Secure Password-Protected Encryption Key for Deduplicated Cloud Storage Systems

Yuan Zhang, *Member, IEEE*, Chunxiang Xu, *Member, IEEE*, Nan Cheng, *Member, IEEE*, and Xuemin Shen, *Fellow, IEEE*

Abstract—In this article, we propose SPADE, an encrypted data deduplication scheme that resists compromised key servers and frees users from the key management problem. Specifically, we propose a proactivization mechanism for the servers-aided message-locked encryption (MLE) to periodically substitute key servers with newly employed ones, which renews the security protection and retains encrypted data deduplication. We present a servers-aided password-hardening protocol to resist dictionary guessing attacks. Based on the protocol, we further propose a password-based layered encryption mechanism and a password-based authentication mechanism and integrate them into SPADE to enable users to access their data only using their passwords. Provable security and high efficiency of SPADE are demonstrated by comprehensive analyses and experimental evaluations.

Index Terms—Message-locked encryption, brute-force attacks, password-hardening protocol, password-based layered encryption, dictionary guessing attacks, password-based authentication

1 Introduction

In the big data era, the volume of digital data increases explosively. A recent report[1] indicates that the data we create and copy are doubling in size every two years, and will reach 175 zettabytes by 2025. As the amount of data has increased exponentially, users suffer from critical problems in data management [2], [3], [4]. With the significant development of cloud storage, people are increasingly outsourcing their data to cloud servers, which enables them to efficiently manage their data without deploying infrastructures and maintaining local devices [5], [6].

Commercial cloud service providers always perform data deduplication across their users to save storage space significantly. Recent literature [7], [8] has demonstrated that such a strategy can save space by more than 65 percent in electronic health systems and 90 percent in backup systems. While storage costs can be reduced by data deduplication, the outsourced data are confronted with critical security issues. One of the most important concerns is data confidentiality [9], [10], [11]. From the data owners' perspective, the contents of outsourced data may contain their privacy information [12], [13]. As such, the data are always encrypted by using conventional encryption algorithms before outsourcing. However, because of the randomness of the encryption (i.e., different users would output different ciphertexts for the same data), deduplication is impeded.

Message-locked encryption (MLE) is a special type of symmetric encryption, in which the MLE key (i.e., the encryption and decryption key) is derived from the plaintext itself [14], [15]. This enables different users to output the same ciphertext for the same plaintext and allows the cloud server to perform deduplication over encrypted data across all its users. Whereas, MLE is inherently vulnerable to brute-force dictionary attacks [16]: given a ciphertext, an adversary (e.g., an adversarial cloud server) can encrypt all plaintext candidates by using MLE and identify the matched ciphertext to recover the content. The most practical and affordable way to resist such a brute-force attack is the server-aided MLE [16], i.e., an independent key server which holds a server-side secret is introduced to assist users in generating MLE keys. In such a scheme, an MLE key is derived from the plaintext itself and the server-side secret, a user requests an MLE key from the key server in an oblivious manner [17], such that she/he can obtain the MLE key without revealing any information about the plaintext to the key server. We note that resistance against brute-force attacks has become a basic security requirement and such a "server-aided mechanism" has become a general architecture for encrypted data deduplication schemes.

Despite the advantages of server-aided MLE, it is also confronted with security and efficiency issues when integrated into cloud storage systems.

From the perspective of security, the server-aided MLE bears a strong assumption that its security against the

1. How much data is there in the world? https://www.bernardmarr.com/default.asp?contentID=1846

- *Yuan Zhang is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China. E-mail: ZY_LoYe@126.com.*
- *Chunxiang Xu is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China, and also with the Key Laboratory of Financial Mathematics, Fujian Province University, Putian University, Putian, Fujian 351100, China. E-mail: chxxu@uestc.edu.cn.*
- *Nan Cheng is with the School of Telecommunication, Xidian University, Xi'an 710071, China. E-mail: dr.nan.cheng@ieee.org.*
- *Xuemin Shen is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada. E-mail: sshen@uwaterloo.ca.*

Manuscript received 19 June 2020; revised 6 Apr. 2021; accepted 13 Apr. 2021. Date of publication 20 Apr. 2021; date of current version 9 July 2022. (Corresponding authors: Yuan Zhang and Chunxiang Xu.) Digital Object Identifier no. 10.1109/TDSC.2021.3074146

brute-force attack relies on the reliability of the key server: once the key server is compromised, the adversary can still break the confidentiality of outsourced data by performing the brute-force attack. As a consequence, the key server becomes a single point of failure in the scheme.

To resolve this tension, existing schemes [8], [18] distribute the generation of MLE keys from the single key server to multiple ones. It surely makes performing the brute-force attack more difficult, but does not resolve the fundamental issue of trusting a specific group of key servers during the lifetime of protected data. In reality, it is feasible and practical for a sophisticated adversary to corrupt these key servers given enough time [19], [20], and for long-lived data, protection provided by servers-aided MLE could be insufficient [21], [22], [23]. A straightforward way to remedy this problem is to periodically replace key servers by some new ones and let the new key servers re-share a new server-side secret. However, it makes the deduplication on the same file protected under different server-side secrets impossible.

In regards to efficiency, the server-aided MLE [16] as well as traditional MLE schemes [15] requires the user to well maintain the MLE key during the lifetime of protected data, which causes the key management problem to the user. Specifically, in existing server-aided MLE schemes, the number of MLE keys linearly increases with the number of outsourced files. Consequently, the user has to bear heavy storage costs to maintain MLE keys when she/he has a large number of files. This problem could be further exacerbated by the fact that in cloud storage systems, a user always needs to access the outsourced files using different devices. To decrypt the outsourced files, the user has to securely store the MLE keys in all potential devices that would be used to access the outsourced files, which introduces new security issues on key migration between different devices and key storage on multiple devices.

To mitigate the key management problem, the existing work [24] utilizes multiple *key storage servers* to help the user in maintaining their MLE keys in a threshold way. However, such a mechanism is unsatisfactory due to the following reasons. First, it could not be well compatible with the server(s)-aided MLE, since the key storage servers should be independent of the key server(s) to secure against the brute-force attack, which changes the user's interaction pattern in the server(s)-aided MLE and introduces additional costs to employ the key storage servers. Second, the mechanism implicitly requires the user to authenticate herself/himself with key storage servers. To this end, the user might need to store a secret key in all potential devices that would be used to access the outsourced files, which also causes the key management problem.

We observe that an affordable way to address the key management problem is to utilize a password-based layered encryption mechanism: the user utilizes the server(s)-aided MLE to encrypt the file, uses a human-memorable password to encrypt the MLE key, and outsources the ciphertexts of the file and the MLE key to the cloud server. In addition, the user authenticates herself/himself with the cloud server and the key server(s) using a password-based authentication protocol. By doing so, the user, who only needs to provide the correct password, can access the outsourced files in different devices. However, since human-memorable passwords are inherently low-entropy [25], [26], such a password-based mechanism is vulnerable to off-line dictionary guessing attacks (short for DGA).

In this paper, we propose a secure password-protected MLE key scheme for deduplicated cloud storage systems, dubbed SPADE, to address the above problems. SPADE is based on the servers-aided MLE to avoid the single-point-of-failure problem, and its security protection is periodically renewed to free from the reliance on a specific group of key servers in a long period of time. This is achieved by a proactivization mechanism: time in SPADE is divided into fixed intervals of predetermined length called epochs; the key servers are replaced by newly employed ones in different epochs such that an adversary who compromises some key servers in previous epochs cannot help in attacking in subsequent epochs. To enable the cloud server to perform deduplication over files outsourced in different epochs, a handoff mechanism is employed, where the server-side secret is transferred from some key servers that are not compromised in the current epoch to all key servers in the next epoch.

The user's password serves as a fundamental role in SPADE. On the one hand, SPADE is built on a password-based layered encryption mechanism described above. The user only needs to retrieve the corresponding MLE key from the cloud server using the correct password when she/he wants to decrypt the outsourced files, which frees the user from the key management problem. On the other hand, in SPADE, the user authenticates herself/himself with the key servers and the cloud server using the same password. The key observation behind SPADE is that *the success of DGA against passwords and the success of the brute-force attack against MLE are essentially caused by the same vulnerability: the only secret is low-entropy and anyone who has the secret can completely execute the prescribed scheme*. With this observation, we present a servers-aided password-hardening protocol that protects the user's password against DGA while remaining the functionalities of the password, which we believe might be of independent interest.

Specifically, the contributions of this paper are summarized as follows.

- We propose a secure proactivization mechanism for servers-aided MLE to resist the brute-force attacks and support periodical changes to key servers to renew the security protection. With the integration of the proposed proactivization mechanism, the security of servers-aided MLE does not rely on a specific group of key servers during the lifetime of protected data, and the cloud server can still perform deduplication over ciphertexts across its user, even if the ciphertexts are protected by different groups of key servers.
- We present a servers-aided password-hardening protocol to protect the user's password against DGA while remaining the functionalities of the password. We further propose a password-based layered encryption mechanism for the encrypted data deduplication scheme, where the user utilizes the password to protect the MLE key such that she/he can manage the MLE via the cloud storage service without

maintaining any secret in her/his local devices. We also propose a password-based authentication mechanism to enable the user to authenticate herself/himself with both the key servers and the cloud server only using the password. Both these mechanisms are built on the servers-aided password-hardening protocol such that they are secure against DGA.

- We integrate the above mechanisms into one scheme called SPADE, which remains all features of each individual mechanism without changing the underlying architecture of servers-aided MLE. We provide formal security proofs to demonstrate that SPADE is secure against brute-force attacks towards ciphertexts and DGA towards users' passwords launched by an adversarial cloud server, even if one or more key servers are compromised. We also evaluate the performance of SPADE via the implementation and comprehensive analysis, which shows that SPADE is efficient and can be easily deployed in reality.

The remainder of this paper is organized as follows. We review the related works in Section 2, and provide the preliminaries in Section 3. We overview SPADE and present its construction in Sections 4 and 5, respectively. In Section 6, we prove the security of SPADE. Then, we evaluate the performance of SPADE in Section 7. Finally, we draw the conclusion and outlook the future research directions in Section 8.

2 Related Work

The data deduplication technique allows a storage server to check duplicate data in its local storage and enables it to save storage space significantly. In cloud storage systems, due to the existence of internal and external threats, data is always outsourced to the cloud server in the ciphertext form. If users utilize convention symmetric-key encryption algorithms, different users would produce different ciphertexts for the same data, which precludes the cloud server from performing deduplication.

The problem of encrypted data deduplication (i.e., deduplication over ciphertexts) is first pointed out by Douceur *et al.* [14], where an elegant solution, called convergent encryption (CE), is proposed. In CE, the encryption/decryption key is the hash value of the data to be encrypted, which enables different users to obtain the same ciphertext for the same data. Subsequently, many CE variants [7], [27], [28], [29] are proposed to improve efficiency. These works mainly target at designing an encrypted deduplication scheme for specific application scenarios, but the theoretical treatment on such a type of encryption algorithms is still lacked. Bellare *et al.* [15] first formalize CE and its variants as a cryptographic primitive of "message-locked encryption" (short for MLE), and such a name indicates that the message is "locked" by itself. In [15], formal security definitions on MLE are also proposed. Theoretically, any encryption algorithm that supports deduplication across multiple users can be subsumed into MLE.

Traditional MLE algorithms can be mainly classified into four types and have been introduced in [15]. In these MLE algorithms, the only secret is the message itself. As a consequence, if the message is low-entropy, the message protected by these MLE algorithms is vulnerable to the brute-force attack: an adversary can enumerate all possible messages and encrypt them using an MLE algorithm one by one, if a ciphertext (or a deduplication fingerprint) matches a target one that has been outsourced to the cloud server, the underlying message is recovered. We stress that vulnerability of MLE against the brute-force attack is a major hindrance towards the broad adoption of MLE, and variants of the attack would cause serious threats on the users' privacy [30].

The first deduplication scheme with resistance against the brute-force attack is proposed by Bellare *et al.* [16]. The key technique is to utilize a server-aided MLE as the underlying encryption algorithm: an independent key server, which has a server-side secret, is introduced to assist users in generating MLE keys. As a result, an MLE key is derived from the message itself and the server-side secret. As long as the server-side secret remains inaccessible to adversaries, the ciphertext is secure against the brute-force attack. Furthermore, the interaction between a user and the key server is oblivious, which ensures that the information about the message would not be leaked to the key server. Subsequently, variants of server-aided MLE [31], [32], [33], [34] have been proposed to improve efficiency. However, the server-aided MLE is confronted with the single-point-of-failure problem: if the key server is incentivized by the adversary to assist him in launching the brute-force attack, the adversary can recover the data content. Although subsequent schemes [8], [18] mitigate this problem by employing multiple key servers, the fundamental issue of trusting a specific group of key servers during the lifetime of protected data still exists. An alternative solution to address the security issues caused by employing key server(s) is to "avoid" the key server(s). The first scheme is proposed by Liu *et al.* [35]. In this scheme, to resist the brute-force attacks in a deduplicated cloud storage system, a user, who wants to upload some data M to the cloud server, needs to encrypt M with the aid of users who previously outsource M to the same server. However, in such a scheme, when a new user wants to upload M to the cloud server, at least one of the owners of M needs to keep online to assist her/him in encrypting M, which introduces additional costs on the "old" user side.

Another line of work [24], [36] focuses on the MLE key management in encrypted data deduplication schemes, where a group of key storage servers is employed to help users in maintaining their MLE keys. However, such a mechanism introduces additional costs on the user side and essentially requires users to securely maintain secret keys (at least one secret key) in all potential devices that would be used to access the outsourced data.

We also note that some password-hardening protocols [37], [38], [39] have been proposed to resist the password dictionary guessing attack (DGA). However, they cannot be integrated into SPADE since they are not compatible with the proactivization mechanism.

In the previous version of this paper [1], a secure encrypted deduplication scheme, dubbed DECKS, is proposed. In DECKS, the security against the brute-force attack does not rely on a specific group of key servers during the lifetime of protected data. Compared with DECKS [1], *we have made the following improvements in this paper*.

- We propose a servers-aided password-hardening protocol to resist DGA. The proposed protocol is well compatible with the proactivization mechanism in DECKS and is highly efficient in terms of computation and communication.
- Based on the servers-aided password-hardening protocol, we further construct a password-based layered encryption mechanism and a password-based authentication mechanism, which enables users to securely and efficiently access the cloud storage services without needing to maintain any secret key in their local devices.
- We integrate the proposed mechanisms into DECKS to construct a new scheme, i.e., SPADE, to allow users to manage their MLE keys in a secure and efficient way, which does not change the system model of DECKS and remains all characteristics. We also provide a formal security proof of SPADE and demonstrate that SPADE is efficient via a comprehensive performance evaluation.

3 Preliminaries

3.1 Notations and Basic Theory

In this paper, we denote by ℓ the security parameter, by $i++$ the operation of $i = i + 1$ for a positive integer i, and by $|\vec{T}|$ the number of components in a finite set $\vec{T}$.

Message-Locked Encryption (MLE). MLE is a special type of symmetric encryption, in which encryption/decryption keys are extracted from messages themselves. It supports encrypted data deduplication, since different users can output the same ciphertext for the same message.

Threshold Cryptography. In a (t, n)-threshold cryptosystem, n players share a secret and each one has a secret share. Any t players can pool their shares and accomplish certain cryptographic operations, but no coalition of fewer than t players can extract any information on the secret from their collective shares [40], [41].

Bilinear Pairing. Suppose that G is an additive group whose order is a prime p, and G_T is a multiplicative group with the same order. $e : G \times G \rightarrow G_T$ is a bilinear pairing if it has three properties. Bilinearity: $e(aX, bY) = e(X, Y)^{ab}$, $\forall X, Y \in G, a, b \in Z_p^*$; non-degeneracy: $\forall X, Y \in G, X \neq Y$, $e(X, Y) \neq 1$; computability: e can be computed efficiently.

3.2 Modeling Servers-Aided Deduplicated Cloud Storage System

As shown in Fig. 1, there are three entities in a servers-aided deduplicated cloud storage system.

- Users. The users are data owners and outsource files to the same cloud server. Users never communicate with each other directly, but they desire to reduce the storage costs from data deduplication. For the purpose of privacy protection, the contents of outsourced files should not be leaked to anyone who does not own the data. Hence, the files should be encrypted on the user side before they are outsourced to the cloud server.
- Cloud server. The cloud server provides storage services for users. It checks the duplicate file across its users and stores only a single copy of the duplicate file to reduce the storage costs. The cloud server only provides services for registered users, therefore, any user needs to register with and log in the cloud server before accessing its services.
- Key servers. Key servers are independent of users and the cloud server. They are employed to help users in generating MLE keys to resist the brute-force attack. To this end, all key servers share a server-side secret via a (t, n)-threshold secret sharing protocol, and each key server has a server-side secret share. An MLE key is derived from the outsourced file itself and the server-side secret and is generated by an algorithm that is interactively executed by the user and all key servers. Similarly, key servers only provide services for registered users, and thereby a registration process and an authentication process between each key server and a user are required.

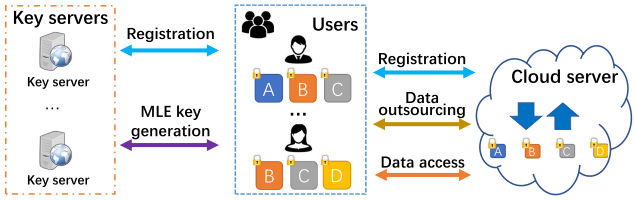

Fig. 1. System model.

From the perspective of data processing, the procedure of a servers-aided deduplicated cloud storage system is described as follows.

To utilize the services provided by the key servers and cloud server, each user needs to register with all key servers and the cloud server. After the registration, there are two cases in the system.

First, when the user generates a new file and wants to outsource it to the cloud server, she/he authenticates herself/himself with the key servers. After the authentication succeeds, the user interacts with key servers to generate an MLE key on the newly generated file, which is achieved by a threshold, deterministic, and oblivious protocol such that

- for the same file, different users would obtain the same servers-derived MLE key,
- key servers can assist users in generating MLE keys but cannot learn anything about the file,
- and compromising any one of the key servers would not break the security of the scheme.

With the servers-derived MLE key, the user encrypts the file by using a secure symmetric-key encryption algorithm. She/he also authenticates herself/himself with the cloud server. After the authentication passes, the user outsources the ciphertext to the cloud server, deletes the file as well as its ciphertext locally, and well maintains the MLE key locally.

Second, when the user wants to access her/his outsourced file, she/he also needs to be authenticated by the cloud server. If the authentication passes, the user is able to download the ciphertext of target file from the cloud server and decrypts it using the MLE key.

In reality, *human-memorable passwords* remain the most prevalent form of user authentication in cloud storage systems.

3.3 Threat Model

In the threat model, we mainly consider two types of adversaries.

Adversarial but Rational Cloud Server. The goal of an adversarial cloud server is to violate the users' privacy. It either attempts to extract the contents of outsourced files, or tries to impersonate the user to pass the key servers' authentication. To this end, the adversarial cloud server may perform various attacks, such as brute-force attacks, compromising key servers, and dictionary guessing attacks (if the password-based authentication is adopted). The cloud server is also a rational party: it would not launch attacks if its profits cannot be increased.

Compromised Key Server(s). Key servers can be fully controlled by the adversarial cloud server to help it to launch various attacks to violate the users' privacy. As the generation of an MLE key requires the user to communicate with key servers, a key server may extract the information about the file from the interaction between them. Furthermore, a compromised key server also attempts to impersonate a target user to pass other key servers' authentication. *We assume that given a fixed time interval, the number of key servers compromised by the adversarial cloud server is limited, otherwise, the costs to launch attacks are higher than the value of protected files*.

We stress that external adversaries exist in reality. They may eavesdrop on the communications between the user and other entities in the system to violate the user's privacy. In addition, they may impersonate the user to pass the cloud server and key servers' authentication. However, as the key servers could be compromised by the cloud server, such an external adversary can be considered as a weakened adversarial cloud server, i.e., any attack performed by the external adversary can also be launched by the adversarial cloud server. Therefore, if the adversarial cloud server can be resisted, the external adversary can also be thwarted.

3.4 Design Goals

SPADE should achieve the following objects.

- *Functionality*. Both the cloud server and the key servers can authenticate users. The cloud server can perform deduplication across all its users. Users are not required to directly communicate with each other and do not need to maintain any secret key in their local storage.
- *Security*. SPADE should be secure against the adversarial cloud server and compromised key servers. Its security against various attacks should not rely on the reliability of a specific group of key servers.
- *Efficiency*. The communication and computation overhead on each entity in SPADE should be as efficient as possible.

4 Overview of SPADE

In this section, we give an overview of SPADE, focusing on the challenges addressed by our scheme.

Resistance Against Compromised Key Servers. We notice that the deduplication schemes constructed on the single-server-aided MLE, e.g., [16], [32], are confronted with the single-point-of-failure problem: an adversarial cloud server, which compromises the key server, is able to break the confidentiality of the outsourced data by launching the brute-force attack. SPADE employs the servers-aided MLE [8] as the underlying encryption algorithm, where n key servers $\{\mathcal{KS}_1, \mathcal{KS}_2, \ldots, \mathcal{KS}_n\}$ are introduced to assist users in generating MLE keys in a deterministic and threshold way as described in Section 3.2. However, such a servers-aided MLE does not resolve the fundamental issue of trusting a specific group of key servers during the lifetime of protected data.

To address this problem, SPADE leverages a proactivization mechanism that periodically replaces the key servers by newly employed ones. Specifically, time in SPADE is divided into fixed intervals of predetermined length called *epochs*. In each epoch, a committee of n key servers shares a server-side secret α in an oblivious (t, n)-threshold manner to assist users in generating MLE keys, where the ith key server has a server-side secret share α_i for $i = 1, 2, \ldots, n$. In different epochs, SPADE updates the committee of key servers to renew the security protection. To enable the cloud server to perform deduplication over ciphertexts outsourced in different epochs, the server-side secret α should be kept consistent. This is achieved by a handoff process described below. We assume that the key servers in the χth epoch form a committee $\vec{\mathcal{KS}}^{(\chi)} = \{\mathcal{KS}_1^{(\chi)}, \mathcal{KS}_2^{(\chi)}, \ldots, \mathcal{KS}_n^{(\chi)}\}$ and the key servers in the $(\chi+1)$th epoch form a committee $\vec{\mathcal{KS}}^{(\chi+1)} = \{\mathcal{KS}_1^{(\chi+1)}, \mathcal{KS}_2^{(\chi+1)}, \ldots, \mathcal{KS}_n^{(\chi+1)}\}$. At the end of each epoch, t honest key servers in the current epoch securely transfer α to all key servers in the next epoch, such that

- α can be re-distributed among all key servers in the next epoch,
- the shares of α owned by key servers in different epochs are independent,
- and any key server cannot extract others' shares, no matter whether they are in the same epoch.

The key technique used here is the Shamir secret sharing protocol [23], [40]. Recall that in the χth epoch, α is shared among $\vec{\mathcal{KS}}^{(\chi)}$, and $\mathcal{KS}_i^{(\chi)}$ has the server-side secret share $\alpha_i^{(\chi)}$. At the end of the χth epoch, t honest and reliable key servers $\{\mathcal{KS}_{i_1}^{(\chi)}, \mathcal{KS}_{i_2}^{(\chi)}, \ldots, \mathcal{KS}_{i_t}^{(\chi)}\}$ are selected, and for $k = 1, 2, \ldots, t$, $\mathcal{KS}_{i_k}^{(\chi)}$ splits its server-side secret share $\alpha_{i_k}^{(\chi)}$ into n fragments $\{\alpha_{i_k,1}^{(\chi)}, \alpha_{i_k,2}^{(\chi)}, \ldots, \alpha_{i_k,n}^{(\chi)}\}$ using the Shamir secret sharing protocol, where $\mathcal{KS}_{i_k}^{(\chi)}$ plays the role of the trusted dealer. Then, $\mathcal{KS}_{i_k}^{(\chi)}$ sends $\alpha_{i_k,j}^{(\chi)}$ to $\mathcal{KS}_j^{(\chi+1)}$ for $j = 1, 2, \ldots, n$ via a secure channel. Finally, $\mathcal{KS}_j^{(\chi+1)}$ utilizes the received t fragments (from $\{\mathcal{KS}_{i_1}^{(\chi)}, \mathcal{KS}_{i_2}^{(\chi)}, \ldots, \mathcal{KS}_{i_t}^{(\chi)}\}$) with the Lagrange coefficients to compute its server-side secret $\alpha_j^{(\chi+1)}$. By doing so, α is securely transferred from $\vec{\mathcal{KS}}^{(\chi)}$ to $\vec{\mathcal{KS}}^{(\chi+1)}$.

MLE Key Management. Regarding to resistance against the brute-force attack, the inefficiency that users employ *key storage servers* to manage their MLE keys has been discussed and analyzed in Sections 1 and 2. SPADE utilizes a layered encryption mechanism: The original file is encrypted by using MLE, and the MLE key is encrypted by using a secure symmetric-key encryption algorithm under *a master key*. Such a mechanism ensures that the storage costs to maintain MLE keys on the user side are constant and are independent

of the number of outsourced files. However, such a mechanism still suffers from the key management problem. Through the lifecycle of outsourced files, the user needs to well maintain the master key in her/his local device, which requires the user to employ additional secure mechanisms and introduces significant costs on the user side. This problem would be further exacerbated by the fact that a user always accesses the outsourced files using different devices.

SPADE addresses the above problem in a very pragmatic way: the user encrypts MLE keys with her/his *password*, i.e., the master key is set to the user's password. By doing so, the user does not need to maintain any secret in her/his local device, and thereby would not be confronted with the key management problem. Nevertheless, such a password-based layered encryption mechanism introduces new challenges in reality.

First, human-memorable passwords are inherently low-entropy, which causes the vulnerability of passwords against off-line dictionary guessing attacks (DGA). Therefore, resistance against DGA is a fundamental security requirement that should be achieved.

Second, it requires the user to authenticate herself/himself with other entities (i.e., the cloud server $\mathcal{CS}$ and key servers $\{\mathcal{KS}_1, \mathcal{KS}_2, \ldots, \mathcal{KS}_n\}$) by using a password-based authentication mechanism. We notice that except the password-based authentication mechanism, others require the user to either maintain a secret in her/his local device (e.g., identity-based authentication mechanisms [42]) or equip a powerful device (e.g., hardware-based authentication mechanisms [43]) for secure authentication. When they are employed in our scheme, the problems addressed by adopting the password-based layered encryption mechanism are introduced again. However, due to the low entropy of passwords, directly integrating existing password-based authentication protocols (e.g., [44], [45], [46]) into our scheme is also vulnerable to DGA.

Resistance Against DGA. The above two problems correspond to the same challenge, i.e., how to design a secure password-hardening protocol that resists DGA while remaining its functionality. In regards to our scheme, such a mechanism also needs to be well compatible with the underlying encrypted data deduplication scheme as well as the proposed proactivization mechanism to constitute a secure system without introducing additional entities.

Recall our observation on the relationship between the vulnerability of password-based cryptographic schemes against DGA and that of MLE against brute-force attack presented in Introduction, it inspires us to adopt the servers-aided mechanism in password-based cryptographic schemes to thwart DGA. Based on the observation, we propose a servers-aided password-hardening protocol. With the proposed protocol, we further design a password-based authentication mechanism and a password-based layered encryption mechanism. Both the mechanisms are secure against DGA while remaining the functionalities and can be utilized in other application scenarios, which we believe might of independent interest.

Particularly, in SPADE, in addition to the server-side secret α, when a user $\mathcal{U}$ registers with $\{\mathcal{KS}_1, \mathcal{KS}_2, \ldots, \mathcal{KS}_n\}$, they generate and share a password-hardening key λ for $\mathcal{U}$ in a distributed (t, n)-threshold way, such that $\mathcal{KS}_i$ has a password-hardening key share λ_i. $\mathcal{U}$ takes her/his password $psw_{\mathcal{U}}$ as input and interacts with the key servers in an oblivious and deterministic way to obtain a signature $\delta_{\mathcal{U}}$ on $psw_{\mathcal{U}}$ under λ. With $\delta_{\mathcal{U}}$, $\mathcal{U}$ can compute a servers-protected password $spsw_{\mathcal{U}}$ as $h(\delta_{\mathcal{U}}||psw_{\mathcal{U}})$, where h is a secure hash function. With $spsw_{\mathcal{U}}$, the authentication credential to authenticate $\mathcal{U}$ could be $\hbar(spsw_{\mathcal{U}})$ and the encryption key to encrypt MLE keys could be $h_1(spsw_{\mathcal{U}})$, where $\hbar$ and h_1 are secure hash functions. Since $spsw_{\mathcal{U}}$ can be interactively generated with key servers, $\mathcal{U}$ only needs to utilize the correct password to retrieve it, which does not require $\mathcal{U}$ to maintain any secret in her/his local device. Meanwhile, $spsw_{\mathcal{U}}$ is computed on both the password and the password-hardening key, as well as t key servers remain unavailable to the adversary, $spsw_{\mathcal{U}}$ is secure against DGA. We also stress that such a servers-aided password-hardening protocol can be well compatible with the proactivization mechanism, i.e., the key servers can be periodically replaced by newly employed ones to provide a strong security guarantee.

However, there is still a subtle security problem. Consider an adversary who compromises a key server and obtains the authentication credential $h(spsw_{\mathcal{U}})$ of $\mathcal{U}$, then he is able to impersonate $\mathcal{U}$ to pass the authentication of $\mathcal{CS}$ and $\mathcal{KS}_i$ without knowing $psw_{\mathcal{U}}$. To resist such an impersonation attack, the authentication credential maintained by $\mathcal{CS}$ and $\mathcal{KS}_i$ should be different, such that an authentication credential maintained by one entity cannot be deduced from those maintained by others. To this end, in SPADE, the authenticate credential maintained by the entity is the hash value of $spsw_{\mathcal{U}}$ and the *entity's identity*.

Resistance Against Online Adversaries. We further consider an online adversary. On the one hand, the adversary may impersonate valid users to request MLE keys on all possible file candidates from key servers. With the MLE keys for these candidates, the adversary can recover the content of a target ciphertext in an off-line manner. Such an attack is called the online brute-force attack. On the other hand, the adversary may impersonate a target user to interact with key servers using all possible passwords. If anyone is valid, the adversary can pass the key servers' authentication. Such an attack is called the online DGA.

SPADE thwarts the online adversary by utilizing a rate-limiting mechanism: in each epoch, *the number of requests on MLE keys* and *the number of failures for login request* made by a user are limited by key servers. To keep the consistency of these two numbers on different key servers without requiring the key servers to communicate with each other, SPADE requires the user to interact with all key servers, rather than t of them, in each request of MLE key and/or servers-protected password. This yields the final scheme, SPADE, which achieves all the design goals presented in Section 3.4.

In Fig. 2, we provide an illustrate the procedure of SPADE to show the purpose of each interaction between two entities.

5 Proposed SPADE

5.1 Construction of SPADE

A set of users $\vec{\mathcal{U}}$, a set of key servers $\vec{\mathcal{KS}} = \{\mathcal{KS}_1, \ldots, \mathcal{KS}_n\}$ with identities $\{ID_{\mathcal{KS}_1}, \ldots, ID_{\mathcal{KS}_n}\}$, and a cloud server $\mathcal{CS}$ are involved in SPADE. In this section, we only show the process that a user $\mathcal{U} \in \vec{\mathcal{U}}$ with an identity $ID_{\mathcal{U}}$ logs in SPADE, outsources files, and accesses the outsourced files

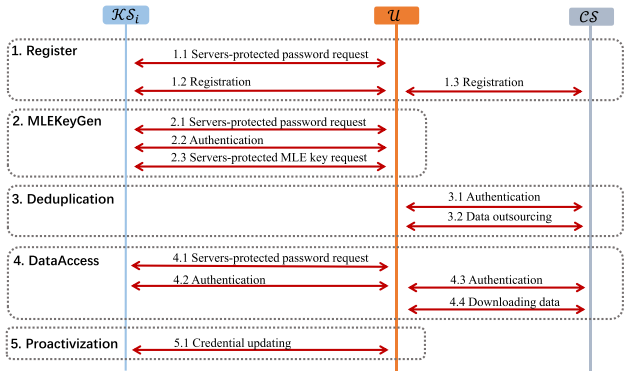

Fig. 2. Procedure of SPADE.

in the following. Other users in $\vec{\mathcal{U}}$ would follow this process, and we would not repeat it.

Algorithm 1. DSGS

Input: Security parameter ℓ, the identities of key servers $\{ID_{\mathcal{KS}_1}, \ldots, ID_{\mathcal{KS}_n}\}$, a threshold t, the index i of key server who executes this algorithm.

Output: A secret s is generated and shared among all key servers and the corresponding public key PS is published. Each key server obtains a secret share (i.e., $\mathcal{KS}_i$ obtains s_i) of s, and the corresponding public share is published.

1: $\mathcal{KS}_i$ randomly chooses a nonce N_i, an element $a_{i,0} \in Z_p^*$ and a polynomial $f_i(x) = a_{i,0} + a_{i,1}x + \cdots + a_{i,t-1}x^{t-1}$ over Z_p with degree at most $t-1$ such that $f_i(0) = a_{i,0}$, and publishes N_i;

2: $\mathcal{KS}_i$ generates a session identity $SID^{(0)} = \{(ID_{\mathcal{KS}_1}, N_1), (ID_{\mathcal{KS}_2}, N_2), \ldots, (ID_{\mathcal{KS}_n}, N_n)\}$, computes $a_{i,0} \cdot P$ and $a_{i,\epsilon} \cdot P$ for $\epsilon = 1, 2, \ldots, t-1$, and calculates $f_i(j)$ for $j = 1, 2, \ldots, n$ and $j \neq i$;

3: For $\epsilon = 1, 2, \ldots, t-1$ and $j = 1, 2, \ldots, n$ and $j \neq i$, $\mathcal{KS}_i$ computes

$$\theta_i = \{SID^{(0)}, a_{i,0} \cdot P, \{a_{i,\epsilon} \cdot P\}, \{\mathsf{Enc}(epk_j, f_i(j))\}\},$$
$$\Phi_i = \mathsf{Sig}(ssk_i, \theta_i),$$

and publishes $\{\theta_i, \Phi_i\}$.

4: For $j = 1, 2, \ldots, n$ and $j \neq i$, $\mathcal{KS}_i$ obtains $\{\theta_j, \Phi_j\}$, accepts it if $SID^{(0)}$ and Φ_j are valid, and decrypts $\mathsf{Enc}(epk_i, f_j(i))$ to get $f_j(i)$. If Equation (1) holds, $\mathcal{KS}_i$ accepts $f_j(i)$.

$$f_j(i)P = \sum_{\epsilon=0}^{t-1} i^{\epsilon} a_{j,\epsilon} \cdot P. \tag{1}$$

5: $\mathcal{KS}_i$ computes its secret share $s_i = \sum_{\gamma=1}^{n} f_\gamma(i)$, the public share $PS_i = s_i \cdot P$;

6: $\mathcal{KS}_i$ computes public key $PS = \sum_{i=1}^{n} a_{i,0} \cdot P$, securely stores s_i and maintains $\{PS_1, PS_2, \ldots, PS_n\}$.

7: The secret key $s = \sum_{i=1}^{n} a_{i,0}$ is shared among all key servers.

Setup. With the security parameter ℓ, public parameters $\{p, P, G, G_T, e, H_1, H_2, h, h_1, \hbar, \mathsf{E}(\cdot), \mathsf{Enc}(\cdot), \mathsf{Sig}(\cdot), t, n, \rho, cf\}$ are determined, where $e : G \times G \to G_T$ is the bilinear pairing, P is a generator of G, $H_1, H_2 : \{0,1\}^* \to G$, $h, h_1 : G \to \{0,1\}^{\ell}$, $\hbar : \{0,1\}^* \to Z_p$ are secure hash functions, $\mathsf{E}(k, M)$ is a symmetric encryption algorithm (CTR[AES]) to encrypt M with k, $\mathsf{Enc}(epk, M)$ is a secure public-key encryption algorithm to encrypt M with epk, $\mathsf{Sig}(ssk, M)$ is a secure signature algorithm [47] to sign M with ssk, t is a threshold, n is the number of key servers, ρ is the upper bound of MLE key requests made by a user in an epoch, cf is an upper bound that a user fails to pass key servers' authentication.

For $i = 1, 2, \ldots, n$, $\mathcal{KS}_i$ generates a signing key pair (ssk_i, spk_i) and a public-key encryption key pair (esk_i, epk_i). All key servers jointly execute the distributed secret generating and sharing protocol (short for DSGS) shown in Algorithm 1 to generate the server-side secret $\alpha = s$. After the execution, $\mathcal{KS}_i$ has a server-side secret share $\alpha_i = s_i$. The corresponding public key is $Q = PS = \alpha \cdot P$, and the corresponding public share is $Q_i = PS_i = \alpha_i \cdot P$.

$\mathcal{KS}_i$ maintains $\rho_{\mathcal{U}}$ to count up how many times $\mathcal{U}$ requests MLE keys in the current epoch.

Register. Before outsourcing data to the cloud server $\mathcal{CS}$, $\mathcal{U}$ registers with $\mathcal{CS}$ and all key servers as follows.

- $\mathcal{U}$ creates a human-memorable password $psw_{\mathcal{U}}$, randomly selects $r \in Z_p^*$, blinds her/his password as $psw_{\mathcal{U}}^* = r \cdot H_1(psw_{\mathcal{U}})$, and sends $(ID_{\mathcal{U}}, psw_{\mathcal{U}}^*)$ to all key servers.
- For $i = 1, 2, \ldots, n$, $\mathcal{KS}_i$ first checks whether $ID_{\mathcal{U}}$ exists in its local storage, if yes, it means that $ID_{\mathcal{U}}$ has been registered; otherwise, all key servers store $ID_{\mathcal{U}}$ and generate a password-hardening key λ by performing DSGS shown in Algorithm 1, where $a_{i,0}, a_{i,1}, \ldots, a_{i,t-1}$ are newly selected and are different from those in generating α. For $\mathcal{U}$, the password-hardening key $\lambda = s$, the corresponding public password-hardening key $V = PS = \lambda \cdot P$, $\mathcal{KS}_i$'s password-hardening key share $\lambda_i = s_i$ and the corresponding public password-hardening key share $V_i = PS_i = \lambda_i \cdot P$.
- For $i \in [1, n]$, $\mathcal{KS}_i$ computes $\delta_i^* = \lambda_i \cdot psw_{\mathcal{U}}^*$ and sends δ_i^* to $\mathcal{U}$.
- $\mathcal{U}$ verifies δ_i^* by verifying

$$e(\delta_i^*, P) = e(psw_{\mathcal{U}}^*, V_i). \tag{2}$$

If the verification passes, δ_i^* is considered as a valid signature. After receiving t valid signatures (for the sake of brevity, these signatures are denoted by $\{\delta_1^*, \delta_2^*, \ldots, \delta_t^*\}$), $\mathcal{U}$ computes

$$\omega_\zeta = \prod_{\substack{1 \le \eta \le t \\ \eta \neq \zeta}} \frac{\eta}{\eta - \zeta}, \tag{3}$$

$$\delta_{\mathcal{U}} = r^{-1} \cdot \sum_{\zeta=1}^{t} \omega_\zeta \cdot \delta_\zeta^*. \tag{4}$$

- $\mathcal{U}$ verifies the validity of $\delta_{\mathcal{U}}$ by verifying

$$e(\delta_{\mathcal{U}}, P) = e(H_1(psw_{\mathcal{U}}), V). \tag{5}$$

Due to Lagrange interpolation, $\delta_{\mathcal{U}} = \lambda \cdot H_1(psw_{\mathcal{U}})$ is a BLS signature of $psw_{\mathcal{U}}$ under the password-hardening key λ.

- $\mathcal{U}$ computes a servers-protected password $spsw_{\mathcal{U}}$ as

$$spsw_{\mathcal{U}} = h(\delta_{\mathcal{U}} || psw_{\mathcal{U}}). \tag{6}$$

- For $i = 1, 2, \ldots, n$, $\mathcal{U}$ computes the authentication credential $cre_i = \hbar(spsw_{\mathcal{U}}||ID_{\mathcal{KS}_i})$ and sends cre_i to $\mathcal{KS}_i$ via a secure channel. $\mathcal{U}$ also computes $cre_{\mathcal{U}} = \hbar(spsw_{\mathcal{U}}||ID_{\mathcal{CS}})$ and sends $(ID_{\mathcal{U}}, cre_{\mathcal{U}})$ to $\mathcal{CS}$.
- $\mathcal{KS}_i$ stores cre_i locally as the authentication credential of $\mathcal{U}$, and initiates a counter $cf_{\mathcal{U}} = 0$ to count up how many times $\mathcal{U}$ fails to pass the authentication in the current epoch.
- $\mathcal{CS}$ checks whether $ID_{\mathcal{U}}$ exists in its local storage, if yes, it means that $ID_{\mathcal{U}}$ has been registered; otherwise, $\mathcal{CS}$ locally stores $(ID_{\mathcal{U}}, cre_{\mathcal{U}})$ in its secure storage for subsequent authentications.

MLEKeyGen. When $\mathcal{U}$ wants to outsource some file to $\mathcal{CS}$, she/he needs to authenticate himself/herself with $\mathcal{CS}$ and encrypt the file before outsourcing it to $\mathcal{CS}$, which requires $\mathcal{U}$ to generate the authentication credential $cre_{\mathcal{U}}$ and the corresponding MLE key. To this end, $\mathcal{U}$ interacts with key servers as follows.

- $\mathcal{U}$ takes her/his password $psw_{\mathcal{U}}$ as input, randomly selects two elements $r_1, r_2 \in Z_p^*$, generates a file M and computes $psw_{\mathcal{U}}^* = r_1 \cdot H_1(psw_{\mathcal{U}})$, $M^* = r_2 \cdot H_2(M)$.
- $\mathcal{U}$ sends $(ID_{\mathcal{U}}, psw_{\mathcal{U}}^*)$ to $\mathcal{KS}_i$ for $i = 1, 2, \ldots, n$.
- After receiving $(ID_{\mathcal{U}}, psw_{\mathcal{U}}^*)$, $\mathcal{KS}_i$ verifies $cf_{\mathcal{U}} \leq cf$, if the verification fails, it aborts; otherwise, it computes $\delta_i^* = \lambda_i \cdot psw_{\mathcal{U}}^*$, randomly selects $\gamma_i \in Z_p$, and sends (δ_i^*, γ_i) to $\mathcal{U}$.
- Upon receiving (δ_i^*, γ_i), $\mathcal{U}$ verifies the validity of δ_i^* by checking Equation (2). After receiving t valid signatures, $\mathcal{U}$ is able to retrieve $\delta_{\mathcal{U}}$ (using r_1) via Equations (3) and (4). With $\delta_{\mathcal{U}}$, $\mathcal{U}$ can compute $spsw_{\mathcal{U}}$ by Equation (6) and further compute $cre_i = \hbar(spsw_{\mathcal{U}}||ID_{\mathcal{KS}_i})$. Then, $\mathcal{U}$ sends $EM = \mathsf{E}(cre_i, \gamma_i||M^*)$ to $\mathcal{KS}_i$.
- After receiving EM, $\mathcal{KS}_i$ decrypts it and obtains γ_i and M^*. If the decryption fails, γ_i is different from the one it sent before, or $\rho_{\mathcal{U}} > \rho$, it aborts and increments $cf_{\mathcal{U}}$ by 1; otherwise, $\mathcal{KS}_i$ increments $\rho_{\mathcal{U}}$ by 1, computes $\sigma_i^* = \alpha_i \cdot M^*$, and sends $ES = \mathsf{E}(cre_i, \sigma_i^*)$ to $\mathcal{U}$.
- Upon receiving ES, $\mathcal{U}$ decrypts it, obtains σ_i^*, and verifies

$$e(\sigma_i^*, P) \stackrel{?}{=} e(M^*, Q_i), \tag{7}$$

to check the validity of σ_i^*. After receiving t valid signatures (similarly, these signatures are denoted by $\{\sigma_1^*, \sigma_2^*, \ldots, \sigma_t^*\}$), $\mathcal{U}$ computes

$$w_k = \prod_{\substack{1 \leq \eta \leq t \\ \eta \neq k}} \frac{\eta}{\eta - k}, \tag{8}$$

$$\sigma = r_2^{-1} \sum_{k=1}^{t} w_k \cdot \sigma_k^*. \tag{9}$$

$\mathcal{U}$ checks whether $e(\sigma, P) = e(H_2(M), Q)$ holds and rejects σ if the checking fails.

- $\mathcal{U}$ computes the MLE key of M as $mk_M = h_1(\sigma)$.

Deduplication. $\mathcal{U}$ first generates all ciphertexts that would be outsourced to $\mathcal{CS}$ as follows:

- Encrypt M as $C_M = \mathsf{E}(mk_M, M)$;
- Compute $\kappa_{\mathcal{U}} = h_1(spsw_{\mathcal{U}}||ID_{\mathcal{U}})$;
- Encrypt mk_M as $C_{mk_M}^{\mathcal{U}} = \mathsf{E}(\kappa_{\mathcal{U}}, mk_M)$.

Then, $\mathcal{U}$ computes $cre_{\mathcal{U}} = \hbar(spsw_{\mathcal{U}}||ID_{\mathcal{CS}})$ and obtains the authentication credential to authenticate herself/himself with $\mathcal{CS}$. With $(ID_{\mathcal{U}}, cre_{\mathcal{U}})$, it can establish an authenticated and secure channel between $\mathcal{U}$ and $\mathcal{CS}$. Using this channel, $\mathcal{U}$ outsources $(C_M, C_{mk_M}^{\mathcal{U}})$ to $\mathcal{CS}$. $\mathcal{CS}$ only performs deduplication on C_M across all its users.

For $\mathcal{CS}$, some methods can be utilized to check duplicates. One of the most efficient methods is to check the hash value of the ciphertext. Specifically, $\mathcal{CS}$ computes $\tau_M = \hbar(C_M)$ as a fingerprint for deduplication and maintains $\{C_M, \tau_M\}$ locally. For a subsequent user who uploads a ciphertext $C^{'}$, $\mathcal{CS}$ verifies $\hbar(C^{'}) \stackrel{?}{=} \tau_M$. If the verification passes, $\mathcal{CS}$ performs deduplication; otherwise, $\mathcal{CS}$ maintains $C^{'}$ locally.

After $(C_M, C_{mk_M}^{\mathcal{U}})$ has been outsourced to $\mathcal{CS}$ successfully, $\mathcal{U}$ deletes it from her/his local storage.

DataAccess. Anytime $\mathcal{U}$ wants to access M, she/he needs to authenticate herself/himself with $\mathcal{CS}$, download $(C_M, C_{mk_M}^{\mathcal{U}})$ from $\mathcal{CS}$ and decrypt C_M to obtain M. This process only requires $\mathcal{U}$ to provide the correct password $psw_{\mathcal{U}}$, and the details are described as follows.

- $\mathcal{U}$ takes $psw_{\mathcal{U}}$ as input, randomly chooses $r_3 \in Z_p^*$, blinds $psw_{\mathcal{U}}$ as $psw_{\mathcal{U}}^* = r_3 \cdot H_1(psw_{\mathcal{U}})$, and sends $(ID_{\mathcal{U}}, psw_{\mathcal{U}}^*)$ to $\mathcal{KS}_i$ for $i = 1, 2, \ldots, n$.
- After receiving $(ID_{\mathcal{U}}, psw_{\mathcal{U}}^*)$, $\mathcal{KS}_i$ verifies $cf_{\mathcal{U}} \leq cf$, if the verification fails, it aborts; otherwise, it computes $\delta_i^* = \lambda_i \cdot psw_{\mathcal{U}}^*$, randomly selects $\gamma_i^* \in Z_p$, and sends (δ_i^*, γ_i^*) to $\mathcal{U}$.
- Upon receiving (δ_i^*, γ_i^*), $\mathcal{U}$ verifies the validity of δ_i^* by checking Equation (2). After receiving t valid signatures, $\mathcal{U}$ is able to retrieve $\delta_{\mathcal{U}}$ (using r_3) via Equations (3) and (4). With $\delta_{\mathcal{U}}$, $\mathcal{U}$ can compute $spsw_{\mathcal{U}}$ by Equation (6) and further compute $cre_i = \hbar(spsw_{\mathcal{U}}||ID_{\mathcal{KS}_i})$. Then, $\mathcal{U}$ sends $EM^* = \mathsf{E}(cre_i, \gamma_i^*)$ to $\mathcal{KS}_i$.
- After receiving EM^*, $\mathcal{KS}_i$ decrypts it and obtains γ_i^*. If the decryption fails, $\mathcal{KS}_i$ did not receive any information from $\mathcal{U}$, or γ_i^* is different from the one it sent before, it aborts and increments $cf_{\mathcal{U}}$ by 1.
- With $spsw_{\mathcal{U}}$, $\mathcal{U}$ computes $cre_{\mathcal{U}} = \hbar(spsw_{\mathcal{U}}||ID_{\mathcal{CS}})$ to obtain the authentication credential and computes $\kappa_{\mathcal{U}} = h_1(spsw_{\mathcal{U}}||ID_{\mathcal{U}})$.
- With $(ID_{\mathcal{U}}, cre_{\mathcal{U}})$, it can establish an authenticated and secure channel between $\mathcal{U}$ and $\mathcal{CS}$. Using this channel, $\mathcal{U}$ can authenticate herself/himself with $\mathcal{CS}$ and download $(C_M, C_{mk_M}^{\mathcal{U}})$ from it.
- After downloading $(C_M, C_{mk_M}^{\mathcal{U}})$, $\mathcal{U}$ first decrypts $C_{mk_M}^{\mathcal{U}}$ with $\kappa_{\mathcal{U}}$ to obtain mk_M, and then decrypt C_M with mk_M to obtain M.

Proactivization. At the end of an epoch, the key servers should be replaced by n new key servers. For the sake of brevity, we assume that the key servers in the χth epoch form a committee

$$\vec{\mathcal{KS}}^{(\chi)} = \{\mathcal{KS}_1^{(\chi)}, \mathcal{KS}_2^{(\chi)}, \ldots, \mathcal{KS}_n^{(\chi)}\},$$

and the new key servers in the next epoch, i.e., the $(\chi + 1)$th epoch, form a committee

User Identity	Counter	Server-Side Secret Share	Password-Hardening Key Share	Authentication Credential
$ID_{\mathcal{U}}$	ρ_u, cf_u	$\alpha_i^{(\chi)}$	$\lambda_i^{(\chi)}$	$cre_i^{(\chi)}$

Fig. 3. $\mathcal{KS}_i^{(\chi)}$'s local storage form.

$$\vec{\mathcal{KS}}^{(\chi+1)} = \{\mathcal{KS}_1^{(\chi+1)}, \mathcal{KS}_2^{(\chi+1)}, \ldots, \mathcal{KS}_n^{(\chi+1)}\}.$$

We notice that the local storage of a key server (say $\mathcal{KS}_i^{(\chi)}$) in the χth epoch has the form shown in Fig. 3, where the public parameters are omitted.

The replacement is achieved by a handoff process, where the server-side secret α and password-hardening key λ are redistributed among $\vec{\mathcal{KS}}^{(\chi+1)}$. This process is shown in the following (unless specified otherwise hereinafter, $k = 1, 2, \ldots, t$ and $j = 1, 2, \ldots, n$).

- t honest and reliable key servers $\{\mathcal{KS}_{i_1}^{(\chi)}, \mathcal{KS}_{i_2}^{(\chi)}, \ldots, \mathcal{KS}_{i_t}^{(\chi)}\}$ are selected. Their indexes form a set $T^{(\chi)} = \{i_1, \ldots, i_t\}$. Since the user's identity $ID_{\mathcal{U}}$ is not a secret, we assume that $ID_{\mathcal{U}}$ has been sent to $\vec{\mathcal{KS}}^{(\chi+1)}$ when they are determined.
- $\mathcal{KS}_{i_k}^{(\chi)}$ generates a nonce $N_{i_k}^{(\chi)}$ and publishes it. $\mathcal{KS}_j^{(\chi+1)}$ generates a nonce $N_j^{(\chi+1)}$ and publishes it.
- $\mathcal{KS}_{i_k}^{(\chi)}$ randomly chooses $b_{i_k,1}, b_{i_k,2}, \ldots, b_{i_k,t-1}, c_{i_k,1}, c_{i_k,2}, \ldots, c_{i_k,t-1} \in Z_p$ and generates

$$g_{i_k}^{(\chi)}(x) = \alpha_{i_k}^{(\chi)} + b_{i_k,1}x + \cdots + b_{i_k,t-1}x^{t-1}, \tag{10}$$

$$l_{i_k}^{(\chi)}(x) = \lambda_{i_k}^{(\chi)} + c_{i_k,1}x + \cdots + c_{i_k,t-1}x^{t-1}. \tag{11}$$

- $\mathcal{KS}_{i_k}^{(\chi)}$ generates a session identity

$$SID^{(\chi)} = \Big\{\Big(ID_{\mathcal{KS}_{i_1}^{(\chi)}}, N_{i_1}^{(\chi)}\Big), \ldots, \Big(ID_{\mathcal{KS}_{i_t}^{(\chi)}}, N_{i_t}^{(\chi)}\Big), \Big(ID_{\mathcal{KS}_1^{(\chi+1)}}, N_1^{(\chi+1)}\Big), \ldots, \Big(ID_{\mathcal{KS}_n^{(\chi+1)}}, N_n^{(\chi+1)}\Big)\},$$

  computes $b_{i_k,1} \cdot P$, $b_{i_k,2} \cdot P$, $\ldots$, $b_{i_k,t-1} \cdot P$, $c_{i_k,1} \cdot P$, $c_{i_k,2} \cdot P, \ldots, c_{i_k,t-1} \cdot P$, $\alpha_{i_k,j}^{(\chi)} = g_{i_k}^{(\chi)}(j)$, and $\lambda_{i_k,j}^{(\chi)} = l_{i_k}^{(\chi)}(j)$.
- $\mathcal{KS}_{i_k}^{(\chi)}$ computes

$$\theta_{i_k}^{(\chi)} = \{SID^{(\chi)}, \{b_{i_k,\epsilon} \cdot P, c_{i_k,\epsilon} \cdot P\}_{\epsilon=1,\cdots,t-1}, \{\mathsf{Enc}(epk_j^{(\chi+1)}, \alpha_{i_k,j}^{(\chi)} || \lambda_{i_k,j}^{(\chi)} || cre_{i_k}^{(\chi)})\}\},$$
$$\Phi_{i_k}^{(\chi)} = \mathsf{Sig}(ssk_{i_k}^{(\chi)}, \theta_{i_k}^{(\chi)}).$$

  $\mathcal{KS}_{i_k}^{(\chi)}$ publishes $\{\theta_{i_k}^{(\chi)}, \Phi_{i_k}^{(\chi)}\}$. Here, $Q_{i_k}^{(\chi)} = \alpha_{i_k}^{(\chi)} \cdot P$ and $V_{i_k}^{(\chi)} = \lambda_{i_k}^{(\chi)} \cdot P$ are the public shares and have been published.
- $\mathcal{KS}_j^{(\chi+1)}$ obtains $\{\theta_{i_k}^{(\chi)}, \Phi_{i_k}^{(\chi)}\}$, accepts it if $SID^{(\chi)}$ and $\Phi_{i_k}^{(\chi)}$ are valid, and decrypts $\mathsf{Enc}(epk_j^{(\chi+1)}, \alpha_{i_k,j}^{(\chi)} || \lambda_{i_k,j}^{(\chi)} || cre_{i_k}^{(\chi)})$ to get $\alpha_{i_k,j}^{(\chi)}$, $\lambda_{i_k,j}^{(\chi)}$, and $cre_{i_k}^{(\chi)}$.
- $\mathcal{KS}_j^{(\chi+1)}$ computes the Lagrange coefficients $w_{i_k}^* = \prod_{\substack{i_1 \le \eta \le i_t \\ \eta \in T^{(\chi)}, \eta \ne i_k}} \frac{\eta}{\eta - i_k}$, and verifies

User Identity	Counter	Server-Side Secret Share	Password-Hardening Key Share	Authentication Credential
$ID_{\mathcal{U}}$	ρ_u, cf_u	$\alpha_i^{(\chi+1)}$	$\lambda_i^{(\chi+1)}$	$cre_{i_1}^{(\chi)}, cre_{i_2}^{(\chi)}, \cdots, cre_{i_t}^{(\chi)}$

Fig. 4. $\mathcal{KS}_i^{(\chi+1)}$'s local storage form after handoff.

$$\alpha_{i_k,j}^{(\chi)} P \stackrel{?}{=} \alpha_{i_k}^{(\chi)} \cdot P + \sum_{\epsilon=1}^{t-1} j^{\epsilon} \cdot b_{i_k,\epsilon} \cdot P, \tag{12}$$

$$\lambda_{i_k,j}^{(\chi)} P \stackrel{?}{=} \lambda_{i_k}^{(\chi)} \cdot P + \sum_{\epsilon=1}^{t-1} j^{\epsilon} \cdot c_{i_k,\epsilon} \cdot P, \tag{13}$$

$$Q \stackrel{?}{=} \sum_{k=1}^{t} w_{i_k}^* \cdot Q_{i_k}^{(\chi)}, \tag{14}$$

$$V \stackrel{?}{=} \sum_{k=1}^{t} w_{i_k}^* \cdot V_{i_k}^{(\chi)}. \tag{15}$$

  If the verification fails, $\mathcal{KS}_j^{(\chi+1)}$ aborts; otherwise, it sends an "Accept" message to other key servers.
- After receiving "Accept" messages from all other key servers, $\mathcal{KS}_j^{(\chi+1)}$ computes its secret shares $\alpha_j^{(\chi+1)}$ and $\lambda_j^{(\chi+1)}$ as

$$\alpha_j^{(\chi+1)} = \sum_{k=1}^{t} w_{i_k}^* \cdot \alpha_{i_k,j}^{(\chi)}, \tag{16}$$

$$\lambda_j^{(\chi+1)} = \sum_{k=1}^{t} w_{i_k}^* \cdot \lambda_{i_k,j}^{(\chi)}. \tag{17}$$

  The corresponding public shares are $Q_j^{(\chi+1)} = \alpha_j^{(\chi+1)} \cdot P$ and $V_j^{(\chi+1)} = \lambda_j^{(\chi+1)} \cdot P$.

After the handoff process, the local storage of a key server (say $\mathcal{KS}_i^{(\chi+1)}$) in the $(\chi+1)$th epoch has the form shown in Fig. 4, where the public parameters are omitted.

Now, in order to authenticate $\mathcal{U}$, $\mathcal{KS}_i^{(\chi+1)}$ needs to use $cre_{i_k}^{(\chi)} (k = 1, 2, \ldots, t)$. However, this would be inefficient and insecure. To address this problem, the credentials on the key servers in the new epoch should also be updated. This process is described as follows.

- In the $(\chi+1)$th epoch, $\mathcal{U}$ first authenticates herself/himself with $\mathcal{KS}_i^{(\chi+1)}$ by using $cre_{i_k}^{(\chi)} (k = 1, 2, \ldots, t)$. This process requires $\mathcal{U}$ to take the correct password $psw_{\mathcal{U}}$ as input and to interact with $\mathcal{KS}_i^{(\chi+1)}$ (which utilizes $\lambda_i^{(\chi+1)}$) to retrieves $spsw_{\mathcal{U}}$.
- With $spsw_{\mathcal{U}}$, $\mathcal{U}$ is able to compute $cre_{i_t}^{(\chi)} = \hbar(spsw_{\mathcal{U}} || ID_{\mathcal{KS}_{i_t}^{(\chi)}})$, which enables $\mathcal{U}$ to authenticate herself/himself with $\mathcal{KS}_i^{(\chi+1)}$. If the authentication succeeds, a secure and authenticated channel between $\mathcal{U}$ and $\mathcal{KS}_i^{(\chi+1)}$ can be established, and $\mathcal{U}$ computes a new authentication credential $cre_i^{(\chi+1)} = \hbar(spsw_{\mathcal{U}} || ID_{\mathcal{KS}_i^{(\chi+1)}})$ and sends $cre_i^{(\chi+1)}$ to $\mathcal{KS}_i^{(\chi+1)}$ via the secure channel.
- $\mathcal{KS}_i^{(\chi+1)}$ maintains $\rho_{\mathcal{U}}$ and $cf_{\mathcal{U}}$ (these counters are initialized to 0 at the beginning of the new epoch).

User Identity	Counter	Server-Side Secret Share	Password-Hardening Key Share	Authentication Credential
ID_u	ρ_u, cf_u	$\alpha_i^{(\chi+1)}$	$\lambda_i^{(\chi+1)}$	$cre_i^{(\chi+1)}$

Fig. 5. $\mathcal{KS}_i^{(\chi+1)}$'s local storage form.

Now, the local storage of $\mathcal{KS}_i^{(\chi+1)}$ has the form shown in Fig. 5, where the public parameters are omitted.

$\mathcal{U}$ executes *MLEKeyGen*, *Deduplication*, and *DataAccess* together with $\vec{\mathcal{KS}}^{(\chi+1)}$ as she/he did in the previous epoch.

5.2 Remark

We first prove that both the server-side secret α and the password-hardening key λ would not be changed after the execution of *Proactivization*. For the sake of brevity, we assume that the selected honest key servers in the χth epoch are $\{\mathcal{KS}_1^{(\chi)}, \mathcal{KS}_2^{(\chi)}, \ldots, \mathcal{KS}_t^{(\chi)}\}$, where $\mathcal{KS}_k^{(\chi)}$'s server-side secret share and password-hardening key share are $\alpha_k^{(\chi)}$ and $\lambda_k^{(\chi)}$, respectively. We notice that α and λ have the form

$$\alpha = \sum_{k=1}^{t} w_k \cdot \alpha_k^{(\chi)} = \sum_{\zeta=1}^{t} w_\zeta^* \cdot \alpha_\zeta^{(\chi+1)}, \tag{18}$$

$$\lambda = \sum_{k=1}^{t} w_k \cdot \lambda_k^{(\chi)} = \sum_{\zeta=1}^{t} w_\zeta^* \cdot \lambda_\zeta^{(\chi+1)}, \tag{19}$$

where both w_k and w_ζ^* are Lagrange coefficients. If $\alpha_k^{(\chi)}$, $\alpha_\zeta^{(\chi+1)}$, $\lambda_k^{(\chi)}$, and $\lambda_\zeta^{(\chi+1)}$ are valid, then we can get

$$\begin{aligned}
\alpha &= \sum_{k=1}^{t} w_k \cdot \alpha_k^{(\chi)} = \sum_{k=1}^{t}\left(w_k \sum_{\zeta=1}^{t} w_\zeta^* \cdot \alpha_{k,\zeta}^{(\chi)}\right) \\
&= \sum_{\zeta=1}^{t}\sum_{k=1}^{t} w_\zeta^* w_k \cdot \alpha_{k,\zeta}^{(\chi)} = \sum_{\zeta=1}^{t} w_\zeta^* \alpha_\zeta^{(\chi+1)}. \\
\lambda &= \sum_{k=1}^{t} w_k \cdot \lambda_k^{(\chi)} = \sum_{k=1}^{t}\left(w_k \sum_{\zeta=1}^{t} w_\zeta^* \cdot \lambda_{k,\zeta}^{(\chi)}\right) \\
&= \sum_{\zeta=1}^{t}\sum_{k=1}^{t} w_\zeta^* w_k \cdot \lambda_{k,\zeta}^{(\chi)} = \sum_{\zeta=1}^{t} w_\zeta^* \lambda_\zeta^{(\chi+1)}.
\end{aligned}$$

This ensures that

- in different epochs, $\mathcal{U}$ can only use the correct password to authenticate herself/himself with the cloud server $\mathcal{CS}$, without requiring $\mathcal{CS}$ to update the authentication credential $cre_{\mathcal{U}}$;
- $\mathcal{CS}$ is able to perform deduplication over files outsourced by different users in different epochs.

Actually, the committees in different epochs may *intersect*, and both the threshold t and the total number of key servers n can be changed in different epochs. In addition, some secret sharing algorithms can also be utilized in SPADE as the underlying component to reduce the communication costs on key servers, such as [48]. Due to the space limitation, the details are not provided.

Furthermore, we do not specify how to determine the honest and reliable t key servers at the end of each epoch, since it depends on the underlying application scenario, and can be achieved by utilizing orthogonal techniques, e.g., digital investigations [46], situation awareness [49], and trust evaluation.[2]

SPADE does not explicitly target at resistance against attacks, such as traffic analysis [50], side-channel attacks [51], and leakage of the hashed plaintext [52], that may be utilized by adversaries to violate users' privacy and data security. However, SPADE is well compatible with orthogonal techniques [8], [30] to thwart these attacks.

5.3 Further Discussion

There is an important difference between the servers-aided MLE to resist the brute-force attack and the servers-aided password-hardening protocol to resist DGA. In the servers-aided MLE, different users should correspond to the same server-side secret α, which enables the cloud server to perform deduplication across all its users. Therefore, in SPADE, for different MLE key requests made by different users, $\mathcal{KS}_i$ always utilizes α_i to compute the signature that is used to compute the MLE key.

However, in the servers-aided password-hardening protocol, if all users share the same password-hardening key, it is vulnerable to online DGA. Specifically, for a target user $\mathcal{U}^*$, an adversary $\mathcal{A}$ can enumerate all possible passwords $\{psw_1, psw_2, \ldots, psw_\varphi\}$. Then $\mathcal{A}$ is able to use these passwords to interact with all key servers to compute the corresponding servers-protected passwords.[3] As a consequence, $\mathcal{A}$ would obtain all possible servers-protected passwords $\{spsw_1, spsw_2, \ldots, spsw_\varphi\}$ and can impersonate $\mathcal{U}^*$ to execute either *MLEKeyGen* or *DataAccess*. Then, $\mathcal{A}$ tries decrypt the ciphertexts received from key servers using $\{spsw_1, spsw_2, \ldots, spsw_\varphi\}$. If the decryption succeeds, $\mathcal{A}$ can recover the correct password of $\mathcal{U}^*$. We notice that this attack does not require $\mathcal{A}$ to compromise any key server.

In order to ensure the security of SPADE against online DGA, a natural way is to have a distinct password-hardening key for every user [25]. However, this would cause significant storage costs on the key server side. To balance the tradeoff between security and efficiency, SPADE can leverage a hybrid mechanism [26] to resist online DGA: all users are grouped according to the registration time; a group of users shares a password-hardening key; different groups of users have distinct password-hardening keys. As a result, for $\mathcal{KS}_i$, its local storage has the form shown in Fig. 6, where every d users form a group. By doing so, the storage costs on the key server side can be reduced significantly.

In SPADE, the authentication credentials stored on key servers should be updated in an epoch. Specifically, in the first epoch, *Register* is executed by $\mathcal{U}$ and $\vec{\mathcal{KS}}^{(1)}$, where the authentication credential $cre_i^{(1)}$ is generated and stored on $\mathcal{KS}_i^{(1)}$; in subsequent epochs, *Proactivization* is executed, which requires $\mathcal{U}$ to interact with all key servers in the epoch to generate a new authentication credential for each of them.

We stress that the updating of authentication credentials in *Proactivization* is necessary for the security of SPADE. If the authentication credentials are not updated, SPADE is vulnerable to credential leakage attacks. Particularly, recall the handoff process in *Proactivization*, if the authentication

2. The Common Criteria. https://www.commoncriteriaportal.org/

3. To avoid the detection, $\mathcal{A}$ can interact with the key servers utilizing different identities.

Group Index	User Identity	Counter	Server-Side Secret Share	Password-Hardening Key Share	Authentication Credential
1	ID_{u_1}	ρ_{u_1}, cf_{u_1}	α_{i,u_1}	$\lambda_{i,1}$	cre_{i,u_1}
	ID_{u_2}	ρ_{u_2}, cf_{u_2}	α_{i,u_2}		cre_{i,u_2}
	...	...	...		...
2	$ID_{u_{d+1}}$	$\rho_{u_{d+1}}, cf_{u_{d+1}}$	$\alpha_{i,u_{d+1}}$	$\lambda_{i,2}$	$cre_{i,u_{d+1}}$
	$ID_{u_{d+2}}$	$\rho_{u_{d+2}}, cf_{u_{d+2}}$	$\alpha_{i,u_{d+2}}$		$cre_{i,u_{d+2}}$
	...	...	...		...
...	...	...	...	...	...

Fig. 6. $\mathcal{KS}_i$'s local storage form with leverage of the hybrid mechanism.

credentials are not updated in subsequent epochs, all key servers in subsequent epochs have to utilize $\{cre_{i_1}^{(1)}, cre_{i_2}^{(1)}, \ldots, cre_{i_t}^{(1)}\}$ as the authentication credentials to authenticate $\mathcal{U}$. It means that all key servers starting from the second epoch have the copy of $\{cre_{i_1}^{(1)}, cre_{i_2}^{(1)}, \ldots, cre_{i_t}^{(1)}\}$. As a consequence, if anyone of the key servers is compromised, all authentication credentials would be leaked, and the password-based authentication would be invalidated. On the other hand, *such an updating needs to be executed only once during one epoch*. Since the frequency at which *Proactivization* is executed is very low, the updating of authentication credentials would not introduce heavy communication costs on the user side.

5.4 Applications

We have designed SPADE that possesses theoretically desirable properties. However, we would like to emphasize the practical nature of SPADE: since there are some promising application scenarios, SPADE can be inexpensively implemented as described.

For general cloud storage systems that only provide the data access interface to their users, SPADE has a clear value, which has been elaborated in Introduction.

However, one may also argue the commercial incentive of users in SPADE, where the users have to bear the additional costs to employ independent key servers. We believe SPADE is also favorable for current mobile cloud storage systems. Specifically, data outsourcing has become the fundamental data management method in several mobile Apps. It not only provides mobile users an efficient way to manage their data, but also enables different mobile service providers to utilize users' data that are stored on the same cloud server. We assume there are n Apps, App_1, App_2, ..., App_n, which are subject to n mobile service providers, $\mathcal{MP}_1, \mathcal{MP}_2, \ldots, \mathcal{MP}_n$, respectively. All these Apps utilize the same cloud server (say $\mathcal{CS}$) to manage users' data and even share the user's data with each other (under the user's permission). In such an application scenario, the mobile service providers are independent of each other and are not fully trusted by users, which is within the threat model of SPADE. Hence, the service providers ($\mathcal{MP}_1, \mathcal{MP}_2, \ldots, \mathcal{MP}_n$) can serve as the key servers in SPADE to provide the password-hardening service and MLE-key generation service simultaneously, and the users do not need to bear additional costs to employ independent key servers.

In such an application scenario, compared with existing scheme, SPADE provides users a stronger protection of security in terms of their passwords and data without introducing additional costs; SPADE enables the cloud server to save storage space significantly and to authenticate users in a secure and convenient way; SPADE also frees mobile service providers from developing data manage module while retaining all functionalities. It motivates the users, mobile service providers, and cloud service provider to employ SPADE in reality.

6 Security Analysis

In the security analysis of SPADE, we would not consider adversaries who interfere with *Register*, which follows existing schemes [25], [26]. All communications in *Register* are well protected, this can be easily achieved by the way that a user establishes a TLS connection with the key servers and the cloud server to secure all messages that she/he needs to send at the beginning of the registration process. Moreover, hereinafter, we also assume that both the outsourced data and passwords are low-entropy, and an adversary is able to enumerate all possible candidates in an off-line manner. According to the threat model presented in Section 3.3, the security of SPADE is analyzed from two aspects.

6.1 Adversarial Cloud Server

We first identify the attacks that an adversarial cloud server $\mathcal{CS}^*$ might launch.

To violate the user's privacy, $\mathcal{CS}^*$ would either extract the contents of the outsourced file, or compromise the user's password. Notice that existing works [8], [16], [26] have proved that $\mathcal{CS}^*$ cannot succeed without colluding with key servers. Here, we consider $\mathcal{CS}^*$ who corrupts key servers to violate the user's privacy. $\mathcal{CS}^*$ now has the following two possible strategies if it has corrupted some key servers:

1) It performs the off-line brute-force attack to extract the contents of the outsourced file;
2) It performs the off-line DGA to retrieve the user's password.

For the strategy 1, SPADE resists $\mathcal{CS}^*$ in two aspects.

First, due to the adoption of the servers-aided MLE [8] (which is based on the (t, n)-threshold blind signature), SPADE is secure against off-line brute-force attacks launched by $\mathcal{CS}^*$ who can compromise up to $t-1$ key servers in an epoch.

Second, the security of SPADE does not rely on the reliability of a specific committee of key servers, due to the proposed proactivization mechanism. Specifically, SPADE leverages different key servers in different epochs. In the extreme case, $\mathcal{CS}^*$ can control $t-1$ key servers in the χth epoch and $t-1$ key servers in the $(\chi+1)$th epoch, such that it has $2t-2$ server-side secret shares from two epochs. For the sake of brevity, these $2t-2$ server-side secret shares are denoted by

$$\{\alpha_1^{(\chi)}, \alpha_2^{(\chi)}, \ldots, \alpha_{t-1}^{(\chi)}, \alpha_1^{(\chi+1)}, \alpha_2^{(\chi+1)}, \ldots, \alpha_{t-1}^{(\chi+1)}\}.$$

To launch the brute-force attack, $\mathcal{CS}^*$ has to enumerate all possible file candidates and computes the corresponding

MLE keys. To this end, given a file candidate M^*, $\mathcal{CS}^*$ needs to computes $\sigma_* = \alpha \cdot H(M^*)$, where α is the server-side secret generated in *Setup*. Recall Equation (9), $\mathcal{CS}^*$ can obtain

$$\sigma^* = \sum_{i=1}^{t-1} w_i \alpha_i^{(\chi)} \cdot H(M^*) + w_t \alpha_t^{(\chi)} \cdot H(M^*), \tag{20}$$

$$\sigma^* = \sum_{i=1}^{t-1} w_i' \alpha_i^{(\chi+1)} \cdot H(M^*) + w_t' \alpha_t^{(\chi+1)} \cdot H(M^*). \tag{21}$$

We notice that there are three unknown elements, i.e., $\alpha_t^{(\chi)}$, $\alpha_t^{(\chi+1)}$, and σ^* but only two equations, and $\mathcal{CS}^*$ can compute infinitely many solutions for these equations. Hence, the offline brute-force attacks launched by $\mathcal{CS}^*$ who compromises key servers cannot work, and the strategy 1 is infeasible.

For the strategy 2, SPADE also resists $\mathcal{CS}^*$ in two aspects which are similar to those that resist the brute-force attack. Since the security against DGA achieved by the servers-aided password-hardening protocol has the essentially same construction as the security of servers-aided MLE against the brute-force attack. It is easy to prove that $\mathcal{CS}^*$ cannot retrieve a target user's password by performing DGA, even if it compromises up to $t-1$ key servers in an epoch.

6.2 Compromised Key Server(s)

We first identify the attacks that a compromised key server $\mathcal{KS}^*$ might perform. Hereinafter, we assume that $\mathcal{KS}^*$ is able to collude with up to $t^{'}(t^{'} < t-1)$ key servers in the current epoch.

In SPADE, the goal of $\mathcal{KS}^*$ is to violate a target user $\mathcal{U}^*$'s privacy, which can be classified into two types:

- Compromising the confidentiality of the data outsourced by $\mathcal{U}^*$;
- Impersonating $\mathcal{U}^*$ to pass the authentication of other key servers and the cloud server or compromising $\mathcal{U}^*$'s password $psw_{\mathcal{U}^*}$.

In regards to the first goal, $\mathcal{KS}^*$ has the following two strategies.

1) It extracts the information on the file (e.g., the hashed file) from the messages received from $\mathcal{U}^*$;
2) It stops counting the number of MLE keys made by an adversary (e.g., the adversarial cloud server).

We prove that $\mathcal{KS}^*$ who utilizes the first strategy cannot obtain any information about the file protected by SPADE, which is formalized as a property of obliviousness.

Lemma 1. *Obliviousness: In SPADE, the interactions between a user and a key server are oblivious, which ensures that when the user requests an MLE key on a file M^* from $\mathcal{KS}^*$, the information on M^* that is extracted by $\mathcal{KS}^*$ would not be more than that extracted from a random string with the same length.*

Proof. We define an obliviousness game between an environment $\mathcal{E}$, $\mathcal{KS}^*$, and a simulator $\mathcal{S}$ as follows.

Obliviousness game.

1) $\mathcal{E}$ initiates SPADE, generates a server-side secret α and its n shares $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ (these shares are generated by using the (t,n)-threshold secret sharing protocol described in Section 5.1). $\mathcal{E}$ generates public parameters and sends them to $\mathcal{KS}^*$ and $\mathcal{S}$.
2) $\mathcal{KS}^*$ randomly chooses a file $M^* \in Z_p$ and sends M^* to $\mathcal{S}$. $\mathcal{S}$ forwards M^* to $\mathcal{E}$.
3) $\mathcal{E}$ randomly selects $r^* \in Z_p^*$, computes $\sigma_{M^*} = r^* \cdot \alpha \cdot H_2(M^*)$, and sends $mk_{M^*} = h_1(\sigma_{M^*})$ to $\mathcal{S}$. $\mathcal{S}$ sends mk_{M^*} to $\mathcal{KS}^*$.
4) $\mathcal{KS}^*$ repeats the above two steps at most $poly(\ell)$ times. Then $\mathcal{KS}^*$ randomly selects $M_0 \in Z_p$, $M_1 \in Z_p$ and sends M_0 and M_1 to $\mathcal{S}$. $\mathcal{S}$ forwards them to $\mathcal{E}$.
5) $\mathcal{E}$ randomly selects $\varsigma \in \{0,1\}$, $r^* \in Z_p^*$. Then
   - if $\lambda = 0$, it computes $C_\varsigma = \mathsf{E}(mk_{M_\varsigma}, M_\varsigma)$, where $mk_{M_\varsigma} = h_1(r^* \cdot \alpha \cdot H_2(M_\varsigma))$;
   - otherwise, it randomly selects C_ς from $[0, |\mathsf{E}(mk_{M_\varsigma}, M_\varsigma)|]$.

   $\mathcal{E}$ sends C_ς to $\mathcal{S}$. $\mathcal{S}$ sends C_ς to $\mathcal{KS}^*$.
6) $\mathcal{KS}^*$ outputs ς^*, and wins if and only if $\varsigma^* = \varsigma$.

If $\mathcal{KS}^*$ wins the game with the probability of $\Pr_{\mathcal{KS}^*}^{Obli,\ell}$, $\mathcal{S}$ is able to break the D-IND$-CPA security of the underlying MLE with the same probability (Refs. [8], [18] have proven that servers-aided MLE is D-IND$-CPA secure). Specifically, if $\mathcal{KS}^*$ outputs ς^* and wins the obliviousness game, $\mathcal{S}$ outputs the same bit to distinguish the ciphertext of MLE and a random $|\mathsf{E}(mk_{M_\varsigma}, M_\varsigma)|$-bit string. This concludes the proof. □

In addition, if $\mathcal{KS}^*$ is able to compromise the confidentiality of the outsourced data by utilizing the second strategy, it essentially contributes to the online brute-force attack. However, as described in Section 5.1, each MLE key request made by a user would be sent to all key servers, and an adversary can obtain the MLE key on a file only if he collects at least t signatures computed on t distinct server-side secret shares. Thus, the second strategy cannot help $\mathcal{KS}^*$ in performing online brute-force attacks.

Regarding to the second goal, i.e., $\mathcal{KS}^*$ performs the impersonation attack, we prove that $\mathcal{KS}^*$ nether can impersonate $\mathcal{U}^*$, nor can compromise the $\mathcal{U}^*$'s password $psw_{\mathcal{U}^*}$. In SPADE, the security against impersonation attacks and compromising password is ensured by the security of the servers-protected password $spsw_{\mathcal{U}^*}$: If $\mathcal{KS}^*$ cannot forge a valid $spsw_{\mathcal{U}^*}$, it cannot impersonate $\mathcal{U}^*$; if $\mathcal{KS}^*$ cannot extract any information about $psw_{\mathcal{U}^*}$ from the interactions that it assists $\mathcal{U}^*$ in generating $spsw_{\mathcal{U}^*}$, it cannot retrieve $psw_{\mathcal{U}^*}$.

The security of $spsw_{\mathcal{U}^*}$ is captured by two security notions: *Unforgeability* and *Obliviousness*.

Lemma 2. *Unforgeability: In SPADE, given a target user $\mathcal{U}^*$'s password $psw_{\mathcal{U}^*}$, $\mathcal{KS}^*$ cannot forge a valid servers-protected password $spsw_{\mathcal{U}^*}$, even if it colludes with up to $t^{'}(t^{'} < t-1)$ key servers in the current epoch.*

Proof. We define an Unforgeability game between an environment $\mathcal{E}$, $\mathcal{KS}^*$, and a simulator $\mathcal{S}$ as follows.

Unforgeability game.

1) $\mathcal{E}$ initiates SPADE, generates a password-hardening key λ and its n shares $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ (these shares are generated by using the (t,n)-threshold

secret sharing protocol described in Section 5.1). $\mathcal{E}$ generates public parameters and sends them to $\mathcal{KS}^*$ and $\mathcal{S}$.

2) $\mathcal{KS}^*$ selects a set of indexes $T^{'} = \{i_1, i_2, \ldots, i_{t'}\}$ and sends $T^{'}$ to $\mathcal{S}$. $\mathcal{S}$ sends $T^{'}$ to $\mathcal{E}$. $\mathcal{E}$ sends $\{\lambda_{i_1}, \lambda_{i_2}, \ldots, \lambda_{i_{t'}}\}$ to $\mathcal{S}$. $\mathcal{S}$ forwards $\{\lambda_{i_1}, \lambda_{i_2}, \ldots, \lambda_{i_{t'}}\}$ to $\mathcal{KS}^*$ and initializes $q_1 = 0$, $q_2 = 0, \ldots, q_n = 0$.
3) $\mathcal{KS}^*$ randomly selects $psw^*_{\mathcal{U}^*} \in Z^*_p$ and $r^* \in Z^*_p$ and computes $p\tilde{s}w^*_{\mathcal{U}^*} = r^* \cdot H_1(psw^*)$.
4) For $i \in [1, n]$ and $i \notin T^{'}$, $\mathcal{KS}^*$ submits a query of $\delta^*_i = \lambda_i \cdot p\tilde{s}w^*_{\mathcal{U}^*}$ to $\mathcal{S}$.
5) After receiving the query submitted by $\mathcal{KS}^*$, $\mathcal{S}$ sets $q_i = q_i + 1$ and forwards the query to $\mathcal{E}$.
6) $\mathcal{E}$ computes δ^*_i using λ_i and sends δ^*_i to $\mathcal{S}$, $\mathcal{S}$ forwards δ^*_i to $\mathcal{KS}^*$.
7) $\mathcal{KS}^*$ repeats the above steps trying different i. Finally, $\mathcal{KS}^*$ outputs δ^*.
8) $\mathcal{KS}^*$ wins, if and only if the following two conditions hold:
   - $\sum_{i=1}^{n} q_i < t - t^{'}$;
   - δ^* is a valid signature on $p\tilde{s}w^*_{\mathcal{U}^*}$ under V (which is the public key corresponding to λ).

SPADE is built on the BLS signature [47] which is existentially unforgeable. However, in the above Unforgeability game, if $\mathcal{KS}^*$ wins the probability of $\Pr^{Unf,\ell}_{\mathcal{KS}^*}$, $\mathcal{S}$ is able to forge a BLS signature with the same probability. We prove this as follows.

Before $\mathcal{KS}^*$ outputs δ^*, $\mathcal{S}$ is able to collect $\{\lambda_{i_1}, \delta^*_{i_1}, \lambda_{i_2}, \delta^*_{i_2}, \ldots, \lambda_{i_{t'}}, \delta^*_{i_{t'}}\}$, $\{\delta_i\}$ ($i \in [1, n]$ and $i \notin T^{'}$), and δ^*.

Here, we stress that for $\mathcal{KS}^*$, the most effective way to forge δ^* under the threat model is to let $\sum_{i=1}^{n} q_i + t^{'} = t - 1$, no matter what attack it would launch. Therefore, after $\mathcal{KS}^*$ outputs δ^*, $\mathcal{S}$ has $t-1$ signatures under $t-1$ password-hardening key shares of the same epoch. Therefore, $\mathcal{S}$ can output δ^*_t as a forged signature, where δ^*_t is a valid signature on $p\tilde{s}w^*_{\mathcal{U}^*}$ under λ_t. □

Lemma 3. *Obliviousness: In SPADE, for a compromised key server $\mathcal{KS}^*$, the interactions between it and a target user $\mathcal{U}^*$ to generate the servers-protected password $spsw_{\mathcal{U}^*}$ would not leak any information about the password $psw_{\mathcal{U}^*}$, even if $\mathcal{KS}^*$ is able to know $spsw_{\mathcal{U}^*}$.*

Proof. This proof is similar to the proof of Lemma 1. We first define an Obliviousness game between an environment $\mathcal{E}$, $\mathcal{KS}^*$, and a simulator $\mathcal{S}$ as follows. To avoid ambiguity, this Obliviousness game is named "Pass-Obliviousness" game.

Pass-Obliviousness game.

1) $\mathcal{E}$ initiates SPADE, generates a password-hardening key λ and its n shares $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ (these shares are generated by using the (t, n)-threshold secret sharing protocol described in Section 5.1). $\mathcal{E}$ generates public parameters and sends them to $\mathcal{KS}^*$ and $\mathcal{S}$.
2) For the target user $\mathcal{U}^*$, $\mathcal{KS}^*$ randomly chooses a password $psw^*_{\mathcal{U}^*} \in Z_p$ and sends $psw^*_{\mathcal{U}^*}$ to $\mathcal{S}$. $\mathcal{S}$ forwards $psw^*_{\mathcal{U}^*}$ to $\mathcal{E}$.
(3) $\mathcal{E}$ randomly selects $r^* \in Z^*_p$, computes $\delta^* = r^* \cdot \lambda \cdot H_1(psw^*_{\mathcal{U}^*})$, and sends $spsw_{\mathcal{U}^*} = h(\delta^* || psw^*_{\mathcal{U}^*})$ to $\mathcal{S}$. $\mathcal{S}$ sends $spsw_{\mathcal{U}^*}$ to $\mathcal{KS}^*$.
4) $\mathcal{KS}^*$ repeats the above two steps at most $poly(\ell)$ times. Then $\mathcal{KS}^*$ randomly selects $psw_0 \in Z_p$, $psw_1 \in Z_p$ and sends psw_0 and psw_1 to $\mathcal{S}$. $\mathcal{S}$ forwards them to $\mathcal{E}$.
5) $\mathcal{E}$ randomly selects $\varsigma \in \{0, 1\}$, $r^* \in Z^*_p$. Then
   - if $\lambda = 0$, it computes $spsw_\varsigma = h(\lambda \cdot r^* \cdot H_1(psw_\varsigma) || psw_\varsigma)$;
   - otherwise, it uniformly selects a string $spsw_\varsigma \in \{0, 1\}^\ell$.

   $\mathcal{E}$ sends $spsw_\varsigma$ to $\mathcal{S}$. $\mathcal{S}$ sends $spsw_\varsigma$ to $\mathcal{KS}^*$.
6) $\mathcal{KS}^*$ outputs ς^*, and wins if and only if $\varsigma^* = \varsigma$.

In the Pass-Obliviousness game, if the advantage that $\mathcal{KS}^*$ wins is $\Pr^{Pas-Obli,\ell}_{\mathcal{KS}^*}$, then $\mathcal{S}$ is able to compromise the security of the hash function with a non-negligible probability. This concludes the proof. □

As a summary, SPADE is secure against the adversarial cloud server and compromised key servers, and its security does not relay on a specific group of key servers in the lifecycle of the system.

7 Implementation and Evaluation

We implement SPADE by using C++ language and MIRACL library 5.6.1 version. To evaluate the performance, we implement several settings described as follows. The experiments are run on a laptop with macOS system, an Intel Core i7 CPU, and 16 GB DDR3 of RAM. We further evaluate the performance of the algorithms that are executed by the user on a smartphone (HUAWEI MT2-L01, with Android 4.2.2 system and a Kirin 910 CPU, and 1596 MHz frequency) to demonstrate that SPADE is highly efficient on the user side. The security level is set to 80 bits. The underlying curve is the MNT curve.[4]

In this section, we elaborate on the implementation of SPADE using functions and provide the corresponding performance evaluations. For clarity, we prefix calls of functions with U if they are made by the user, with KS if they are made by the key server, and with CS if they are made by the cloud server. We assume that all public parameters have been distributed among all entities and would not explicitly show *Setup* in the following, for the sake of brevity.

7.1 Implementation and Evaluation of Register

A schematic of *Register* implementation is depicted in Fig. 7. In *Register*, a user $\mathcal{U}$, whose identity is $ID_{\mathcal{U}}$, selects a human-memorable password $psw_{\mathcal{U}}$ and invokes $Random()$ to generate a random element. Then, $\mathcal{U}$ blinds $psw_{\mathcal{U}}$ to obtain $psw^*_{\mathcal{U}}$ and establishes a TLS connection with each key server such that subsequent communications can be well protected. Next, $\mathcal{U}$ sends $\{ID_{\mathcal{U}}, psw^*_{\mathcal{U}}\}$ to each key server. Upon receiving $\{ID_{\mathcal{U}}, psw^*_{\mathcal{U}}\}$, each key server (say $\mathcal{KS}_i$) interacts with other key servers to jointly execute DSGS by invoking $DSGS()$ shown in Algorithm 1, which enables key servers to generate and share a password-hardening key λ for $\mathcal{U}$ and allows each of them to obtain a password-hardening

4. In Section 5.1, SPADE is constructed on the Type-1 pairing. However, the MNT curve is utilized to construct Type-3 pairing. We stress that it would not impact the feasibility, correctness, and security of SPADE when we construct SPADE on the Type-3 pairing.

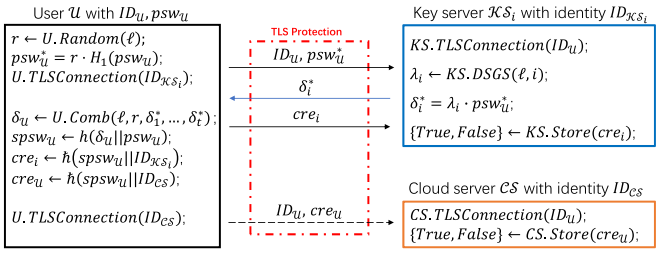

Fig. 7. Implementation of *Register*.

key share. $\mathcal{KS}_i$ signs $psw_{\mathcal{U}}^*$ using λ and responses $\mathcal{U}$ with the signature. With a threshold number of valid signatures, $\mathcal{U}$ invokes $Comb()$ to combine these signatures with the random element r to retrieve the signature (i.e., $\delta_{\mathcal{U}}$) on $psw_{\mathcal{U}}$ under λ. With $\delta_{\mathcal{U}}$, $\mathcal{U}$ computes an authentication credential for each key server and sends it via the established TLS connection. Finally, $\mathcal{U}$ establishes a TLS connection with the cloud server $\mathcal{CS}$ and sends her/his identity and the authentication credential (i.e., $\{ID_{\mathcal{U}}, cre_{\mathcal{U}}\}$) to $\mathcal{CS}$. After the key servers and the cloud server securely store the identity and the authentication credential of the user, *Register* is completed.

We evaluate the performance of *Register* in terms of communication and computation.

The communication costs include the ones between $\mathcal{U}$ and $\mathcal{KS}_i$ and the ones between $\mathcal{U}$ and $\mathcal{CS}$, which only takes hundreds of bytes in our setting (actually, it takes within 200 bytes on $\mathcal{U}$ and within 100 bytes on $\mathcal{CS}$ for one user). We notice that in *Register*, key servers need to communicate with each other to generate the password-hardening key. The communication costs on the key server are shown in Fig. 8, where we only consider the single user case. As discussed in Section 5.3, in reality, the communication costs can be amortized over multiple users who share the same password-hardening key.

The computational costs on $\mathcal{U}$ consist of blinding the password, computing the servers-protected password, and computing authentication credentials, which increases with the total number of key servers (i.e., n) and the threshold number t. The computational costs on $\mathcal{KS}_i$ consist of executing $DSGS$ and generating a signature. For one user, the computational costs on $\mathcal{KS}_i$ depend on n and t, and the computational costs on $\mathcal{CS}$ are constant. In Fig. 9, we show the computation delay on $\mathcal{U}$ and $\mathcal{KS}_i$, where we fix $n = 30$.

7.2 Implementation and Evaluation of MLEKeyGen

A schematic of *MLEKeyGen* implementation is depicted in Fig. 10. In *MLEKeyGen*, to request an MLE key from key

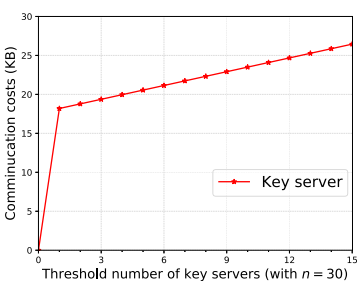

Fig. 8. Communication costs on $\mathcal{KS}_i$ in *Register*.

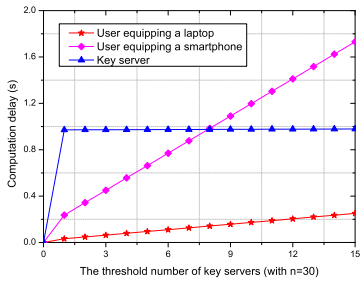

Fig. 9. Computation delay on $\mathcal{U}$ and $\mathcal{KS}_i$ in *Register*.

servers, $\mathcal{U}$ needs to authenticate herself/himself with them. To this end, $\mathcal{U}$ first interacts with key servers to recover the servers-protected password $spsw_{\mathcal{U}}$, which requires two rounds of communications. With $spsw_{\mathcal{U}}$, $\mathcal{U}$ can compute the authentication credential. $\mathcal{U}$ also generates a file M (by invoking $GenFile$) that would be outsourced to $\mathcal{CS}$, blinds M using a random element to obtain M^*, and sends M^* (in the ciphertext form) to $\mathcal{KS}_i$. $\mathcal{KS}_i$ computes the corresponding signature σ_i^* using the server-side key share α_i and responses $\mathcal{U}$ with encrypted σ_i^*. With t valid σ_i^*, $\mathcal{U}$ can retrieve the signature on M under the server-side secret α by invoking $Comb()$. We notice that in SPADE, since both $\mathcal{U}$ and $\mathcal{KS}_i$ utilize the authentication credential as the encryption/decryption key, the authentication between $\mathcal{U}$ and $\mathcal{KS}_i$ and the MLE key generation can be completed in two rounds of communications, which is highly efficient in terms of communication costs.

Similarly, we evaluate the performance of *MLEKeyGen* in terms of communication and computation.

The communication costs on $\mathcal{U}$ in *MLEKeyGen* are slightly more than those in *Register*, since $\mathcal{U}$ needs to receive an additional random element, which protects SPADE from reply attacks. However, we stress that the communications between $\mathcal{U}$ and key servers in *MLEKeyGen* are under public channels, the user does not need to establish a TLS connection with a key server. On the other hand, the communication costs on the key server in *MLEKeyGen* are much smaller than those in *Register*, since key servers do not need to communicate with each other. As a result, in *MLEKeyGen*, it only takes within 200 bytes on both $\mathcal{U}$ and $\mathcal{KS}_i$.

The computation costs on $\mathcal{U}$ consist of three parts: retrieving the servers-protected password, recovering the authentication credentials, and computing the MLE key, which increases with n and t. For one user, the computation costs on $\mathcal{KS}_i$ are constant. In our experiment, the computation

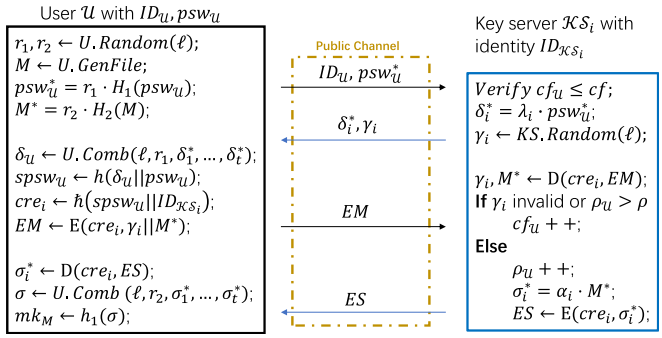

Fig. 10. Implementation of *MLEKeyGen*.

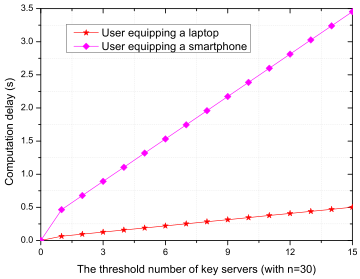

Fig. 11. Computation delay on $\mathcal{U}$ in *MLEKeyGen*.

delay on the key server is within 3 ms. In Fig. 11, we show the computation delay on the user side in *MLEKeyGen*, where we set $n = 30$.

7.3 Implementation and Evaluation of Deduplication

A schematic of *Deduplication* implementation is depicted in Fig. 13. In *Deduplication*, $\mathcal{U}$ first computes the authentication credential $cre_{\mathcal{U}}$ to authenticate herself/himself with $\mathcal{CS}$. With the correct $cre_{\mathcal{U}}$, a secure channel between $\mathcal{U}$ and $\mathcal{CS}$ can be established. Then, $\mathcal{U}$ encrypts M with the MLE key mk_M to obtain the encrypted file C_M and encrypts the MLE key mk_M with a key $\kappa_{\mathcal{U}}$ derived from $spsw_{\mathcal{U}}$ to obtain the encrypted MLE key $C^{\mathcal{U}}_{mk_M}$. Finally, $\mathcal{U}$ outsources $\{C_M, C^{\mathcal{U}}_{mk_M}\}$ to $\mathcal{CS}$. The latter performs deduplication over C_M and well maintains $C^{\mathcal{U}}_{mk_M}$.

In *Deduplication*, the communication costs on $\mathcal{U}$ and $\mathcal{CS}$ mainly depend on $|M|$ (i.e., the size of the file to be outsourced by the user). Compared with $|M|$, communication costs caused by sending/receiving other parameters can be negligible.

The computation costs on $\mathcal{U}$ consist of encrypting M and encrypting the MLE key. For $|M| = 1$ GB, the computation delay is within 11 seconds; For $|M| = 10$ GB, the computation delay is within 210 seconds. The computation costs on $\mathcal{CS}$ mainly depend on the underlying deduplication strategy and could also be impacted by the number of files that have been stored.

7.4 Implementation and Evaluation of DataAccess

A schematic of *DataAccess* implementation is depicted in Fig. 14. In *DataAccess*, $\mathcal{U}$ first interacts with key servers to retrieve the servers-protected password $spsw_{\mathcal{U}}$, which enables $\mathcal{U}$ to authenticate herself/himself with both the key servers and the cloud server. Then, $\mathcal{U}$ can download the ciphertext of the target data (i.e., C_M) and the ciphertext of the corresponding MLE key (i.e., $C^{\mathcal{U}}_{mk_M}$) from $\mathcal{CS}$. With $spsw_{\mathcal{U}}$, $\mathcal{U}$ can decrypt $C^{\mathcal{U}}_{mk_M}$ to obtain mk_M and further decrypts C_M with mk_M to retrieve the file M.

In *DataAccess*, the communication costs on the user and the cloud server are mainly caused by receiving/sending data. The communication costs on the key server are slightly less than those in *MLEKeyGen*.

We mainly focus on the computation costs on each entity. For $\mathcal{U}$, the computation costs consist of retrieving the servers-protected password, computing authentication credentials, and recovering the target file. If $|M|$ is more than 100 MB, the computation delay to decrypt the file would be much more than that to compute other parameters. To demonstrate the high efficiency of SPADE, we set $|M| = 10$ MB to evaluate the computation delay on $\mathcal{U}$ and provide the experiment results in Fig. 12.

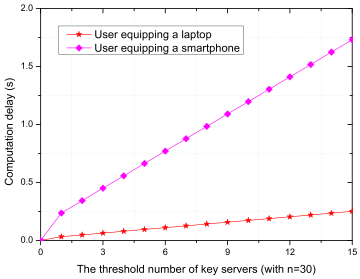

Fig. 12. Computation delay on $\mathcal{U}$ in *DataAccess*.

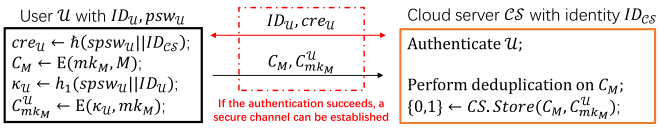

Fig. 13. Implementation of *Deduplication*.

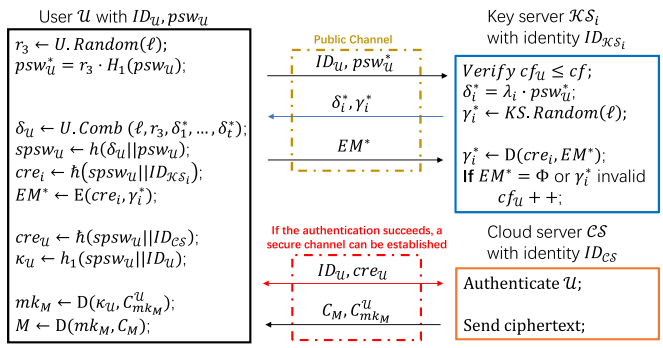

Fig. 14. Implementation of *DataAccess*.

7.5 Implementation and Evaluation of Proactivization

A schematic of *Proactivization* implementation is depicted in Fig. 15. In *Proactivization*, the handoff algorithm is jointly executed between t honest key servers in the χth epoch (say $\mathcal{KS}_i^{(\chi)}$ for $i = 1, 2, \ldots, t$) and n key servers in the $(\chi+1)$th epoch (i.e., $\mathcal{KS}_j^{(\chi+1)}$ for $j = 1, 2, \ldots, n$), such that α and λ can be re-distributed among $\mathcal{KS}_j^{(\chi+1)}$. Then, $\mathcal{U}$ interacts with $\mathcal{KS}_j^{(\chi+1)}$ to update her/his authentication credential.

In *Proactivization*, t key servers (say $\mathcal{KS}_i^{(\chi)}$) in the current epoch and all key servers (say $\mathcal{KS}_j^{(\chi+1)}$) need to participate in the handoff process to transfer α and λ. The communication costs on $\mathcal{KS}_i^{(\chi)}$ are different from those on $\mathcal{KS}_j^{(\chi+1)}$, and

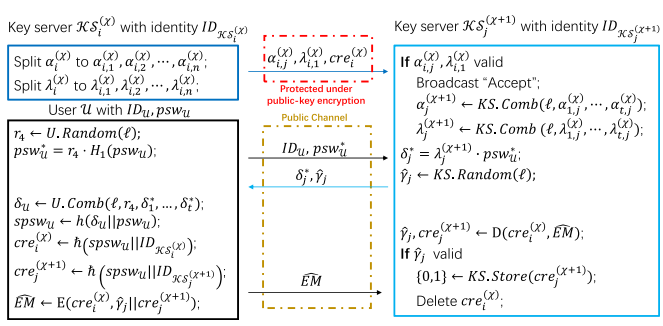

Fig. 15. Implementation of *Proactivization*.

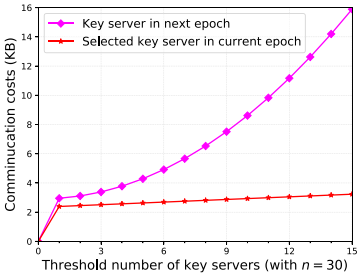

Fig. 16. Communication costs on key servers in *Proactivization*.

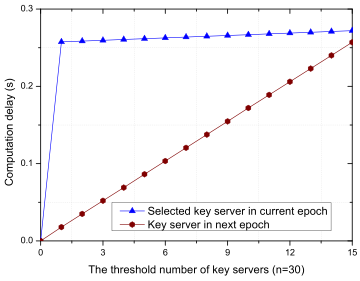

Fig. 17. Computation delay on key servers in *Proactivization*.

the experiment results are provided in Fig. 16. The communication costs on $\mathcal{U}$ are slightly less than that in *MLEKeyGen*.

The computation costs on $\mathcal{KS}_i^{(\chi)}$ are also different from those on $\mathcal{KS}_j^{(\chi+1)}$, which is shown in Fig. 17. The computation costs on $\mathcal{U}$ are slightly more than those in *Register*, since $\mathcal{U}$ only needs to additionally compute some hash values to update authentication credentials.

Now, we compare SPADE with existing schemes, e.g., DupLess [16], D14 [18], Dekey [24], and LAP15 [35] in terms of functionality, security, and efficiency. The comparison is provided in Table 1, where "Y" denotes that the scheme has the property, "N" denotes that the scheme does not have the property, "$\perp$" denotes the scheme does not focus on the property, and the experiments are tested on the laptop described before with the same security level ($n = 30, t = 15$, and the file size $|M| = 1$ MB). In Table 1 "BFA" refers to "brute-force attacks", "SPF" refers to "single point of failure", "KMP" refers to "key management problem", "PS" refers to "proactive security", and "MKG" refers to "MLE key generation". For LAP15, c refers to the number of "checkers" (the experiment results are evaluated under $c = 10$). Moreover, in Table 1, $\mathbb{N}$ denotes the RSA module; for a set S, H_S, Exp_S, Mul_S, Add_S denote "hashing a value into S", "exponentiation in S", "multiplication in S", "addition in S", respectively

In SPADE, as well as the existing schemes, the computation delay to generate an MLE key on the user who equips a laptop is within 1 second, which can be negligible, compared with the computation delay to encrypt some data in reality.

As a summary, from the performance evaluation, we observe that SPADE is efficient in terms of communication and computation costs. Specifically, a user, who only equips a (not-so-powerful) smartphone, can complete all required operations in SPADE with a tolerant delay and can easily access her/his outsourced files without maintaining any secret except her/his password. The handoff process of SPADE only requires key servers to bear slight communication and computation costs. With the security analysis, such a proactivization mechanism surely enhances the security without introducing heavy costs on key servers and users. Therefore, SPADE is easily to be deployed and provides a strong security guarantee.

8 Conclusion and Future Work

In this paper, we have proposed a secure password-protected MLE key scheme for deduplicated cloud storage systems, namely SPADE. SPADE is based on two key techniques: the first one is the servers-aided MLE with a proactivization mechanism which is secure against the brute-force attack without trusting a specific group of key servers during the lifetime of protected data; the second one is the servers-aided password-hardening protocol which is robust against the dictionary guessing attack while remaining functionalities of password and is compatible with the proposed proactivization mechanism. We have proved the security of SPADE against various attacks and have demonstrated that SPADE is efficient in terms of communication and computation costs.

For our future work, we will investigate how to achieve the same functionalities of SPADE with the same security guarantee without independent key servers. We notice that there is an alternative method proposed in [35] to resist brute-force attacks without key servers. However, the proposed password-hardening mechanism is not compatible with the scheme in [35] to address the key management problem, which needs to be further explored. We also notice that one can utilize the key techniques of SPADE in other scenarios to resist similar attacks. Specifically, public-key encryption with keyword search (PEKS) is inherently vulnerable to the keyword guessing attack. We notice that the key components of SPADE is well compatible with any

TABLE 1
Comparison

	DupLess [16]	D14 [18]	Dekey [24]	LAP15 [35]	SPADE
Security against BFA	Y	Y	N	Y	Y
No SPF	N	Y	$\perp$	Y	Y
No KMP	N	N	Y	N	Y
PS	N	N	N	$\perp$	Y
MKG overhead	$O(1)$	$O(t)$	$O(1)$	$O(c)$	$O(t)$
MKG computation costs	$2H_{Z_\mathbb{N}} + 2Exp_{Z_\mathbb{N}} + 2Mul_{Z_\mathbb{N}}$	$2H_{Z_\mathbb{N}} + (t+2)Exp_{Z_\mathbb{N}} + (t+2)Mul_{Z_\mathbb{N}} + (t-1)Add_{Z_\mathbb{N}}$	H_{Z_p}	$2H_{Z_p} + (7c+3)Mul_G + (3c+2)Add_G + cAdd_{Z_p}$	$H_G + (t+2)Mul_G + 4Pair + tAdd_G$
MKG delay	< 12 ms	< 20 ms	< 5 ms	< 42 ms	< 45 ms

PEKS scheme to protect keywords from the guessing attack without the single-point-of-failure problem, which would provide a stronger security guarantee compared with existing schemes [19], [53], [54].

Acknowledgments

This work was supported in part by the National Nature Science Foundation of China under Grants 62002050, 61872060, and 61902327, in part by the Key Research and Development Program of Sichuan under Grant 2021YFG0158, and in part by Central University Basic Research Funds Foundation under Grant A030202063008083. A preliminary version [1] of this article was presented at the 2019 IEEE Global Communications Conference (GLOBECOM' 19).

References

[1] Y. Zhang, C. Xu, N. Cheng, and X. Shen, "Secure encrypted data deduplication for cloud storage against compromised key servers," in *Proc. IEEE Glob. Commun. Conf.*, 2019, pp. 1–6.

[2] J. Liang, Z. Qin, S. Xiao, L. Ou, and X. Lin, "Efficient and secure decision tree classification for cloud-assisted online diagnosis services," *IEEE Trans. Dependable Secur. Comput.*, to be published, doi: 10.1109/TDSC.2019.2922958.

[3] K. Xue, N. Gai, J. Hong, D. Wei, P. Hong, and N. Yu, "Efficient and secure attribute-based access control with identical sub-policies frequently used in cloud storage," *IEEE Trans. Dependable Secur. Comput.*, to be published, doi: 10.1109/TDSC.2020.2987903.

[4] M. Li, L. Zhu, and X. Lin, "Privacy-preserving traffic monitoring with false report filtering via fog-assisted vehicular crowdsensing," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2019.2903060.

[5] Y. Zhang, C. Xu, X. Lin, and X. Shen, "Blockchain-based public integrity verification for cloud storage against procrastinating auditors," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2019.2908400.

[6] X. Zhang, H. Wang, and C. Xu, "Identity-based key-exposure resilient cloud storage public auditing scheme from lattices," *Inf. Sci.*, vol. 472, pp. 223–234, 2019.

[7] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Trans. Storage*, vol. 7, no. 4, pp. 1–20, 2012.

[8] Y. Zhang, C. Xu, H. Li, K. Yang, J. Zhou, and X. Lin, "HealthDep: An efficient and secure deduplication scheme for cloud-assisted eHealth systems," *IEEE Trans. Ind. Inform.*, vol. 14, no. 9, pp. 4101–4112, Sep. 2018.

[9] J. Stanek and L. Kencl, "Enhanced secure thresholded data deduplication scheme for cloud storage," *IEEE Trans. Dependable Secur. Comput.*, vol. 15, no. 4, pp. 694–707, Jul./Aug. 2018.

[10] H. Li, Y. Yang, T. H. Luan, X. Liang, L. Zhou, and X. Shen, "Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data," *IEEE Trans. Dependable Secur. Comput.*, vol. 13, no. 3, pp. 312–325, May/Jun. 2016.

[11] J. Liang, Z. Qin, S. Xiao, J. Zhang, H. Yin, and K. Li, "Privacy-preserving range query over multi-source electronic health records in public clouds," *J. Parallel Distrib. Comput.*, vol. 135, pp. 127–139, 2020.

[12] Y. Miao *et al.*, "Privacy-preserving attribute-based keyword search in shared multi-owner setting," *IEEE Trans. Dependable Secur. Comput.*, to be published, doi: 10.1109/TDSC.2019.2897675.

[13] Y. Miao, R. Deng, X. Liu, K. R. Choo, H. Wu, and H. Li, "Multi-authority attribute-based keyword search over encrypted cloud data," *IEEE Trans. Dependable Secur. Comput.*, to be published, 2019, doi: 10.1109/TDSC.2019.2935044.

[14] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming space from deplicate files in a serverless distributed file ststem," in *Proc. IEEE 22nd Int. Conf. Distrib. Comput. Syst.*, 2002, pp. 617–624.

[15] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn. Adv. Crypt.*, 2013, pp. 296–312.

[16] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Dupless: Server-aided encryption for deduplicated storage," in *Proc. USENIX Secur. Symp.*, 2013, pp. 179–194.

[17] D. Chaum, "Blind signatures for untraceable payments," in *Proc. Adv. Cryptol.*, 1983, pp. 199–203.

[18] Y. Duan, "Distributed key generation for encrypted deduplication achieving the strongest privacy," in *Proc. ACM 6th Ed. ACM Workshop Cloud Comput. Secur.*, 2014, pp. 57–68.

[19] Y. Zhang, C. Xu, J. Ni, H. Li, and X. Shen, "Blockchain-assisted public-key encryption with keyword search against keyword guessing attacks for cloud storage," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2019.2923222.

[20] H. Li, X. Lin, H. Yang, X. Liang, R. Lu, and X. Shen, "EPPDR: An efficient privacy-preserving demand response scheme with adaptive key evolution in smart grid," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 8, pp. 2053–2064, Aug. 2014.

[21] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Proc. Annu. Int. Cryptology Conf. Adv. Cryptogr.*, 1995, pp. 339–352.

[22] A. Yang, J. Xu, J. Weng, J. Zhou, and D. S. Wong, "Lightweight and privacy-preserving delegatable proofs of storage with data dynamics in cloud storage," *IEEE Trans. Cloud Comput.*, vol. 9, no. 1, pp. 212–225, Jan.–Mar. 2021.

[23] T. M. Wong, C. Wang, and J. M. Wing, "Verifiable secret redistribution for archive systems," in *Proc. 1st Int. IEEE Secur. Storage Workshop*, 2002, pp. 94–105.

[24] J. Li, X. Chen, M. Li, J. Li, P. P. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1615–1625, Jun. 2014.

[25] S. Agrawal, P. Miao, P. Mohassel, and P. Mukherjee, "PASTA: PASsword-based threshold authentication," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2018, pp. 2042–2059.

[26] Y. Zhang, C. Xu, H. Li, K. Yang, N. Cheng, and X. Shen, "Protect: Efficient password-based threshold single-sign-on authentication for mobile users against perpetual leakage," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2020.2975792.

[27] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: Making backup cheap and easy," *ACM SIGOPS Operating Syst. Rev.*, vol. 36, no. SI, pp. 285–298, 2002.

[28] J. Xu, E. Chang, and J. Zhou, "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage," in *Proc. ACM ASIA Conf. Comput. Commun. Secur.*, 2013, pp. 195–206.

[29] M. Li, C. Qin, and P. P. Lee, "CDStore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal," in *Proc. USENIX Annu. Tech. Conf.*, 2015, pp. 111–124.

[30] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, 2011, pp. 491–500.

[31] Y. Zheng, X. Yuan, X. Wang, J. Jiang, C. Wang, and X. Gui, "Toward encrypted cloud media center with secure deduplication," *IEEE Trans. Multimedia*, vol. 19, no. 2, pp. 251–265, Feb. 2017.

[32] F. Armknecht, J. Bohli, G. O. Karame, and F. Youssef, "Transparent data deduplication in the cloud," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 886–900.

[33] J. Li, C. Qin, P. P. C. Lee, and J. Li, "Rekeying for encrypted deduplication storage," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2016, pp. 618–629.

[34] C. Qin, J. Li, and P. P. C. Lee, "The design and implementation of a rekeying-aware encrypted deduplication storage system," *ACM Trans. Storage*, vol. 13, no. 1, pp. 1–30, 2017.

[35] J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 874–885.

[36] M. Wen, K. Ota, H. Li, J. Lei, C. Gu, and Z. Su, "Secure data deduplication with reliable key management for dynamic updates in CPSS," *IEEE Trans. Comput. Social Syst.*, vol. 2, no. 4, pp. 137–147, Dec. 2015.

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

[38] 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.*, 2016, pp. 1192–1203.

[39] R. W. Lai, C. Egger, D. Schröder, and S. S. Chow, "Phoenix: Rebirth of a cryptographic password-hardening service," in *Proc. USENIX Secur. Symp.*, 2017, pp. 899–916.
[40] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[41] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Boca Raton, FL, USA: CRC Press, 2014.
[42] H. Li, R. Lu, L. Zhou, B. Yang, and X. Shen, "An efficient Merkle-tree-based authentication scheme for smart grid," *IEEE Syst. J.*, vol. 8, no. 2, pp. 655–663, Jun. 2014.
[43] C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Capkun, "Smartphones as practical and secure location verification tokens for payments," in *Proc. 21st Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 23–26.
[44] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, pp. 770–772, 1981.
[45] B. C. Neuman and T. Ts'o, "Kerberos: An authentication service for computer networks," *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 33–38, Sep. 1994.
[46] Y. Zhang, X. Lin, and C. Xu, "Blockchain-based secure data provenance for cloud storage," in *Proc. Int. Conf. Inf. Commun. Secur.*, 2018, pp. 3–19.
[47] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Pro. of Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2001, pp. 514–532.
[48] S. K. D. Maram *et al.*, "CHURP: Dynamic-committee proactive secret sharing," in *Proc. ACM Comput. Classification Syst.*, 2019, pp. 2369–2386.
[49] W. Yu, G. Xu, Z. Chen, and P. Moulema, "A cloud computing based architecture for cyber security situation awareness," in *Proc. IEEE Conf. Commun. Netw. Secur.*, 2013, pp. 488–492.
[50] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *Proc. Netw. Distrib. Syst. Symp.*, 2012, pp. 1–15.
[51] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Secur. Privacy*, vol. 8, no. 6, pp. 40–47, Nov./Dec. 2010.
[52] S. Li, C. Xu, and Y. Zhang, "CSED: Client-side encrypted deduplication scheme based on proofs of ownership for cloud storage," *J. Inf. Secur. Appl.*, vol. 46, pp. 250–258, 2019.
[53] J. Byun, H. Rhee, H. Park, and D. Lee, "Off-line keyword guessing attacks on recent keyword search schemes over encrypted data," in *Proc. Workshop Secure Data Manage.*, 2006, pp. 75–83.
[54] X. Zhang, C. Xu, H. Wang, Y. Zhang, and S. Wang, "FS-PEKS: Lattice-based forward secure public-key encryption with keyword search for cloud-assisted industrial Internet of Things," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2019.2914117.

Yuan Zhang (Member, IEEE) received the BSc and PhD degrees from the University of Electronic Science Technology of China (UESTC), in 2013 and 2019, respectively. From 2017 to 2019, he was a visiting PhD student with BBCR Lab, Department of Electronics and Communication Engineering, University of Waterloo, Canada. He is currently an associate professor with the School of Computer Science and Engineering, UESTC. His research interests include applied cryptography, data security, and blockchain technology.

Chunxiang Xu (Member, IEEE) received the BSc and MSc degrees in applied mathematics from Xidian University, Xi'an, China, in 1985 and 2004, respectively, and the PhD degree in cryptography from Xidian University in 2004. She is currently a professor with the School of Computer Science and Engineering, University of Electronic Science and Technology of China. Her research interests include information security, cloud computing security, and cryptography.

Nan Cheng (Member, IEEE) received the BE and MS degrees from Tongji University, Shanghai, China, in 2009 and 2012, respectively, and the PhD degree from the University of Waterloo, Waterloo, ON, Canada, in 2016. He is currently a professor with the School of Telecommunication, Xidian University. His research interests include on big data in vehicular networks and self-driving system.

Xuemin Shen (Fellow, IEEE) received the PhD degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990. He is currently a university professor with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research interests include network resource management, wireless network security, Internet of Things, 5G and beyond, and vehicular *ad hoc*, and sensor networks. He is currently a registered professional engineer at Ontario, Canada, and a distinguished lecturer of the IEEE Vehicular Technology Society and Communications Society. He was the recipient of the R.A. Fessenden Award in 2019 from the IEEE, Canada, the Award of Merit from the Federation of Chinese Canadian Professionals (Ontario) presented in 2019, the James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, the Education Award in 2017 from the IEEE Communications Society, the Joseph LoCicero Award in 2015, and the Technical Recognition Award from AHSN Technical Committee (2013) and Wireless Communications Technical Committee (2019). He was also the recipient of the Excellent Graduate Supervision Award in 2006 from the University of Waterloo, and the Premier's Research Excellence Award (PREA) in 2003 from the Province of Ontario, Canada. He was the technical program committee chair/co-chair for the IEEE Globecom'16, the IEEE Infocom'14, the IEEE VTC'10 Fall, the IEEE Globecom'07, the symposia chair for the IEEE ICC'10, and the chair for the IEEE Communications Society Technical Committee on Wireless Communications. He is the elected IEEE Communications Society vice president for Technical and Educational Activities, the vice president for Publications, a member-at-large on the Board of Governors, the chair of the Distinguished Lecturer Selection Committee, and a member of the IEEE Fellow Selection Committee. He was/is the editor-in-chief of the *IEEE Internet of Things Journal*, the *IEEE Network*, *IET Communications*, and the *Peer-to-Peer Networking and Applications*. He is a fellow of Engineering Institute of Canada, fellow of Canadian Academy of Engineering, fellow of Royal Society of Canada, and a foreign member of the Chinese Academy of Engineering.