mathcal{A}$ generates a key pair (k_s, PK_s) using $\Pi.\mathsf{KGen}(1^\lambda)$. The service provider challenger generates a secret key k_c.
- Setup 2: The adversary $\mathcal{A}$ can access the oracle $\mathbb{O} \leftarrow \{\Pi. < \mathcal{A}, \mathcal{A} >_X, \Pi. < \mathcal{A}, \mathcal{R} >_X, X \in \{erl, dec, S\,rot, C\,rot\}\}$. $\mathcal{A}$ generates two different pairs (un_0, pw_0, m_0) and (un_1, pw_1, m_1). Two pairs are sent to the challenger $\mathcal{R}$.
- Challenge: The challenger $\mathcal{R}$ randomly selects $b \leftarrow \{0, 1\}$, and performs the enrollment protocol $\Pi. < \mathcal{C}(un_b, pw_b, m_b, k_c, PK_s), \mathcal{S}(k_s, PK_s) >_{erl}$ to generate the enrollment record T.
- Output: The challenger $\mathcal{R}$ interacts with the adversary $\mathcal{A}$ to perform $\Pi. < \mathcal{C}(un_0, pw_0, k_c, PK_s, T), \mathcal{S}(k_s, PK_s) >_{dec}$ and $\Pi. < \mathcal{C}(un_1, pw_1, k_c, PK_s, T), \mathcal{S}(k_s, PK_s) >_{dec}$, the verification results r_0 and r_1 are returned to $\mathcal{R}$. Meanwhile, $r_0 = r_1$ means $\mathcal{A}$ wins the game. The advantage of $\mathcal{A}$'s winning is defined as

$$Adv^{\mathcal{A},\mathbb{O}}_{PH-Bi}(\lambda) = |\Pr[b_0 \wedge b_1] - \frac{1}{2}|. \tag{3}$$

where $b_0 :=$ ($r_0 =$ "1" or "0") and $b_1 :=$ ($r_1 =$ "1" or "0"). $b_0 \wedge b_1 = 1$ represents $r_0 = r_1 =$ "1" or "0".

Definition 4 (Strong Soundness): If for a PHE protocol in $\mathbb{O}$ and PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that $Adv^{\mathcal{A},\mathbb{O}}_{PH-Sd}(\lambda) \leq negl(\lambda)$, then this PHE protocol can be considered as strong sound.

4) Forward Security: Key rotation should invalidate old keys and enrollment records. The adversary should not be able to obtain the rotated keys, even if she compromises the device and eavesdrops on messages. We define games PH-FS1 and PH-FS2 to formalize the forward security requirements of the service provider and PH server, respectively.

The game PH-FS1 is used to define the forward security of the service provider's key and is played between the honest service provider in the first phase, the PH server, and the

compromised service provider in the second phase. They are regarded as the challenger $\mathcal{R}c$, the PH server $\mathcal{R}s$, and the adversary $\mathcal{A}$. The description is as follows:

- Setup 1: $\Pi.\mathsf{Setup}(1^\lambda)$ generates public parameters pp based on the secret parameter λ. The PH server challenger $\mathcal{R}_s$ generates a key pair (k_s, PK_s) using $\Pi.\mathsf{KGen}(1^\lambda)$. The service provider challenger generates a secret key k_c and sends it to the adversary $\mathcal{A}$.
- Setup 2: The adversary $\mathcal{A}$ can access the oracle $\mathbb{O} \leftarrow \{\Pi. < \mathcal{A}, \mathcal{A} >_X, \Pi. < \mathcal{A}, \mathcal{R} >_X, X \in \{erl, dec, Srot, Crot\}\}$. $\mathcal{A}$ generates two different pairs (un_0, pw_0, m_0) and (un_1, pw_1, m_1). Two pairs are sent to the challenger $\mathcal{R}_c$.
- Challenge: The challenger $\mathcal{R}_c$ randomly selects $b \leftarrow \{0, 1\}$, and performs the enrollment protocol $\Pi. < \mathcal{R}_c(un_b, pw_b, m_b, k_c, PK_s), \mathcal{R}_s(k_s, PK_s) >_{erl}$ to generate the enrollment record T. $\mathcal{R}_c$ sends T to the adversary $\mathcal{A}$, then generates a new key k'_c and performs the key rotation protocol $\Pi. < \mathcal{C}(k_c, k'_c, T) >_{Crot}$. T is updated to T'. $\mathcal{A}$ holds the original record T and can access $\mathbb{O}$.
- Output: $\mathcal{A}$ outputs the guessed b', $b' = b$ means $\mathcal{A}$ wins the game. The advantage of $\mathcal{A}$'s winning is defined as

$$Adv^{\mathcal{A},\mathbb{O}}_{PH-FS1}(\lambda) = |\Pr[b' = b] - \frac{1}{2}|. \tag{4}$$

The game PH-FS2 is used to define the forward security of the PH server's key and is played between the client, the first-phase honest server, and the second-phase compromised server, they are regarded as the challenger $\mathcal{R}_c$, $\mathcal{R}_s$, and the adversary $\mathcal{A}$. The description is as follows:

- Setup 1: $\Pi.\mathsf{Setup}(1^\lambda)$ generates public parameters pp based on the secret parameter λ. The PH server challenger $\mathcal{R}_s$ generates a key pair (k_s, PK_s) using $\Pi.\mathsf{KGen}(1^\lambda)$. The service provider challenger generates a secret key k_c. The key pair (k_s, PK_s) is sent to the adversary $\mathcal{A}$.
- Setup 2: $\mathcal{A}$ can access the oracle $\mathbb{O} \leftarrow \{\Pi. < \mathcal{A}, \mathcal{A} >_X, \Pi. < \mathcal{A}, \mathcal{R}_c >_X, X \in \{erl, dec, Srot, Crot\}\}$. $\mathcal{A}$ generates two different pairs (un_0, pw_0, m_0) and (un_1, pw_1, m_1). Two pairs are sent to the challenger $\mathcal{R}_c$. Two pairs are sent to the challenger $\mathcal{R}_c$.
- Challenge: The challenger $\mathcal{R}_c$ randomly selects $b \leftarrow \{0, 1\}$, and performs the enrollment protocol $\Pi. < \mathcal{R}_c(un_b, pw_b, m_b, k_c, PK_s), \mathcal{R}_s(k_s, PK_s) >_{erl}$ to generate the enrollment record T. $\mathcal{R}_c$ sends T to $\mathcal{A}$.
  The PH server challenger $\mathcal{R}_s$ generates a key pair (k'_s, PK'_s) using $\Pi.\mathsf{KGen}(1^\lambda)$ and performs the key rotation protocol $\Pi. < \mathcal{C}(k_c, PK_s, T), \mathcal{S}(k_s, PK_s, k'_s, PK'_s) >_{Srot}$. T is updated to T'. $\mathcal{A}$ holds the original record T and can access the oracle $\mathbb{O}$.
- Output: $\mathcal{A}$ outputs the guessed b', $b' = b$ means $\mathcal{A}$ wins the game. The advantage of $\mathcal{A}$'s winning is defined as

$$Adv^{\mathcal{A},\mathbb{O}}_{PH-FS2}(\lambda) = |\Pr[b' = b] - \frac{1}{2}|. \tag{5}$$

Definition 5 (Forward Security): If for a PHE protocol in $\mathbb{O}$ and any PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that $Adv^{\mathcal{A},\mathbb{O}}_{PH-FS1}(\lambda) \le negl(\lambda)$ and $Adv^{\mathcal{A},\mathbb{O}}_{PH-FS2}(\lambda) \le negl(\lambda)$, this protocol is forward secure.

5) (Multi-Query) Privacy: In addition to the rate-limiting parameters, the external PH server should not be aware of parameters that track legitimate user login habits or expose user information, such as usernames and authentication results.

Previous researches (Pythia [12], PO-COM [35], PW-Hero [19]) reveal the fixed tweak t (or username un in Phoenix [24], C_0 in PHE [23], (t, m)-PHE [6], and APHE [14]) to the server for rate-limiting. Therefore, in the definition of privacy, we restrict the PH server to only provide rate-limiting and prohibit it from accessing other information related to users.

The corresponding game is called PH-Pr and played between the service provider and the malicious PH server, they are regarded as the challenger $\mathcal{R}$ and the adversary $\mathcal{A}$. The game description is as follows:

- Setup: $\mathcal{C}$ runs $\Pi.\mathsf{Setup}(1^\lambda)$ to generate pp, and $\Pi.\mathsf{KGen}(1^\lambda)$ to generate PH server keys (ks, PK_s). $\mathcal{C}$ samples a random password $pw \leftarrow \mathcal{D}$ (from a password dictionary) and username un. $\mathcal{C}$ generates a root key $k \leftarrow \{0, 1\}^\lambda$ and runs the enrollment protocol to produce record T (as in $\Pi.\langle C(un, pw, k, kc, PK_s), S(ks, PK_s)\rangle^{erl}$).
- Query: Given pp, PK_s, T (simulating compromise of the service provider's storage). $\mathcal{A}$ adaptively issues up to $q(\lambda)$ decryption queries. For each query $i \in [1, q]$
  $\mathcal{A}$ chooses a candidate password pw'_i and username un'_i. $\mathcal{C}$ samples bit $b_i \leftarrow \{0, 1\}$. If $b_i = 0$, $\mathcal{C}$ uses the correct (un, pw) for the interaction; if $b_i = 1$, $\mathcal{C}$ uses the incorrect (un'_i, pw'_i). $\mathcal{C}$ simulates the decryption protocol $\Pi.\langle C(un'_i, pw'_i, kc, PK_s, T), S(ks, PK_s)\rangle^{dec}$, and sends the PH server-side parameters to $\mathcal{A}$. $\mathcal{A}$ does not see the final key k or verification result.
- Challenge: After all queries, $\mathcal{A}$ selects an index $j \in [1, q]$ and outputs a guess b'_j for b_j. $\mathcal{A}$ wins if $b'_j = b_j$. The advantage of $\mathcal{A}$'s winning is defined as

$$Adv^{\mathcal{A},\mathbb{O}}_{PH-MPr}(\lambda) = |\Pr[b' = b] - \frac{1}{2}|. \tag{6}$$

Definition 6 (Privacy): If for a PHE protocol in oracle $\mathbb{O}$ and PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that $Adv^{\mathcal{A},\mathbb{O}}_{PH-MPr}(\lambda) \le negl(\lambda)$, then this PHE protocol can be considered as private.

III. Our HPHE Scheme

The PHE construction proposed in this work does not originate from existing PHS schemes and is entirely different from the initial PHE construction [23] and its derived schemes [6], [14]. We implicitly compare the hashed candidate password with random values in the exponent, avoiding the password verification result leakage associated with explicit comparison [14], [23], [24] and offline password guessing without randomness [12], [35]. Additionally, we extend the usability of PHE, enabling users to securely revoke or restrict access to shared encrypted files. The converter can be directly applied to existing PHE schemes.

A. Construction Idea

To begin with, we explain the thought process behind the scheme's design. We observe that encrypting or decrypting

a single message (selected from group elements) using PHE requires at least two rounds of interaction. The overhead is approximately four times that of public-key encryption with equivalent parameters, and 4×10^3 to 6×10^3 times that of symmetric encryption [25]. Therefore, PHE is not suitable for large-scale message encryption. A practical solution is for the service provider to use PHE to encrypt the high-entropy root key. For users, switching from PHE-based message encryption and decryption to symmetric encryption using the high-entropy root key is seamless, significantly improving efficiency when handling large-scale data.

Although the above transformation addresses the issue of encryption and decryption efficiency, using a single key to encrypt all messages leads to key reuse problems and makes it inconvenient for users to manage the ciphertext. Consider a scenario where a service provider offers encrypted cloud storage. Although irrecoverable secure deletion is technically feasible, in practice cloud providers still struggle to meet users' expectations for secure deletion due to factors such as distributed data redundancy, backup policies, and constraints imposed by legacy systems. Moreover, if all ciphertexts are encrypted with the same key, users are unable to restrict decryption permissions for shared ciphertexts, potentially compromising the security of other ciphertexts. In this work, we construct a hash-based Puncturable Pseudorandom Functions (PPRF) for large-scale encryption and management of ciphertext, which is also applicable to other PHE schemes.

PPRF derives a high-entropy root key used to encrypt different messages with multiple high-entropy sub-keys. PPRF provides puncturable sub-keys, where punctured keys cannot be recovered. When a user deletes a (batch of) ciphertext, the root key is updated to a punctured key. Even if the ciphertexts are later maliciously recovered by the service provider, it is not possible to obtain the corresponding decryption keys. When a user shares a batch of ciphertexts, puncturing other keys allows for shared keys with restricted access.

For the first time, we construct a PHE scheme that hides the verification result from the external PH server (rate-limiter). During the enrollment phase, the service provider computes a salted hash $q_c \leftarrow H(un \parallel pw \parallel r_c)$ based on the user's username un and password pw, then interacts with the PH server to compute $K_{cs} \leftarrow k^{d_c k_s}$ and $Q_c \leftarrow k^{u_c q_c}$, where $d_c \leftarrow q_c^2 k_c$, k_c and k_s are secret keys of the service provider and PH server, respectively, u_c is a random number chosen by the service provider and is encrypted with the PH server's public key in the enrollment record, k is a high-entropy root key selected by the service provider. Thus, in the enrollment record, the exponents of k in the two parameters K_{cs} and Q_c respectively contain the PH server's private key k_s and the random number u_c encrypted by PH server's public key. An adversary cannot recover k or guess the password without knowing k_s and u_c.

The use of $q_c^2 k_c$ (rather than a linear $q_c k_c$) in d_c is motivated by the need to prevent malleability in the blinded interaction. Intuitively, the quadratic blinding amplifies randomness. Meanwhile, this design ensures that only when the candidate password is correct, the exponent simplification in the decryption phase can be satisfied, which is a necessary condition for

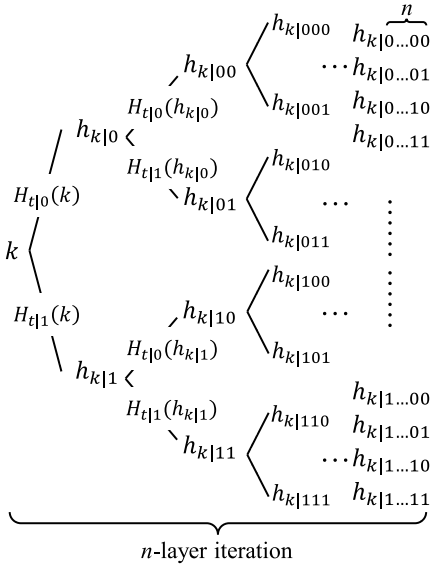


Fig. 2. Schematic diagram of the structure of a diffusion hash chain. When iterating n layers, a total of 2^n derived keys are derived, denoted as $h_{k|0...00}$ to $h_{k|1...11}$.

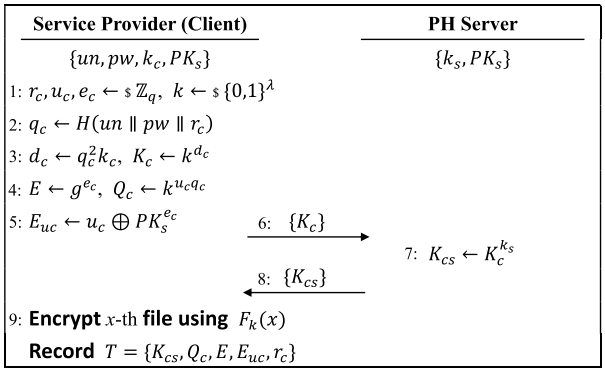


Fig. 3. The enrollment/encryption phase of HPHE. The service provider and the PH server perform one round of interaction to jointly generate an encrypted record T and derive a high-entropy root key k, which is then used to symmetrically encrypt multiple files under the one-data-one-key policy.

recovering the root key k and maintaining strong soundness. A linear $q_c k_c$ would require additional NIZK proofs, thus undermining the intended efficiency improvement.

During decryption, the service provider computes $q_c' \leftarrow H(un' \parallel pw' \parallel r_c)$ based on the candidate username un' and candidate password pw'. The basic idea of PHE (PHS) for verifying the candidate password is that if the candidate q' matches q of the enrollment record, then q can be easily reduced in the final parameters. The challenge in constructing the interaction protocol lies in preventing a malicious service provider from performing offline password guessing (as seen in the attack faced by PO-COM [35], with steps detailed in Phoenix [24]) after obtaining parameters derived from the original password hash. In this work, we construct a novel encryption and reduction strategy utilizing randomness, where both parties use temporary secrets during each interaction to enhance security, addressing the aforementioned challenge and preventing either party from password guessing.

The service provider selects different random numbers t_c and n_c used as part of the exponents for $X_c \leftarrow K_{cs}^{t_c} (= k^{d_c k_s t_c})$

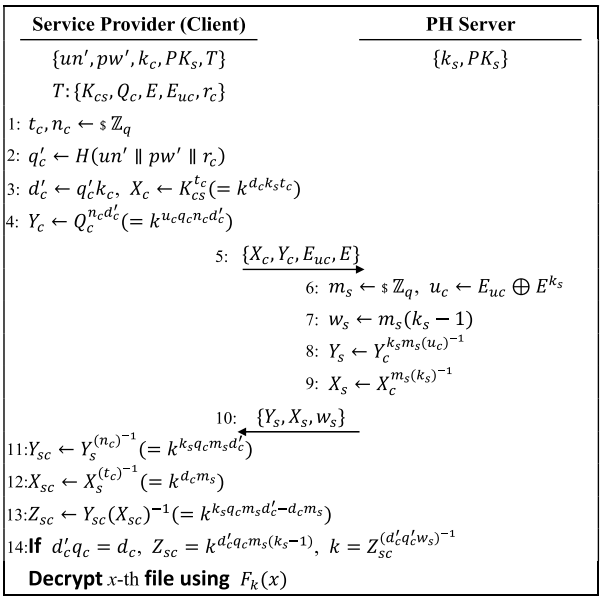


Fig. 4. The decryption phase of HPHE. The service provider and the PH server jointly verify the candidate username un' and password pw'. The PH server assists in eliminating its own secret from the computation, ensuring that password verification results remain hidden. The root key k is recovered only if pw' is correct.

and $Y_c \leftarrow Q_c^{n_c d'_c} (= k^{u_c q_c n_c d'_c})$ (see step 2 in Figure 4), where k is the high-entropy root key, k_s is the PH server's secret key, u_c is a random number encrypted by PH server's public key. Note that the construction of $d'_c \leftarrow q'_c k_c$ here differs from $d_c \leftarrow q_c^2 k_c$ in the enrollment phase. The exponent d_c in X_c and q_c in Y_c contain the real password from the enrollment phase, but the service provider cannot compare q_c and q'_c to verify the candidate password because the PH server's secret key k_s and random number u_c are unavailable. The service provider sends X_c and Y_c to the PH server to eliminate k_s and u_c. The PH server selects a random number m_s and computes $Y_s \leftarrow Y_c^{k_s m_s (u_c)^{-1}} (= k^{k_s m_s q_c n_c d'_c})$ and $X_s \leftarrow X_c^{m_s (k_s)^{-1}} (= k^{d_c m_s t_c})$ (see step 5 in Figure 4). The PH server returns Y_s, X_s, and $w_s \leftarrow m_s(k_s - 1)$ to the service provider.

After receiving Y_s and X_s, the service provider eliminates random numbers t_c and n_c to obtain $Y_{sc} \leftarrow Y_s^{(n_c)^{-1}} (= k^{k_s q_c m_s d'_c})$ and $X_{sc} \leftarrow X_s^{(t_c)^{-1}} (= k^{d_c m_s})$. Note that the random number m_s and the key k_s in exponents of parameters Y_{sc} and X_{sc} are unavailable to the service provider. It is unable to eliminate them and perform offline password guessing. The service provider computes $Z_{sc} \leftarrow Y_{sc}(X_{sc})^{-1} (= k^{k_s q_c m_s d'_c - d_c m_s})$ and merges the exponents into $k_s q_c m_s d'_c - d_c m_s$. At this point, if $q_c d'_c = q_c q'_c k_c = q_c q_c k_c = d_c$, then $k_s q_c m_s d'_c - d_c m_s = d'_c q_c m_s (k_s - 1) = d'_c q'_c w_s$. The service provider can obtain k by calculating $k = Z_{sc}^{(d'_c q'_c w_s)^{-1}}$ (see step 10 in Figure 4).

Next, we list the constructions that readers may have questions about and explain the design reasons.

Q1: *Why does the service provider select two different random numbers t_c and n_c (see step 1 in Figure 4)?*

A1: The two different random numbers are used to hide the result of the password verification. If the service provider select one random number t_c, then after the PH server eliminates k_s and u_c, the constructions of $X_c = k^{d_c t_c}$ and $Y_c = k^{q_c d'_c t_c}$ would be identical. At this point, the PH server could compare if $X_c = Y_c$ to learn whether q_c is equal to q'_c, thus revealing the result of the user's password verification.

Q2: *Why does the PH server remove the random number u_c and the secret key k_s from exponents of Y_c and X_s while simultaneously adding the parameter $k_s m_s$ and a random number m_s (see step 5 in Figure 4)?*

A2: Adding unrecoverable secret values prevents a malicious service provider from performing offline password guessing. After the PH server eliminates the secret values u_c and k_s in $Y_c = k^{u_c q_c n_c d'_c}$ and $X_c = k^{d_c k_s t_c}$, if no additional secrets are added, the service provider can obtain $Y_c = k^{q_c d'_c}$ and $X_c = k^{d_c t_c} = k^{d_c}$ and perform offline password guessing by calculating $Y_c^{d''_c{}^{-1} d''_c}$ and comparing it with X_c, where d''_c calculated based on other candidate passwords. If only one secret is used, the above attack could still be successful. Therefore, different secret values should be used to encrypt the two returned parameters. Here, k_s can also be replaced with other random numbers.

Q3: *In decryption, why doesn't the PH server simply eliminate u_c and k_s directly?*

A3: In the protocol construction, we assume that the other party may be malicious in order to avoid leaking characteristics that could be used for password guessing. Assuming the service provider is malicious, if the PH server directly eliminates u_c and k_s, the service provider can obtain k^{q_c}, making offline password guessing easy. Therefore, we construct a novel secret-based hiding strategy to restrict each decryption interaction to be mandatory, preventing the other party from obtaining an explicit combination of the password and ciphertext after a single interaction. From the analysis of the scheme, it can be seen that the key k can only be recovered if the candidate password is correct. The proposed scheme limits offline guessing and restricts password guessing to online attempts, which can be controlled in frequency.

B. Formal Description

1) Setup Phase: Inspired by the tree-based construction of PRF functions by Goldreich et al. [13], we utilize a tree structure to derive keys, which we refer to as the diffusion hash chain. The hash function is defined as $H_t : \{0,1\}^* \rightarrow \{0,1\}^{2\lambda}$. As shown in Figure 2, $H_{t|0}(.)$ and $H_{t|1}(.)$ represent the first half and the second half of the hash value, respectively, with the values denoted as $h_{k|0}$ and $h_{k|1}$. Note that $H_{t|0}(.)$ and $H_{t|1}(.)$ are the result of a truncation operation from a single hash, rather than two separate hash operations. When repeating $H_{t|0}(.)$ and $H_{t|1}(.)$ operations on the obtained values n times, a total of 2^n derived keys of length 2λ are generated, requiring $2^n - 1$ hash operations. The generation function of the PPRF can be denoted as $F_k(x) = H_{t|x(0)}(H_{t|x(1)}(\ldots H_{t|x(n)}(k))) = h_{k|x}$, where x is a number in binary representation, and $x(0), x(1), \ldots, x(n)$ represent the bit values of x from the most significant to the least significant bit, respectively.

When puncturing a certain key or branch, we retain the value of its neighboring branch as the new key k^*. Let's explain the puncturing process when $n = 3$ as shown in Figure 2. At this point, we get 8 keys denoted as $h_{k|000}$ to $h_{k|111}$. Suppose puncturing the key $h_{k|010}$, the punctured key

k^* will be $\{h_{k|00}, h_{k|011}, h_{k|1}\}$. We can compute all keys except $h_{k|010}$ based on the key k^*, and $h_{k|010}$ cannot be recovered.

Π denotes our scheme, the setup algorithm $\Pi.\mathsf{Setup}$ selects a cyclic group $\mathbb{G}$ of prime order q, and let g be a generator. Let $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ be a hash function. The parameters $\{F_{(.)}(.), g, H(.)\}$ are published as the public parameters pp. The PH server executes $\Pi.\mathsf{KGen}$ algorithm to generate a random private-public key pair $\{k_s, PK_s = g^{k_s}\}$. The service provider selects a secret key k_c.

2) Enrollment/Encryption Phase: The service provider and external PH server interact at this phase to encrypt multiple files and register based on a password. The corresponding steps are described in Figure 3.

EP1: The service provider holds its secret key k_c, the username un, user's password pw, and the public key PK_s of the PH server. The service provider selects random numbers from the cyclic group $\mathbb{G}$, denoted as $r_c, u_c, e_c \leftarrow \$\mathbb{Z}_q$. Then the service provider randomly generates a long root key $k \leftarrow \$\{0,1\}^\lambda$ to derive sub-keys for encryption. The service provider calculates the hash $q_c \leftarrow H(un \parallel pw \parallel r_c)$ and $d_c \leftarrow q_c^2 k_c$, then calculates $K_c \leftarrow k^{d_c}$ using d_c as the exponent of the root key k. Afterward, the service provider calculates $E \leftarrow g^{e_c}$ and $Q_c \leftarrow k^{u_c q_c}$ and encrypts u_c into $E_{uc} \leftarrow u_c \oplus PK_s^{e_c}$ using the public key PK_s of the PH server. The service provider sends K_c to the PH server.

EP2: The PH server holds its secret and public key pair $\{k_s, PK_s\}$. On receiving K_c, the PH server calculates $K_{cs} \leftarrow K_c^{k_s}$ and returns K_{cs} to the service provider.

EP3: The service provider constructs the corresponding PPRF F based on the number of files to be encrypted. For the x-th file, the service provider encrypts it using the derived key $F_k(x)$. The service provider stores the record $T = \{K_{cs}, Q_c, E, E_{uc}, r_c\}$ and the encrypted files.

3) Decryption Phase: The service provider and the PH server interact to decrypt. The corresponding steps are described in Figure 4.

DP1: After registration, the service provider holds the secret key k_c, public key PK_s, and the enrollment record $T = \{K_{cs}, Q_c, E, E_{uc}, r_c\}$. The user enters the candidate username un' and password pw'. The service provider selects random numbers, denoted as $t_c, n_c \leftarrow \$\mathbb{Z}_q$. The service provider calculates the corresponding hash $q'_c \leftarrow H(un' \parallel pw' \parallel r_c)$ and $d'_c \leftarrow q'_c k_c$ based on the candidate password. Additionally, it computes $X_c \leftarrow K_{cs}^{t_c} (= k^{d_c k_s t_c})$ and $Y_c \leftarrow Q_c^{n_c d'_c} (= k^{u_c q_c n_c d'_c})$ for the stored record T. Afterward, the service provider sends $\{X_c, Y_c, E_{uc}, E\}$ to the PH server.

DP2: The PH server holds its secret and public key pair $\{k_s, PK_s\}$. On receiving $\{X_c, Y_c, E_{uc}, E\}$, the PH server selects a random number from the cyclic group $\mathbb{G}$, denoted as $m_s \leftarrow \$\mathbb{Z}_q$. Afterward, the PH server recovers $u_c \leftarrow E_{uc} \oplus E^{k_s}$ from E^{k_s} and calculates $w_s \leftarrow m_s(k_s - 1)$ using the secret key k_s and the random number m_s. Note that since the random number m_s is unknown to the service provider, k_s cannot be obtained by the service provider. The PH server calculates $Y_s \leftarrow Y_c^{k_s m_s (u_c)^{-1}}$ and $X_s \leftarrow X_c^{m_s (k_s)^{-1}}$, and sends $\{Y_s, X_s, w_s\}$ to the service provider.

DP3: The service provider calculates $Y_{sc} \leftarrow Y_s^{(n_c)^{-1}} (= k^{k_s q_c m_s d'_c})$ and $X_{sc} \leftarrow X_s^{(t_c)^{-1}} (= k^{d_c m_s})$ to eliminate the random numbers n_c and t_c from the exponent, and then computes $Z_{sc} \leftarrow Y_{sc}(X_{sc})^{-1} (= k^{k_s q_c m_s d'_c - d_c m_s})$. Note that at this point, the exponent of Z_{sc} is $k_s q_c m_s d'_c - d_c m_s$. Since the random number m_s and PH server's secret key k_s are unavailable, the service provider cannot perform offline matching of $d_c m_s$ after obtaining the returned parameters by constructing $m_s q_c d'_c$. However, if the candidate username un' and password pw' are correct, that is, $q'_c \leftarrow H(un' \parallel pw' \parallel r_c) = q_c \leftarrow H(un \parallel pw \parallel r_c)$, $d'_c q_c = k_c q'_c q_c = k_c q_c^2 = d_c$, the exponent of Z_{sc} can be reduced to $d'_c q_c m_s (k_s - 1)$, allowing for the straightforward calculation of recovering $k = Z_{sc}^{(d'_c q'_c w_s)^{-1}}$. The service provider decrypts the x-th file using $F_k(x)$.

4) Key Puncture & Update: The service provider offers password-based encryption and storage. Additionally, encrypted files can be shared online and deleted. Key puncturing is applicable in the following scenarios: (1) Users share the punctured key with specific visitors, thereby limiting their access to certain files; (2) After deleting files, users puncture the corresponding keys to achieve secure deletion. Even if an adversary employs special methods (e.g., hard drive recovery) to restore the file and attempts an online password guess to recover k^*, she will not be able to obtain the punctured key.

KU1: The service provider interacts with the PH server to recover k, following the same steps as outlined in Section III-B3. The user selects the key x or the set S to be punctured, resulting in the punctured key k^*.

If it involves deleting a file, then the service provider selects random numbers from the cyclic group $\mathbb{G}$, denoted as $u'_c, e'_c \leftarrow \$\mathbb{Z}_q$. Then the service provider calculates the hash $q_c \leftarrow H(un \parallel pw \parallel r_c)$ and $d_c \leftarrow q_c k_c$, then calculates $K_c^* \leftarrow (k^*)^{d_c}$ using d_c as the exponent of the punctured key k^*. Afterward, the service provider calculates $E \leftarrow g^{e'_c}$ and $Q'_c \leftarrow (k^*)^{u'_c}$ and encrypt u'_c into $E'_{uc} \leftarrow u'_c \oplus PK_s^{e'_c}$ uses the public key PK_s of the PH server. The service provider sends K_c^* to the PH server.

KU2: The PH server holds its secret and public key pair $\{k_s, PK_s\}$. On receiving K_c^*, the PH server calculates $K_{cs}^* \leftarrow (K_c^*)^{k_s}$ and returns K_{cs}^* to the service provider.

KU3: The service provider stores $T = \{K_{cs}^*, Q_c, E, E_{uc}, r_c\}$ and deletes ciphertexts with punctured keys.

5) Key Rotation: We provide key rotation to address server key leakage. Key rotation initiated by the PH server is global.

KR1: The PH server generates a new private-public key pair $\{k'_s, PK'_s\} \leftarrow \$\Pi.\mathsf{KGen}$ and publishes the public key PK'_s.

KR2: The service provider sends K_{cs} to the PH server.

KR3: The PH server computes $K'_{cs} \leftarrow K_{cs}^{k'_s (k_s)^{-1}}$ and returns K'_{cs} to the service provider. The service provider updates the enrollment record to $T = \{K'_{cs}, Q_c, E, E_{uc}, r_c\}$.

In addition, the service provider can select a new key k'_c, and update K_{cs} to $K'_{cs} \leftarrow K_{cs}^{k'_c (k_c)^{-1}}$ locally. For the *informal security analysis*, please see the Appendix B of the full version available at https://wangdingg.weebly.com/publications.html.

IV. Security Proof

In this section, we formally prove that the proposed protocol satisfies the security properties defined in Section II.

A. Selective PPRF Security

Theorem 1: For any Probabilistic Polynomial-Time (PPT) adversary A making at most q evaluation queries on inputs of length m in the selective puncturing setting, there exists a PRF distinguisher B such that:

$$Adv_{\mathcal{A}}^{PPRF}(\lambda) \leq m \cdot Adv_{B}^{PRF}(\lambda) + O(m/2^{\lambda/2}) + negl(\lambda) \quad (7)$$

where the $O(m/2^{\lambda/2})$ arises from statistical loss due to Leftover Hash Lemma (LHL) [17] on truncations, and is negligible for $\lambda \geq 256$. The negligible term includes any additional statistical losses from the ROM assumption on H_t.

Proof: We use a hybrid argument akin to GGM's tree-based PRF proof [13], where each level is randomized step-by-step.

- **Hybrid$_0$**: The real game with $b = 0$.
- **Hybrid$_1$**: Replace the evaluation of non-punctured paths with a lazy-sampling oracle that computes on-the-fly using real H_t, but for the punctured path to x^*, replace the hash outputs along the path with random values (except the root). This hybrid bounds the impact of puncturing: since puncturing removes subkeys, the adversary cannot compute down the punctured path, and the replacement is indistinguishable if H_t is pseudorandom. Distinguishing Hybrid 0 and 1 reduces to H_t's PRF security. Construct reducer $\mathcal{B}$: $\mathcal{B}$ simulates the tree using its PRF oracle for H_t. For the punctured path, if $\mathcal{A}$ tries to distinguish, $\mathcal{B}$ queries its oracle on the path prefixes and checks if the output looks random. The advantage transition is at most $Adv_{\mathcal{B}}^{PRF}(\lambda)$ per tree level, totaling $m \cdot Adv_{\mathcal{B}}^{PRF}(\lambda)$, where m is the depth of the path to x^*.
- **Hybrid$_2$**: Introduce truncation analysis. In our construction, each H_t output is 2λ bits, truncated to two λ-bit keys. In this hybrid, we model truncation as splitting a random string: if the full output is uniform, each half is uniform over $\{0,1\}^{\lambda}$, with distinguishing advantage bounded by the LHL [17]. Specifically, if the min-entropy of H_t's output is at least $\lambda + \log(1/\epsilon)$, the truncated half has statistical distance ϵ from uniform.
  For each truncation along the path, the advantage loss is $O(1/2^{\lambda/2})$ by LHL [17], since splitting a 2λ-bit random string yields two λ-bit strings each $\sqrt{2^{-\lambda}}$-close to uniform. Over m levels, this adds $O(m/2^{\lambda/2})$ to the total advantage (negligible for $\lambda \geq 256$).
- **Hybrid$_3$**: From Hybrid 2, the punctured path is fully randomized and near-uniform after truncations. Since queries are selectively on $x_i \neq x^*$, their evaluations use non-punctured paths, which remain unchanged. Thus, $F_k(x^*)$ is indistinguishable from random. No $q/2^m$ or $q \cdot Adv_{B}^{PRF}$ term applies, as there is no accidental collision in the selective setting.
  In summary, $Adv_{\mathcal{A}}^{PPRF}(\lambda) \leq m \cdot Adv_{B}^{PRF}(\lambda) + O(m/2^{\lambda/2}) + negl(\lambda)$, which is negligible if H_t is a secure PRF. □

1) Entropy Loss in Hash Iterations and Truncations: Our hash-based PPRF is constructed using a cryptographic hash function $H_t : \{0,1\}^* \to \{0,1\}^{2\lambda}$, where we instantiate H_t with SHA-512 (outputting 512 bits, with $\lambda = 256$ for 256-bit keys). The derivation process involves iterative hashing to generate puncturable keys, with truncations to produce the final output keys of length λ.

Under the Random Oracle Model (ROM) assumption for H_t, the output of each hash iteration is computationally indistinguishable from uniform randomness, provided the input has sufficient min-entropy $H_\infty(X) = -\log_2 \max_x \Pr[X = x]$. The root key k is assumed to be sampled from a high-entropy source with min-entropy $H_\infty(k) \geq 256$ bits. In the ROM, a single application of H_t (SHA-512) preserves min-entropy up to the output length $H_\infty = 512$ bits. Truncation from 512 bits to $\lambda = 256$ bits reduces the maximum possible min-entropy by 256 bits, with a loss of 256 bits in the information-theoretic sense. The loss is irreversible in that discarded bits cannot be recovered, but security holds as long as the remaining entropy meets the target security level.

The root key k must have min-entropy $H_\infty(k) \geq 256$ bits to meet or exceed NIST SP 800-90B [40] recommendations for 256-bit security. For intuitive reference, this equates to a guessing hardness of 2^{256}, far beyond dictionary attacks.

B. Message Hiding

Theorem 2: Let $Adv_{PH-Hi}^{\mathcal{A},\mathbb{O}}$ represents the advantage of the adversary $\mathcal{A}$ breaking *Message Hiding* in PPT. Then, the advantage can be described as:

$$Adv_{PH-Hi}^{\mathcal{A},\mathbb{O}}(\lambda) \leq C' \cdot q^{s'} + negl(\lambda) \quad (8)$$

where C' and s' are the linear regression parameters of the password distribution $\mathbb{D}$ [7], q is the number of decryption oracle executions.

Proof: We formalize four games to prove the above theorem, which infers the advantage of adversary $\mathcal{A}$ distinguishing the bit b from $(0,1)$ in $Adv_{PH-Hi}^{\mathcal{A},\mathbb{O}}$. The idea of proof is to reveal $\mathcal{A}$ is indistinguishable from the enrollment record T for (m_0, m_1).

- **Game$_0$**: Which is equivalent to the predefined game called PH-Hi. The details and formal definition of this game can be found in section II-C.
- **Game$_1$**: Which represents a transformation of **Game$_0$** where the challenger $\mathcal{R}_s$ simulates an oracle. In the enrollment record, the root key k (recorded as m_b in this game) is stored in the form of $K_{cs} = k^{d_c k_s}$ and $Q_c = k^{u_c q_c}$, where $d_c = q_c^2 k_c$, k_s is the PH server's secret key, k_c is the service provider's secret key, u_c is a random number encrypted by PH server's public key, and $q_c = H(un \parallel pw \parallel r_c)$. As the challenger interacting with $\mathcal{A}$, $\mathcal{R}_s$ simulates the outputs Y_s and X_s. If the received X_c and Y_c are based on m_0 or m_1, or if the request is a duplicate, $\mathcal{R}_s$ outputs a random value. The statement that this game is identical to the above game indicates that the transformation from **Game$_0$** to **Game$_1$** does not change the fundamental properties of the game. The probabilities of the adversary's success in both games are the same, meaning that any advantage the adversary has in **Game$_0$** is preserved in **Game$_1$**.
- **Game$_2$**: After obtaining the enrollment record T, the message m_b in the record is stored by the PH server's key and a nonce. The formats are $K_{cs} = m_b^{d_c k_s}$ and $Q_c = m_b^{u_c q_c}$, allowing the adversary to eventually recover $m_b^{q_c^2 k_s}$

and $m_b^{u_c q_c}$. However, since k_s, u_c and q_c are unknown, the adversary cannot determine m_0 or m_1. At this point, the advantage of directly guessing b can be described as $Adv_{\mathcal{A}}^{\text{PH-Hi}}(\lambda) = |\Pr[b' = b] - \frac{1}{2}|$.

- **Game$_3$**: Since the adversary possesses a password distribution $\mathbb{D}$, she can execute the decryption protocol to interact with the PH server, launching sustained online password-guessing attacks. If the adversary successfully guesses the password, they can leverage the decryption protocol to gain an advantage in determining b. The probability of success for online guessing is $C' \cdot q^{s'}$, where C' and s' are the linear regression parameters of the password distribution $\mathbb{D}$ [7], q is the number of decryption oracle executions.
  Therefore, the probability of $\mathcal{A}$ successfully distinguishing in **Game$_3$** is bounded by the adversary's advantage in guessing passwords, denoted as $C' \cdot q^{s'}$.
  The conclusion ties back to the overall security goal: ensuring that the adversary's ability to distinguish the bit $b \in (0, 1)$ (indicating which message is encrypted in the enrollment record) is negligible. This is formally proven in theorem IV-A1. The negligibility of the advantage means that the probability of the adversary successfully distinguishing the bit is so small that it is considered insignificant for practical purposes, thereby affirming the protocol's security.

□

C. Partial Oblivious

Theorem 3: Let $Adv_{PH-Ob}^{\mathcal{A},\mathbb{O}}(\lambda)$ represents the advantage of the adversary $\mathcal{A}$ breaking PH-Ob in PPT. Then, the advantage can be described as:

$$Adv_{PH-Ob}^{\mathcal{A},\mathbb{O}}(\lambda) \leq Adv_{\mathcal{A}}^{DL-base} + negl(\lambda) \tag{9}$$

where $Adv_{\mathcal{A}}^{DL-base}$ quantifies the adversary's advantage in solving for the base g when given the result $y = g^x$ but neither the exponent x nor the base g. This problem is computationally harder than the standard DL problem, which requires solving for x given g and y. $negl(\lambda)$ is a negligible function.

Proof: We formalize four games to prove the above theorem, which infers the advantage of adversary $\mathcal{A}$ distinguishing the bit b from $(0, 1)$ in $Adv_{PH-Ob}^{\mathcal{A},\mathbb{O}}$.

- **Game$_0$**: Which is equivalent to the predefined game called PH-Ob. The details and formal definition of this game can be found in section II-C.
- **Game$_1$**: Which represents a transformation of **Game$_0$** where the challenger $\mathcal{R}_c$ simulates an oracle. In the enrollment record, the root key k (recorded as m_b in this game) is stored in the form of $K_{cs} = k^{d_c k_s}$ and $Q_c = k^{u_c q_c}$, where $d_c = q_c^2 k_c$, k_s is the PH server's secret key, k_c is the service provider's secret key, u_c is a random number encrypted by PH server's public key, and $q_c = H(un \parallel pw_b \parallel r_c)$. As the challenger $\mathcal{R}_c$ interacting with $\mathcal{A}$, $\mathcal{R}_c$ simulates the outputs K_{cs} and Q_c. If the received requests are duplicate or based on (pw_0, pw_1), $\mathcal{R}_c$ outputs a random value. The statement that this game is identical to the above game indicates that the transformation from **Game$_0$** to **Game$_1$** does not change the fundamental properties of the game. The probabilities of the adversary's success in both games are the same, meaning that any advantage the adversary has in **Game$_0$** is preserved in **Game$_1$**.
- **Game$_2$**: After obtaining the enrollment record T, the message m_b and the password pw_b in the record are stored by the service provider's key and nonce. The formats are $K_{cs} = m_b^{d_c k_s t_c}$ and $Q_c = m_b^{u_c q_c n_c d'_c}$, allowing the adversary to eventually recover $m_b^{d_c t_c}$ and $m_b^{q_c n_c d'_c}$. However, since k_c, q_c, q'_c, and n_c are unknown, the adversary cannot determine pw_0, m_0 or pw_1, m_1. At this point, the advantage of directly guessing b can be described as $Adv_{\mathcal{A}}^{PH-Ob}(\lambda) = |\Pr[b' = b] - \frac{1}{2}|$.
- **Game$_3$**: Which continues the security proof by considering a scenario where the adversary $\mathcal{A}$ has compromised the PH server's secret key k_s. Despite this compromise, recovering the value m_b from the expression $m_b^{d_c t_c}$ or $m_b^{q_c n_c d'_c}$ is challenging without knowing the exponents. The game emphasizes that recovering m_b is much harder than solving the Discrete Logarithm (DL) problem. Unless the adversary can recover the base without knowing the exponent, with the advantage denoted as $Adv_{\mathcal{A}}^{DL-base}$, it cannot gain an edge in breaking PH-Ob. The negligibility of the advantage means that the probability of the adversary successfully distinguishing the bit is so small that it is considered insignificant for practical purposes, thereby affirming the protocol's security.

□

D. Soundness

Theorem 4: Let $Adv_{PH-Sd}^{\mathcal{A},\mathbb{O}}(\lambda)$ represents the advantage of the adversary $\mathcal{A}$ breaking PH-Sd in PPT. Then, the advantage can be described as:

$$Adv_{PH-Sd}^{\mathcal{A},\mathbb{O}}(\lambda) \leq Adv_{\mathcal{A}}^{DL-base} + negl(\lambda) \tag{10}$$

where $Adv_{\mathcal{A}}^{DL-base}$ quantifies the adversary's advantage in solving for the base g when given the result $y = g^x$ but neither the exponent x nor the base g. This problem is computationally harder than the standard DL problem, which requires solving for x given g and y. $negl(\lambda)$ is a negligible function.

E. Forward Security

Theorem 5: Let $Adv_{PH-FS\,1,2}^{\mathcal{A},\mathbb{O}}(\lambda)$ represents the advantage of the adversary $\mathcal{A}$ breaking PH-FS in PPT. Then, the advantage can be described as:

$$Adv_{PH-FS\,1,2}^{\mathcal{A},\mathbb{O}}(\lambda) \leq Adv_{\mathcal{A}}^{DL-base} + negl(\lambda) \tag{11}$$

where $Adv_{\mathcal{A}}^{DL-base}$ quantifies the adversary's advantage in solving for the base g when given the result $y = g^x$ but neither the exponent x nor the base g. This problem is computationally harder than the standard Discrete Logarithm (DL) problem, which requires solving for x given g and y. $negl(\lambda)$ is a negligible function. The proofs of *Soundness* and *Forward Security* follow a structure similar to that of *Partial Obliviousness*, please see the Appendix C and D of the full version available at https://wangdingg.weebly.com/publications.html.

F. (Multi-Query) Privacy

Theorem 6: Let $Adv^{\mathcal{A},\mathbb{O}}_{PH-MPr}(\lambda)$ represents the advantage of any adversary $\mathcal{A}$ breaking PH-Pr in PPT. Then, the advantage can be described as:

$$Adv^{\mathcal{A},\mathbb{O}}_{PH-MPr}(\lambda) \leq q(\lambda) Adv^{DL-base}_{\mathcal{A}} + negl(\lambda) \tag{12}$$

where $Adv^{DL-base}_{\mathcal{A}}$ quantifies the adversary's advantage in solving for the base g when given the result $y = g^x$ but neither the exponent x nor the base g. This problem is computationally harder than the standard Discrete Logarithm (DL) problem, which requires solving for x given g and y. $\mathcal{A}$ makes at most $q(\lambda)$ decryption queries. $negl(\lambda)$ is a negligible function.

Proof: We formalize a sequence of games to prove the above theorem, which bounds the advantage of adversary $\mathcal{A}$ in distinguishing the bit b_j. The proof uses a hybrid argument to handle the multi-query setting: we gradually replace the real decryption interactions with simulated ones, ensuring that $\mathcal{A}$ cannot distinguish whether each query uses the correct or incorrect credentials without solving a hard DL-base instance. Let $\Pr[G_i]$ denote the probability that $\mathcal{A}$ wins in Game i.

- **Game$_0$**: This is equivalent to the predefined game PH-MPr. The details and formal definition of this game can be found in Section II-C.
- **Game$_1$**: This modifies **Game$_0$** by having the challenger $\mathcal{R}$ simulate the oracle for all decryption queries. In the enrollment phase, the root key k is stored in the record T as $K_{cs} = k^{d_c k_s}$ and $Q_c = m^{u_c q_c}$. During decryption queries, for each query i, $\mathcal{R}$ computes the parameters using temporary random exponents t_c and n_c (per query), outputting $X_c = K_{cs}^{t_c}$ and $Y_c = Q_c^{n_c d'_c}$ if the query is non-duplicate. This simulation is indistinguishable from **Game$_0$** under the random oracle model for H, as duplicates are rare (probability $\leq 1/2^\lambda$ per query) and the exponents hide the underlying hashes. The transition advantage is negligible:$|\Pr[G_1] - \Pr[G_0]| \leq \mathsf{negl}(\lambda)$.
- **Game$_{2.0}$ to $_{2.q}$**: To handle multiple queries, we introduce $q+1$ games. In **Game$_{2.\ell}$** (for $\ell = 0$ to q), the first ℓ decryption queries are "randomized" (i.e., the challenger forces $b_i = 1$ for $i \leq \ell$, using incorrect credentials, and simulates responses as random elements in $\mathbb{G}$ indistinguishable from real ones), while queries $i > \ell$ remain as in **Game$_1$**. **Game$_{2.0}$** is identical to **Game$_1$**, and **Game$_{2.q}$** has all queries randomized. For each transition from **Game$_{2.\ell-1}$** to **Game$_{2.\ell}$**, the difference arises only in query ℓ: in **Game$_{2.\ell-1}$**, it uses the real b_ℓ; in **Game$_{2.\ell}$**, it forces incorrect credentials and simulates random parameters. Distinguishing this requires $\mathcal{A}$ to detect whether the parameters for query ℓ hide the correct q_c or incorrect q'_c in the exponents, without knowing k_s, t_c, n_c. After all queries, if $\mathcal{A}$ chooses $j \neq \ell$, the hybrids are identical; if $j = \ell$, distinguishing reduces to solving the DL-base problem. By a hybrid lemma, the total transition bound is:$|\Pr[G_{2.q}] - \Pr[G_{2.0}]| \leq q(\lambda) \cdot \mathsf{Adv}^{DL-base}_{\mathcal{A}}$. In **Game$_{2.q}$**, since all responses are random and independent of the b_i, $\mathcal{A}$'s guess for any j is random: $\Pr[G_{2.q}] = 1/2$.

TABLE II
EXECUTION TIME OF BASIC OPERATIONS

Operation	Time (μs)	Operation	Time (μs)
$Hash_{(SHA-256)}$	28.57	$Enc_{(AES-256)}$	147.6
$Hash_{(SHA-512)}$	32.39	$Dec_{(AES-256)}$	167.8
$Mul_{(256*256bit)}$	1.35	$Mul_{(NIST-P256)}$	86.37
$Sig_{(RSA-1024)}$	872.7	$Ver_{(RSA-1024)}$	50.78

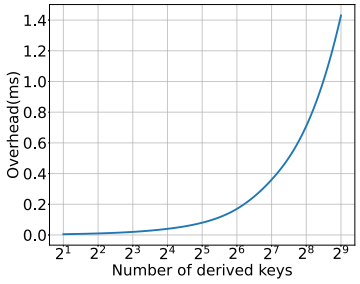


Fig. 5. The overhead of PPRF-derived keys. The cost of generating 2^n keys is $2^n - 1$ hash executions.

- **Game$_3$**: This extends **Game$_{2.q}$** by considering a compromise of the PH server's secret key k_s after all queries. Even with k_s, recovering or distinguishing the bases/exponents in the simulated parameters remains hard without knowing the per-query random t_c, n_c, d'_c, reducing again to DL-base. The advantage is bounded by $\mathsf{Adv}^{DL-base}_{\mathcal{A}}$, but since all queries are already randomized, the transition is at most negligible:$|\Pr[G_3] - \Pr[G_{2.q}]| \leq \mathsf{negl}(\lambda)$.

In Game 3, $\Pr[G_3] = 1/2$. Summing the bounds, we have:

$$\mathsf{Adv}^{PH-MPr}_{\mathcal{A},\mathcal{O}}(\lambda) \leq q(\lambda) \cdot \mathsf{Adv}^{DL-base}_{\mathcal{A}} + \mathsf{negl}(\lambda).$$

which is negligible if the DL-base problem is hard in $\mathbb{G}$. □

V. IMPLEMENTATIONS AND EVALUATIONS

A. Experimental Setup

We use two devices executed on Ubuntu (Version: 22.04.3 LTS, Intel Xeon Gold 6320R @2.10GHz, 256GB of RAM, NVIDIA RTX 3090 GPU) to separately deploy the service provider and the PH server. The communication code is written in Python, and WebSocket (Version: 15.0.1) [41] is utilized for communication between the service provider (referred to as client C) and the PH server (referred to as server S). WebSocket is a protocol that enables full-duplex communication over a single TCP connection. Additionally, WebSocket supports TLS, which can be enabled for secure connections by using the "wss://" prefix. WebSocket establishes a connection with a single handshake and maintains an open state afterward, eliminating the need for additional "Keep-Alive" headers. This makes WebSocket particularly well-suited for scenarios that require high real-time performance and frequent communication over short intervals. We measured the communication latency between the service provider and the PH server to be approximately 1 *ms* in a Local Area Network (LAN) and 90 *ms* over the Wide Area Network (WAN).

We implemented the PHE protocol using Python, with the hashlib library providing the SHA-256 algorithm for hashing and the OpenSSL library providing large integer operations, and the group based on NIST-P256. The construction of PPRF

TABLE III
COMPARISON OF OVERHEAD AMONG PHE SCHEMES

Scheme		Enrollment/Encryption (ms)	Decryption (ms)	Storage	
				PH server	Service provider
APHE [14]	2024	$4H_{ZQ}+2Enc+Ver+Sig(1.37\pm0.05)$	$7H_{ZQ}+2Dec+Ver+Sig+5E_G(1.98\pm0.04)$	L_{ZQ}	$5L_{ZQ}$
(t,m)-PHE [6]	2020	$(5+5p)H_{ZQ}+(2+11p)E_G+pPoK(129.01\pm0.45)$	$(7+20p)H_{ZQ}+(19+59p)E_G+pPoK(135.92\pm0.51)$	L_{ZQ}	$L_{ZQ}+Enc(2L_G)$
PHE [23]	2018	$6H_{ZG}+5E_G+PoK(128.41\pm0.30)$	$6H_{ZG}+6E_G+PoK(129.15\pm0.24)$	$(2t+1)L_{ZQ}$	$3L_{ZQ}+2L_G$
Our PHE		$H_{ZQ}+5E_G(0.46\pm0.03)$	$H_{ZQ}+9E_G(0.81\pm0.07)$	L_{ZQ}	$6L_{ZQ}$

H_{ZQ}: Hash from $\{0,1\}^*$ to $\mathbb{Z}_q$. H_{ZG}: Hash from $\{0,1\}^*$ to $\mathbb{G}$. PoK: Proof and verification of NIZKPoK. t: Number of record.
H_{ZN}: Hash from $\{0,1\}^*$ to $\mathbb{Z}_n$. L_{ZQ}: Length of H_{ZQ}. L_G: Length of the element in $\mathbb{G}$. Sig/Ver: Signature/verification.
p: Servers, $t \le p \le m$, here $p=1$. Enc/Dec: Encryption/Decryption. E_G: Exponentiation in $\mathbb{G}$.

is based on SHA-512. The generation function of PPRF returns 256-bit keys. Symmetric encryption is based on AES-256 under the pycryptodome (Version: 3.23.0) library. We measure the overhead of the basic operations involved in the scheme, which is provided in Table II for reference. The source code of HPHE is available at https://github.com/dingzixuan8899.

B. Performance of HPHE

1) Setup Phase: During the setup phase, the protocol constructs a hash function and PPRF. The PH server generates a public-private key pair, while the service provider generates a private key. The overhead for key generation is 0.0871 *ms*.

2) Enrollment/Encryption Phase: During the enrollment phase, the service provider selects three random numbers and a root key, performs one hash operation, two multiplications, and four point multiplication operations.

The measured average overhead is 0.351 *ms*. The PH server performs one point multiplication, and the overhead is 0.0863 *ms*. The number of keys generated by the PPRF, as summarized in section III-B.1, produces 2^n keys after n iterations, requiring $2^n - 1$ hash executions. The cost of key generation is shown in Figure 5.

3) Decryption Phase: In the decryption phase, the service provider generates two random numbers and calculates a hash based on the candidate username and password. It then performs two point multiplication operations. After receiving the parameters returned by the PH server, the service provider calculates four point multiplication operations, the total overhead is 0.521 *ms*. The PH server selects a random number, performs one multiplication and two point multiplication operations. The total overhead is 0.179 *ms*.

4) Key Rotation Phase: In the key rotation phase, the PH server generates a new key pair, and calculates one point multiplication operation, measured overhead is 0.174 *ms*.

C. Comparison Among Relevant Schemes

Table III lists the calculations involved in the relevant PHE schemes (APHE [14], (t,m)-PHE [6], and PHE [23]) and provides reference values under the same hardware conditions. In APHE [14], the service provider verifies the signature of the PH server, then multiplies the message with the hash value and encrypts the result based on the secret key. Here, we use RSA-1024 for signature simulation and AES-256 for symmetric encryption. (t,m)-PHE [6] is a threshold PHE scheme, where we set the threshold value to 1 to avoid excessive unfair overhead from multiple threshold servers. In the encryption/enrollment phase, the service provider interacts with the PH server for 3 messages. In the decryption phase, the interaction extends to 3 rounds. Throughout the interaction, the service provider iteratively undergoes Shamir key recovery and navigates authentication with the PH server.

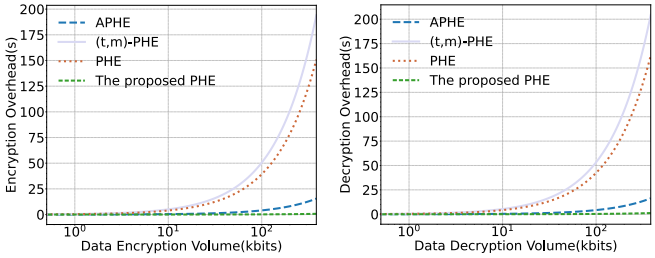
TABLE IV
LATENCY AND THROUGHOUT COMPARISON

Scheme		Latency (LAN)	Latency (WAN)	Throughput (req/s)
APHE [14]	Encryption	2.39±0.12ms	94.74±2.18ms	1070±45
	Decryption	3.31±0.15ms	90.21±2.40ms	871±25
(t,m)-PHE [6]	Encryption	131.12±4.85ms	220.14±5.71ms	500±10
	Decryption	139.75±2.14ms	227.31±4.12ms	469±13
PHE [23]	Encryption	131.02±2.63ms	220.17±3.19ms	520±22
	Decryption	130.87±2.19ms	215.14±2.97ms	500±20
Our PHE	Encryption	1.39±0.07ms	90.11±0.85ms	11522±75
	Decryption	1.91±0.04ms	89.47±0.79ms	5537±43

In PHE [23], the service provider takes the message as the base, the key as the exponent, and multiplies it by the hash value. During the interaction, the PH server proves the result's honesty to the service provider using a non-interactive zero-knowledge proof of knowledge (NIZKPoK), based on the PoK operations and constructions from CCS' 18 [20].

In the case of Encrypting/Decrypting 1024 files in a one-data-one-key manner, the PHE schemes here encrypt/decrypt only the keys rather than the data, and the overhead of symmetric encryption/decryption is ignored. APHE [14] requires 1.37s/2.38s, PHE [23] requires 15.17s/15.28s, (t,m)-PHE [6] requires 16.25s/16.98s, while our PHE requires only 33.59ms/33.94ms, achieving approximately a 450× speedup compared to the original PHE [23] and a 40× speedup over APHE [14]. Focusing only on the core interactive phase, our PHE scheme is at least 61% faster than the most efficient previous scheme APHE [14].

Under the given hardware and network environment, we test the latency of the proposed protocol. In the LAN environment, the delay for the enrollment/encryption phase, from the user entering a password to completing enrollment, is approximately 1.453 *ms*. In the WAN environment, the delay is around 87.64 *ms*. In the decryption phase within the LAN environment, the latency is approximately 1.71 *ms*. In the WAN environment, the latency is around 88.65 *ms*. As a comparison, we programmed the relevant PHE schemes (APHE [14], (t,m)-PHE ($p=1$) [6], and PHE [23]) in Python to test under the same conditions and measured the latency, with the results shown in Table IV.



(a) Overhead of Encryption (b) Overhead of Decryption

Fig. 6. Comparison of encryption/decryption overheads between PHE schemes (APHE [14], (t,m)-PHE [6], and PHE [23]). The original PHE schemes are used for data encryption/decryption, with a single data volume of 256 bits.

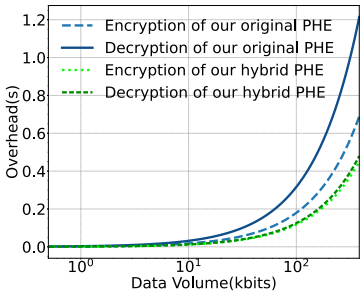


Fig. 7. The overhead comparison of our original and hybrid PHE schemes. Specifically, the main key is encrypted with PHE, and then symmetric encryption with PPRF-derived keys is applied to the data.

We use Netperf to test the throughput of the PH server under large-scale concurrent requests. Netperf allows pre-filling data into the send buffer from a specified file, with the file's contents serving as the data to be sent. This enables us to customize the requests in compliance with protocol specifications. Through measurements, our proposed solution achieved a throughput of approximately 11,000-12,000 requests per second during the enrollment/encryption phase and about 5,500-5,600 requests per second during the decryption phase. We additionally tested related schemes, and average throughput results are shown in Table IV.

D. Further Analysis of Encryption/Decryption

As shown in Table III, although the overhead of PHE schemes is limited to 200 *ms*, the message size for a single encryption is constrained due to limitations in protocol design (e.g., in PHE [23], the message M is drawn from the q-order finite multiplicative cyclic group $\mathbb{G}$).

Given WAN latency, the encryption and decryption rate of PHE is approximately *1120-2870 bits per second*. PHE is clearly unsuitable for directly encrypting large-scale data. As shown in Figure 6, we present the overhead of the original PHE schemes when handling encryption and decryption of large-scale data. Furthermore, original PHE schemes lack mechanisms for ciphertext management. Therefore, this work combines PHE and PPRF to achieve efficient encryption/decryption and ciphertext management. Figure 7 shows a comparison of the efficiency of our hybrid PHE scheme versus our original PHE scheme.

The ciphertext strategy proposed in this work addresses both the efficiency limitations of large-scale encryption and decryption and provides a reliable ciphertext management solution, making it compatible with related PHE schemes. In the proposed strategy, PHE does not directly encrypt the message but rather encrypts a root key. The root key can derive multiple sub-keys via a PPRF function, which can then be punctured to securely delete corresponding ciphertexts or restrict access. The symmetric encryption/decryption rate based on derived keys is approximately *870k bits per second*. Under the same hardware conditions and with no network latency, the current PHE encryption/decryption efficiency is approximately *1.9k to 19.1k bits per second*.

VI. Conclusion

This work proposes a hybrid PHE architecture *for the first time*, which improves the efficiency of large-scale data encryption/decryption by approximately 450 times compared to original PHE schemes. The proposed hash-based PPRF enables puncturable ciphertext management, allowing secure key deletion and restricted access control. We formalize *Privacy* as a concrete security property *for the first time*. The proposed HPHE addresses the issue of authentication result leakage that previous PHE schemes faced. Deployment experiments show that our proposed architecture is feasible in terms of efficiency and scalability, and it can be extended to existing PHE schemes. Particularly, the one-data-one-key approach and puncturable keys provide secure ciphertext management and sharing. This work enhances the privacy, extends it to large-scale data encryption, and significantly improves encryption/decryption efficiency.

References

[1] M. Abdalla, F. Benhamouda, and P. MacKenzie, "Security of the J-PAKE password-authenticated key exchange protocol," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 571–587.

[2] M. Backendal, H. Davis, F. Günther, M. Haller, and K. G. Paterson, "A formal treatment of end-to-end encrypted cloud storage," in *Proc. CRYPTO*, 2024, pp. 40–74.

[3] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," *ACM Trans. Comput. Syst.*, vol. 33, no. 3, pp. 1–26, Sep. 2015.

[4] M. S. Bohuk, M. Islam, S. Ahmad, M. Swift, T. Ristenpart, and R. Chatterjee, "Gossamer: Securely measuring password-based logins," in *Proc. USENIX SEC*, 2022, pp. 1867–1884.

[5] D. Boneh and B. Waters, "Constrained pseudorandom functions and their applications," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Security (ASIACRYPT)*, Apr. 2013, pp. 280–300.

[6] J. Brost, C. Egger, R. W. F. Lai, F. Schmid, D. Schröder, and M. Zoppelt, "Threshold password-hardened encryption services," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 409–424.

[7] W. Ding, C. Haibo, W. Ping, H. Xinyi, and J. Gaopeng, "Zipf's law in passwords," *IEEE Trans. Inform. Foren. Secur.*, vol. 12, no. 11, pp. 2776–2791, Nov. 2017.

[8] W. Ding, S. Xuan, D. Qiying, S. Yaosheng, and J. Chunfu, "No single silver bullet: Measuring the accuracy of password strength meters," in *Proc. USENIX SEC*, Apr. 2023, pp. 947–964.

[9] (Feb. 2025). *Global IoT Data Leak Exposes 2.7 Billion Records and Wi-Fi Passwords Worldwide*. [Online]. Available: https://gbhackers.com/global-iot-data-leak-exposes-2-7-billion-records/

[10] P.-A. Dupont, J. Hesse, D. Pointcheval, L. Reyzin, and S. Yakoubov, "Fuzzy password-authenticated key exchange," in *Proc. EUROCRYPT*, Apr. 2018, pp. 393–424.

[11] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. IT-31, no. 4, pp. 469–472, Jul. 1985.

[12] A. Everspaugh, R. Chaterjee, S. Scott, A. Juels, and T. Ristenpart, "The pythia PRF service," in *Proc. USENIX Secur. Symp.*, 2015, pp. 547–562.

[13] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. ACM*, vol. 33, no. 4, pp. 792–807, Aug. 1986.
[14] G. Ha, C. Jia, X. Ge, J. Yuan, H. Chen, and M. Li, "Efficient and anonymous password-hardened encryption services," *Inf. Sci.*, vol. 653, Jan. 2024, Art. no. 119771.
[15] S. Halevi and H. Krawczyk, "Public-key cryptography and password protocols," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 3, pp. 230–268, Aug. 1999.
[16] M. X. Heiligenstein. (Oct. 2023). *Twitter Data Breaches: Full Timeline Through 2023*. [Online]. Available: https://firewalltimes.com/twitter-data-breach-timeline/
[17] R. Impagliazzo and D. Zuckerman, "How to recycle random bits," in *Proc. FOCS*, vol. 30, Feb. 1989, pp. 248–253.
[18] S. Jarecki, H. Krawczyk, and J. Xu, "OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks," in *Proc. EUROCRYPT*, 2018, pp. 456–486.
[19] C. Jia, S. Wu, and D. Wang, "Reliable password hardening service with opt-out," in *Proc. 41st Int. Symp. Reliable Distrib. Syst. (SRDS)*, Sep. 2022, pp. 250–261.
[20] J. Katz, V. Kolesnikov, and X. Wang, "Improved non-interactive zero knowledge with applications to post-quantum signatures," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 525–537.
[21] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias, "Delegatable pseudorandom functions and applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 669–684.
[22] K. Kosling. (Jul. 2024). *Rockyou2024: Nearly 10 Billion Unique Plaintext Passwords Leaked*. [Online]. Available: https://cybernews.com/security/rockyou2024-largest-password-compilation-leak/
[23] R. W. F. Lai, C. Egger, M. Reinert, S. S. M. Chow, M. Maffei, and D. Schröder, "Simple password-hardened encryption services," in *Proc. USENIX SEC*, May 2018, pp. 1405–1421.
[24] R. W. F. Lai, C. Egger, D. Schröder, and S. S. M. Chow, "Phoenix: Rebirth of a cryptographic password-hardening service," in *Proc. USENIX SEC*, 2017, pp. 899–916.
[25] V. Lozupone, "Analyze encryption and public key infrastructure (PKI)," *Int. J. Inf. Manage.*, vol. 38, no. 1, pp. 42–44, Feb. 2018.
[26] R. McLean. (Jul. 2019). *A Hacker Gained Access to 100 Million Capital One Credit Card Applications and Accounts*. [Online]. Available: https://edition.cnn.com/2019/07/29/business/capital-one-data-breach/index.html
[27] Microsoft. (Jul. 2025). *Transport Layer Security (TLS) Registry Settings*. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings?tabs=diffie-hellman
[28] S. Oesch and S. Ruoti, "That was then, this is now: A security evaluation of password generation, storage, and autofill in browser-based password managers," in *Proc. USENIX SEC*, 2019, pp. 2165–2182.
[29] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Proc. Annu. Int. Cryptol. Conf.*, 1991, pp. 129–140.
[30] M. Peter, W. M. Collins, L. M. Michelle, and J. A. Adam, "Why users (don't) use password managers at a large educational institution," in *Proc. USENIX SEC*, 2022, pp. 1849–1866.
[31] V. Petkauskas. (2024). *Mother of All Breaches Reveals 26 Billion Records: What we Know so Far*. [Online]. Available: https://cybernews.com/security/billions-passwords-credentials-leaked-mother-of-all-breaches/
[32] Q. Wang and D. Wang, "Understanding failures in security proofs of multi-factor authentication for mobile devices," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 597–612, 2023.
[33] E. Rescorla. (Aug. 2018). *The Transport Layer Security (TLS) Protocol Version 1.3*. [Online]. Available: https://www.rfc-editor.org/info/rfc8446
[34] S. Sahin, S. A. Roomi, T. Poteat, and F. Li, "Investigating the password policy practices of website administrators," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2023, pp. 552–569.
[35] J. Schneider, N. Fleischhacker, D. Schröder, and M. Backes, "Efficient cryptographic password hardening services from partially oblivious commitments," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1192–1203.
[36] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
[37] M. Shirvanian, N. Saxena, S. Jarecki, and H. Krawczyk, "Building and studying a password store that perfectly hides passwords from itself," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 5, pp. 770–782, Sep. 2019.
[38] Y. Song, C. Xu, Y. Zhang, and S. Li, "Hardening password-based credential databases," *IEEE Trans. Inform. Foren. Security*, vol. 19, pp. 469–484, 2023.
[39] T. Team. (2024). *What Happened in the Canva Data Breach?*. [Online]. Available: https://www.twingate.com/blog/tips/canva-data-breach
[40] M. S. Turan, E. Barker, J. Kelsey, K. McKay, M. Baish, and M. Boyle. (Jan. 2018). *Recommendation for the Entropy Sources Used for Random Bit Generation*. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/90/b/final
[41] (2024). *Websocket*. [Online]. Available: https://developer.mozilla.org/en-U.S./docs/Web/API/WebSocket
[42] Y. Zhang, C. Xu, N. Cheng, and X. Shen, "Secure password-protected encryption key for deduplicated cloud storage systems," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 4, pp. 2789–2806, Jul. 2022.
[43] Z. Zhang, Y. Wang, and K. Yang, "Strong authentication without temper-resistant hardware and application to federated identities," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, Sep. 2020, pp. 1–15.

Zixuan Ding is currently pursuing the Ph.D. degree with the College of Cryptology and Cyber Science, Nankai University, Tianjin, China. He has published articles at venues, such as IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, IEEE INTERNET OF THINGS JOURNAL, and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY. His research interests include applied cryptography, authentication protocols, and password-based authentication.

Ding Wang received the Ph.D. degree in information security from Peking University in 2017. He was supported by the "Boya Postdoctoral Fellowship" at Peking University from 2017 to 2019. Currently, he is a Full Professor with Nankai University. As the first author (or the corresponding author), he has published more than 100 papers at venues, such as IEEE S&P, ACM CCS, NDSS, Usenix Security, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, and IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY. His research has been reported by over 200 media like The Wall Street Journal, Daily Mail, Forbes, IEEE Spectrum, and Communications of the ACM, and resulted in the revision of the authentication guideline NIST SP800-63-2. He has been involved in the community as the PC Chair/a TPC Member for over 60 international conferences, such as NDSS 2023–2025, ACM CCS 2022, PETS 2022–2024, ACSAC 2020–2024, RAID 2024/2023, IEEE EuroS&P 2025, FC 2026/2025, ACM AsiaCCS 2025/2022/2021, ICICS 2018–2025, and SPNCE 2020–2022. He has received the ACM China Outstanding Doctoral Dissertation Award, the Best Paper Award at INSCRYPT 2018, the First Prize of Cryptologic Innovation Award of the China Association for Cryptologic Research, and the First Prize of Natural Science Award of the Ministry of Education. His main research interests focus on passwords, authentication, and provable security.