This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2869333, IEEE Transactions on Cloud Computing

KeyD: Secure Key-Deduplication with Identity-Based Broadcast Encryption

Ling Liu, Yuqing Zhang and Xuejun Li

Abstract—Deduplication, which can save storage cost by enabling us to store only one copy of identical data, becomes unprecedentedly significant with the dramatic increase in data stored in the cloud. For the purpose of ensuring data confidentiality, they are usually encrypted before outsourced. Traditional encryption will inevitably result in multiple different ciphertexts produced from the same plaintext by different users' secret keys, which hinders data deduplication. Convergent encryption makes deduplication possible since it naturally encrypts the same plaintexts into the same ciphertexts. One attendant problem is how to reliably and effectively manage a huge number of convergent keys. Several deduplication schemes have been proposed to deal with the convergent key management problem. However, they either need to introduce key management servers or require interaction between data owners. In this paper, we design a novel client-side deduplication protocol named KeyD without such an independent key management server by utilizing the identity-based broadcast encryption (IBBE) technique. Users only interact with the cloud service provider (CSP) during the process of data upload and download. Security analysis demonstrates that KeyD ensures data confidentiality and convergent key security, and well protects the ownership privacy simultaneously. A thorough and detailed performance comparison shows that our scheme makes a better tradeoff among the storage cost, communication and computation overhead.

Index Terms—Data Deduplication, Convergent Encryption, Convergent Key Management, Identity-Based Broadcast Encryption.

1 Introduction

THE stored data is growing intensely with the advent of the era of Big Data. We need to constantly increase the storage devices if we continue using the traditional storage way. Alternatively, more and more users are prone to outsource their storage to cloud such as Amazon Web Services (AWS) [1] for economic savings. The ever-increasing data and users, coupled with multiple backup and other factors, result in more and more duplication of files or blocks in the cloud. In order to improve the storage efficiency in the pay-as-you-go model [2], deduplication operation is adopted for eliminating duplicate copies of redundant data on the cloud-side. Consider an example that m users outsource the same data copies[1] of n TB to the CSP. With data deduplication, only one copy is actually stored in the cloud, and the subsequent instances are referenced back to the saved copy for reducing storage roughly from mn to n TB. However, in order to protect the safety of the outsourced data, they are usually encrypted by their owners before outsourced to the CSP. Then it comes the problem, how can the CSP perform deduplication when these same data copies are encrypted into different ciphertexts by different users ?

Convergent encryption (CE) [3], which encrypts a data copy with a convergent key derived by computing the cryptographic hash value of the content of the data copy itself and thus can produce identical ciphertext from identical plaintext, brings the hope to realize deduplication while ensuring data confidentiality. This property of convergent encryption allows the CSP to perform deduplication on encrypted data. In particular, users encrypt their data copies using corresponding convergent keys and outsource encrypted data to the CSP. They just need to keep convergent keys locally so that they can later restore the data. However, the number of convergent keys increases linearly with the number of data copies since a data copy corresponds to a convergent key. As we all know, in practical file storage systems such as Google File System GFS [4] and Hadoop Distribute File System HDFS [5], data files are usually divided into fine-grained blocks to facilitate deduplication management, which makes the convergent key storage even more serious. Suppose the n TB data in the above example is divided into blocks of size 4 KB each, and that each convergent key is the hash value of SHA-256. Then each owner has to store a total size of $8n$ GB[2] of convergent keys. Such a storage overhead is still a great burden for the user.

A naive solution to this problem is described in [6]. Each data owner encrypts all his data copies using the corresponding convergent keys and further encrypts these convergent keys using his master key. Both encrypted data copies and convergent keys are stored in the cloud, and users keep their master keys and the metadata about the outsourced data locally. Although the encrypted data can be deduplicated by the cloud, the storage of encrypted convergent keys will increase linearly with the number of users.

- *L. Liu is with The School of Cyber Engineering, Xidian University, Xi'an, China, 710071.*
  E-mail: liul@nipc.org.cn
- *Y. Zhang is with The National Computer Network Intrusion Protection Center, Graduate University of Chinese Academy of Sciences, Beijing, China, 100049, and also with The School of Cyber Engineering, Xidian University, Xi'an, China, 710071.*
  E-mail: zhangyq@ucas.ac.cn
- *X. Li is with The School of Cyber Engineering, Xidian University, Xi'an, China, 710071.*
  E-mail: aluckydd@mail.xidian.edu.cn

1. A data copy can represent a whole file or a more fine-grained data block, depending on different granularities.

2. This storage space is calculated by the following equation: n TB / 4KB (per block) * 256 bits (per convergent key) = $8n$ GB.

2168-7161 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2869333, IEEE Transactions on Cloud Computing

Performing deduplication on these encrypted convergent keys is just as important as performing deduplication on encrypted data copies.

Several attempts have been made to reduce the convergent key management complexity from $O(m)$ to $O(1)$, namely the cost of storing convergent keys is independent of the number of owners. Li et al. [6] employs the Ramp secret sharing scheme (RSSS) [7] [8] to construct secret shares for the convergent keys and distribute them across multiple independent Key Management Cloud Service Providers (KM-CSPs). To recover data copies, a user must access a minimum number of key servers through authentication and obtain the secret shares to reconstruct the convergent keys. The premise of the scheme security is that the number of colluded KM-CSPs is not more than a predefined threshold, such that a convergent key cannot be guessed by the colluded KM-CSPs. Wen et al. [9] propose a session-key-based convergent key management scheme and an improved version. A trustable gateway is needed to check the duplication and buffer the new data blocks for periodically uploading them to the cloud storage server (CSS). As far as we know, there is no deduplication scheme that can effectively manage convergent keys, without the aid of additional independent key management servers or other trusted parties like Gateway. For the purpose of performing secure deduplication on encrypted convergent keys effectively and practically, we propose a new deduplication scheme KeyD without such a strong assumption. The three main contributions of this paper are summarized as follows.

- We propose a novel client-side deduplication scheme. Specifically, we make a combination of convergent encryption (CE) and ID-based broadcast encryption (IBBE) to achieve secure and efficient convergent key management, without introducing any other independent key management servers or trusted third parties.
- Security analysis demonstrates that our scheme ensures the confidentiality of data files and the security of convergent keys.
- A comprehensive performance comparison between KeyD and several present works is given, showing that our scheme makes a better tradeoff among the storage cost, communication overhead and computation overhead.

2 Related Work

Data deduplication in cloud storage [10] [11] is an effective technique to improve storage space utilization by eliminating duplicate copies. According to the location where it is performed, deduplication strategies can be divided into server-side deduplication and client-side deduplication. In server-side schemes, the CSP performs deduplication after it receives all data, thus part of the network bandwidth will be wasted to upload duplicate copies. In client-side schemes, each data copy is uploaded just once. When users upload duplicate data, they will be informed by the CSP about the duplication and do not need to upload the data any longer. Another categorization criteria is the data granularity and deduplication strategies can be divided into file-level deduplication and block-level deduplication. File-level deduplication eliminates duplicate data files, while block-level deduplication eliminates duplicate blocks in different files. Block-level deduplication systems are undoubtedly more flexible and efficient to handle nonidentical files with identical blocks. Our work falls in the category of client-side deduplication and can work in both file-level and block-level, without the aid of key management servers or other trusted third parties.

A majority of existing deduplication schemes are based on Convergent Encryption (CE) [3], which was proposed by Douceur et al. in 2002 and later formalized as Message-Locked Encryption (MLE) in [12] [13]. Storer et al. are the first to use CE to implement deduplication [14]. They proposed two secure deduplication schemes in single-server storage and distributed storage systems, while an independent metadata server is needed to store metadata in the later. Anderson et al. [15] designed a deduplication scheme for fast and secure laptop backups, which successfully reduced the number of files to be scanned and hence decreased backup times. A local server is needed to implement multi-user authentication to shared data. Both [14] and [15] works on file-level. More fine-grained block-level deduplication was achieved in [16], with a metadata manager introduced to implement key management. [17] reduces the linear pairing comparison times of the scheme in [13] to nearly logarithmic times. All the above schemes based on convergent encryption are server-side deduplication.

Several client-side deduplication schemes have been made based on the Proof of Ownership (PoW) [18] technique. PoW enables users to prove their ownership of data when duplication happens and then users do not need to upload duplicate data, thus greatly saving the upload bandwidth. The state-of-the-art PoW was further extended by Pietro et al. [19], but neither [18] nor [19] addressed data privacy. The private data deduplication [20] can be viewed as a complement of [18], requiring a trusted coordinator be allowed to help users perform decryption. [21] allows a bounded amount one-time leakage of a target file compared with [18] that allows a bounded amount multi-time leakage, without block-level deduplication in consideration. Stanek et al. [22] achieved a better tradeoff between the efficiency and security by providing different security level for popular and unpopular data, but two trusted entities are needed to identify duplication and manage data information.

There is another line of works for secure data deduplication based on a trusted third party without MLE/CE. [25], [26] and [9] achieve data confidentiality by transmitting predictable messages into unpredictable ones. [25] introduces a key server to derive keys, instead of setting keys to be hashes of messages. Miao et al. [26] argue that the DupLESS does not work when the key server is corrupted by the cloud server and they propose a new multi-server-aided deduplication scheme based on the threshold blind signature. However, both of the two schemes work on server-side. [9] realizes client-side deduplication to support dynamic updates by introducing a trusted third party.

Schemes making a combination of CE and PoW [6] [23]are able to ensure data privacy and achieve client-side deduplication simultaneously, while facing the problem of how to manage a large number of convergent keys according to Li et al. [6]. Storing convergent keys locally is quite a

2168-7161 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2869333, IEEE Transactions on Cloud Computing

heavy burden for users, but encrypting them with different master keys and outsourcing them will lead to the duplication of encrypted convergent keys on the server side. To realize convergent key deduplication securely and reliably, Li et al. [6] introduce multiple key management servers. They compute secret shares for each convergent key and distribute these shares to key management servers, instead of encrypting the convergent key. [9] also aims at managing convergent keys and supporting dynamic updates. A trustable gateway is needed to check the duplication and buffer the new data blocks for periodically uploading them to the cloud storage server. To our best knowledge, [24] is currently the only solution that supports client-side deduplication without additional independent servers. Unfortunately, either [9] or [24] involves complex interactions between clients for key sharing, which violates the ownership privacy during the data deduplication. Therefore, designing a secure and effective client-side deduplication scheme without other independent servers or trusted third parties is still an open problem.

3 PRELIMINARY

We utilize convergent encryption (CE) technique to encrypt files and blocks, proof of ownership (PoW) for a client to prove that he does have the file, and identity-based broadcast encryption (IBBE) combined with a symmetric encryption (SE) scheme to encrypt convergent keys. As is well-known, a SE scheme is a triple of algorithms $(KeyGen(1^\lambda) \rightarrow k, Encrypt(k, M) \rightarrow C, Decrypt(k, C) \rightarrow M)$, where λ is a security parameter and k, M, C are the key, plaintext and ciphertext respectively. Note that our plaintext data are unpredictable for adversaries who do not own them, and they cannot be feasibly enumerated by adversaries. Therefore, our data encryption scheme supports an arbitrary form of Message-Locked Encryption (MLE) [13] but not limited to Convergent Encryption (CE), despite of the determinacy of MLE and CE.

3.1 Convergent Encryption

Convergent encryption, introduced by Douceur et.al. [3], has been widely used in the deduplication of data stored in the cloud. It is a cryptosystem that produces identical ciphertext files from identical plaintext files, irrespective of their encryption keys. To encrypt a data copy (a file or a block) using convergent encryption, a user first computes a cryptographically strong hash value from the data copy, and then using this hash value as the convergent key to encrypt the data copy. The user could also derive a tag for the data copy, which will be used to detect duplication. In this way, the same data copy will be encrypted by the same key, resulting in the same ciphertext and tag. Then the ciphertext and the tag are given to the server and the user retains the convergent key. The server can now perform deduplication on the ciphertext, checking whether or not it is already stored [12]. Note that both the convergent key and the tag are independently derived, and the tag cannot be used to deduce the convergent key and compromise data confidentiality [6]. A convergent encryption scheme can be formally defined as a tetrad of the following four algorithms (KeyGen, Encrypt, Decrypt, TagGen) :

- $KeyGen(M) \rightarrow K$ is the key generation algorithm that maps a data copy M to a convergent key K;
- $Encrypt(K, M) \rightarrow C$ is the symmetric encryption algorithm that takes both the convergent key K and the data copy M as inputs and then outputs a ciphertext C;
- $Decrypt(K, C) \rightarrow M$ is the decryption algorithm that takes both the ciphertext C and the convergent key K as inputs and then outputs the original data copy M;
- $TagGen(M) \rightarrow T(M)$ is the tag generation algorithm that maps the original data copy M and outputs a tag $T(M)$.

3.2 Proof of Ownership

Deduplication requires that if two or more users own the same file, only a single copy should be stored in the cloud. When users upload a file that has been existing, he will prove his ownership of the file to the user and obtains the information associated with the storage, such as the pointer of the copy. The user directly sending a hash value of the data copy to the server for verification seems like a possible solution. However, as the data is encrypted, the cloud server cannot compute the hash value of the data copy and needs to store many hash values along with data copies, especially when the amount of data is huge. Halevi et.al. [18] first proposed the notion of proof of ownership (PoW), which enables the client to prove to the server that he has a copy of the file, without actually sending the file. It's essentially an interactive protocol performed between a prover (user) and a verifier (server). The verifier first derives a short value $\phi(M)$ from the data copy M. In order to prove the ownership of a data copy M, the prover is supposed to send ϕ' to the verifier. If $\phi' = \phi(M)$, the verifier will believe that the prover indeed has M. In this way, a bandwidth efficient prove and verification is achieved between user and server. For simplicity, we also use the notations of POW_F and POW_B as in [6] to denote PoW for a file F and block B, respectively.

3.3 Identity-based Broadcast Encryption

We adopt the Identity-Based Broadcast Encryption (IBBE) [27] to encrypt convergent keys before sending them to the Cloud Service Provider (CSP). A necessary authority involved in an IBBE is the Private Key Generator $\mathcal{PKG}$. Using its master secret key MSK, the $\mathcal{PKG}$ can generate a decryption key sk_{ID_i} for each new member with identity ID_i to decrypt messages. An attractive feature of the IBBE scheme is that the broadcaster does not hold any private information. Messages can be encrypted with the help of a public key PK and the set $\mathcal{S}$ of identities of the receivers. Then all the identities in $\mathcal{S}$ are able to decrypt the messages.

Given security parameter λ and maximal size m of the target set, an identity-based broadcast encryption scheme $IBBE$ can be formally described as a tuple of algorithms (Setup, Extract, Encrypt, Decrypt):

- $Setup(\lambda, M)$. Takes as input the security parameter λ and m the maximal size of the set of receivers for one encryption, and outputs a master secret key MSK and a public key PK.

2168-7161 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2869333, IEEE Transactions on Cloud Computing

- $Extract(MSK, ID_i)$. Takes as input the master secret key MSK and a user identity ID_i. $Extract$ generates a user private key sk_{ID_i}.
- $Encrypt(\mathcal{S}, PK)$. Takes as input the public key PK and a set of included identities $\mathcal{S} = \{ID_1, \ldots, ID_s\}$ with $s \leq m$, and outputs a pair (Hdr, K). When a message $M \in \{0,1\}$ is to be broadcast to users in $\mathcal{S}$, the broadcaster generates $(Hdr, K) \leftarrow Encrypt(\mathcal{S}, PK)$, computes the encryption C_M of M under the symmetric key $K \in \mathcal{K}$ and broadcasts $(Hdr, \mathcal{S}, C_M)$. We will refer to Hdr as the header (which actually encapsulates the identity-based broadcast encryption key K) of broadcast ciphertext, K as the message encryption key and C_M as the broadcast body.
- $Decrypt(\mathcal{S}, ID, sk_{ID}, Hdr, PK)$. Takes as input a subset $\mathcal{S} = \{ID_1, \ldots, ID_s\}$ (with $s \leq m$), an identity ID and the corresponding private key sk_{ID}, a header Hdr, and the public key PK. If $ID \in \mathcal{S}$, the algorithm outputs the message encryption key K which is then used to decrypt the broadcast body C_M and recover M.

4 Problem Formulation

4.1 System Model

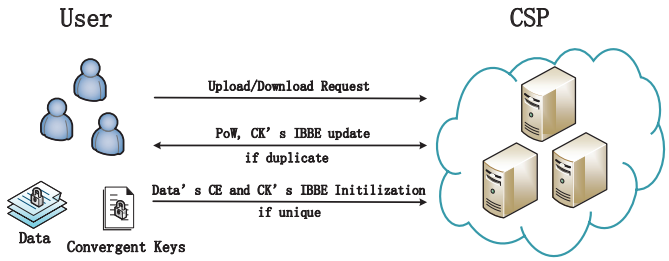


Fig. 1. The system model of KeyD

There are two types of entities in our data storage outsourcing model, including the user and the cloud service provider (CSP), as shown in Fig.1. Both file-level and block-level deduplication are supported in our system. Specifically, file-level deduplication can be reduced to block-level deduplication and we will put emphasis on the latter.

- The *user* owns a large amount of data, and he is usually limited in his storage space and computing capability. So he wants to outsource the data to the cloud and access them later. For the purpose of saving network bandwidth in the data upload phase, the user only uploads the data that has not been stored in the cloud. If duplication happens, he runs the PoW protocol with the CSP to prove his ownership, and updates materials related to the convergent key (CK)'s identity-based broadcast encryption (IBBE) with the help of CSP. Otherwise, he CE-encrypts the data and IBBE-encrypts the convergent key respectively and then uploads them, namely "Data's CE and CK's IBBE Initialization" in Fig.1. In our setting, the user only needs to protect his secret key and maintain his ID so that he can later decrypt the convergent keys to further retrieve the outsourced blocks.
- The *CSP* is an entity that provides cloud storage services, including data, convergent keys and other relative materials. It maintains a list of tags of data files and blocks for checking deduplication, and helps the user to do some expensive computations. When the CSP receives a file upload request, it first checks if the file duplicates. If yes, it informs the user to prove his ownership of the file, returns the file pointer to the user, and updates the materials related to CKs' IBBE version of all the blocks making up this file.

4.2 Security Model

We mainly consider two types of security threats from the CSP and outside attackers respectively. The CSP in our system is assumed to be "honest-but-curious", namely that it will honestly execute the assigned tasks in the deduplication protocol, but would like to learn as much secret information as possible. For instance, it may attempt to extract useful information of data or convergent keys. An outside attacker refers to a malicious user that intends to obtain some knowledge about data files not owned by him through interactions with the CSP. Therefore, all the data files and convergent keys are sensitive and need to be fully protected against outside attackers and the CSP.

4.3 Design Goal

In this paper, we construct a secure deduplication scheme to manage convergent keys, which bears the following function and security goals.

- *No Additional Independent Servers or Trusted Third Parties*. Our primary goal is to achieve secure and reliable convergent key deduplication, without additional independent servers or trusted third parties. The scheme should support client-side deduplication by interactions between only the users and the cloud server.
- *Confidentiality of Data*. Our scheme should guarantee the confidentiality of data, implying that the user cannot get the ownership of the data from the CSP by running the PoW protocol if he does not have the data, and the user who cannot prove his ownership should be unable to decrypt the ciphertext stored in the cloud storage. The data should also be protected well against the curious server.
- *Semantic Security of Convergent Keys*. The security of convergent keys should fulfill that even if an adversary obtains some encrypted convergent keys, he is unable to recover the convergent keys for files that he does not own, and he cannot distinguish convergent keys according to their corresponding encrypted versions. The convergent keys should be semantically secure against the honest-but-curious cloud server as well.
- *Ownership Privacy*. The ownership of a data copy by a user should be private to both malicious users and

2168-7161 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2869333, IEEE Transactions on Cloud Computing

other users who share the identical copy. It means users who do not own the data cannot know who has/have it in any way, and data owners cannot know who share(s) the data with them when duplication happens, even through interactions with the CSP during ownership update process.

5 Construction of KeyD

We consider two processes of file upload and file download in our KeyD. In order to save upload bandwidth, users only upload new data that do not exist in cloud storage. Upon receiving a file upload request $T(F)$ with a user identity ID, the CSP first checks if it has stored F. If not, then it performs block-level duplication check and only unique blocks need to be uploaded. Before outsourcing the storage of data to the CSP, the user first encrypts data using convergent keys, which will guarantee that the same data copy will result in the same ciphertext. In terms of the large number of convergent keys, we utilize the Identity-Based Broadcast Encryption (IBBE) key instead of the user's master secret key to encrypt them. Let $\mathcal{S}$ be a set of identities of the users sharing the same convergent key (i.e the same data copy). Specifically, the encrypt algorithm of IBBE will produce an IBBE key k_B and a broadcast header Hdr to encapsulate k_B. The convergent key will be encrypted using k_B and sent to the CSP. All users in $\mathcal{S}$ are capable of recovering this convergent key using their own private keys. For those duplicate blocks, the user runs the Proof of Ownership (Pow) protocol with the CSP to prove his ownership. Then the ownership of these blocks (i.e. $\mathcal{S}$ and Hdr) will be updated through interactions between the user and the CSP. If F is duplicate, then all its blocks will be duplicate as well, and the ownership of each block needs to be updated. File download is a single-round interaction between the user and the CSP. Upon receiving a file download request from a user, the CSP checks if the user is eligible. If not, the download process will be aborted. Else, the CSP returns encrypted blocks and encrypted convergent keys along with IBBE headers and receiver sets of all blocks constructing this file. And the user can finally restore the entire file step by step. The construction details of KeyD are described as follows.

5.1 System Setup

In this phase, various necessary system parameters are initialized.

The three encryption schemes will be used are initialized as follows:(1) A symmetric encryption scheme with the primitive functions $(KeyGen_{SE}, Enc_{SE}, Dec_{SE})$; (2) A convergent encryption scheme consisted of four algorithms $(KeyGen_{CE}, Enc_{CE}, Dec_{CE}, TagGen_{CE})$; (3) An identity-based broadcast encryption (IBBE) scheme of four algorithms $(Setup_{IBBE}, Extract_{IBBE}, Enc_{IBBE}, Dec_{IBBE})$. To facilitate the scheme description later, it is necessary to emphasize that the $Setup_{IBBE}$ algorithm will produce the following materials: a bilinear map group system $\mathcal{B} = (p, G_1, G_2, G_T, e(\cdot,\cdot))$ and a hash function H as system public parameters, (g, γ) as the master key MSK, and $(\omega, \nu, h, h^{\gamma}, \ldots, h^{\gamma^m})$ as the public key PK. Here, $|p|$ is the security parameter λ, g and h are the generators of groups G_1 and G_2, γ is a secret value, $\omega = g^{\gamma}$, and $\nu = e(g, h)$. And $Extract_{IBBE}(MSK, ID_i)$ produces a private key

$$sk_{ID_i} = g^{\frac{1}{\gamma + H(ID_i)}} \tag{1}$$

for the user with identity ID_i. In addition, a Proof of Ownership (PoW) algorithm for files and blocks, denoted by PoW_F and PoW_B respectively, will be initialized as well.

The CSP needs to initialize its storage system for data copy and other relevant information. More specifically, the CSP maintains a list for every file F, with each item being $\{T(F), \{T(B_i)\}, \{c_i\}, \{d_i\}, \{ck_i\}, \{Hdr_i\}, \{\mathcal{S}_i\}\}$. Here, $T(F)$ is the tag of F, $\{T(B_i)\}$ is the set of tags of all blocks of F, $\{c_i\}$ is the set of encrypted blocks, $\{d_i\}$ is the set of encrypted random numbers that are randomly selected from Z_q^* by the first uploader of each block B_i. $\{ck_i\}$ is the set of encrypted (by identity-based broadcast encryption scheme) convergent keys, $\{Hdr_i\}$ is the set of the headers used to encapsulate the IBBE keys, and each $\mathcal{S}_i$ is the set of receivers of block B_i (i.e. the set of users owning B_i). The list is initialized to be $\perp$.

5.2 File Upload

Initially, the user with identity ID precomputes $H(ID)$. When he wants to upload a file F, he is supposed to interact with the Cloud Service Provider (CSP) following the steps below.

Step-1: The user computes a tag $T(F) = TagGen_{CE}(F)$ for F and sends it along with ID to the CSP.

Step-2: Upon receiving the $T(F)$, the CSP stores ID and checks whether it has stored the same tag. If so, it responds to the user "file duplicates", or "no file duplicates" otherwise.

Step-3: If the user receives the response "no file duplicates", it proceeds to Step-7 to perform block-level deduplication. Otherwise, the user runs the PoW protocol PoW_F on $T(F)$ with the CSP to prove his ownership of file F.

Step-4: If PoW_F fails, the CSP aborts the upload operation. Otherwise, the CSP believes that the user indeed has the file F. In this case, the user does not need to upload F but simply adds his identity into the receiver set of each block constructing F. It is known that the server stores $\{T(F), T(B_i), c_i, d_i, ck_i, Hdr_i, \mathcal{S}_i\}$ for all blocks of F, where Hdr_i and $\mathcal{S}_i$ contains all the identities of users owning block B_i. According to the description of IBBE scheme,

$$Hdr_i = (C_{i1}, C_{i2}) = (\omega^{-k_i}, h^{k_i \prod_{j=1}^{s_i} (\gamma + H(ID_j))}) \tag{2}$$

. For the user, adding his identity ID into $\mathcal{S}_i$ is easy, while updating the header Hdr_i to include ID is too expensive a computation for him. Therefore, the major computation is undertaken by the CSP in our protocol. Since k_i is a secret value randomly chosen by the first uploader of block B_i, the CSP can not directly compute a new

$$C_{i2}{}' = {C_{i2}}^{\gamma + H(ID)} = h^{k_i \prod_{j=1}^{s_i} (\gamma + H(ID_j))(\gamma + H(ID))} \tag{3}$$

for the user. Instead, it computes a

$$C_i = h^{\prod_{j=1}^{s_i} (\gamma + H(ID_j))(\gamma + H(ID))} \tag{4}$$

2168-7161 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2869333, IEEE Transactions on Cloud Computing

for each block[3] and returns $\{d_i\}, \{Hdr_i\}, \{C_i\}$ along with the file pointer of F to the user.

Step-5: Upon receiving $\{d_i\}, \{Hdr_i = (C_{i1}, C_{i2})\}, \{C_i\}$, the user needs to check whether all C_is are correctly computed by the CSP. He first computes the convergent key $k_{c_i} = KeyGen_{CE}(B_i)$ for each B_i and uses k_{c_i} to decrypt d_i to obtain k_i. Then he computes ${C_{i2}}' = {C_i}^{k_i}$ and verifies if

$$e({C_{i2}}', h) = e(C_{i2}, h^{\gamma + H(ID)}) \tag{5}$$

for all blocks. If the verification passes, the user stores the file pointer of F and sets ${Hdr_i}' = (C_{i1}, {C_{i2}}')$. Finally, he sends all new headers $\{Hdr_i'\}$ to the CSP.

Step-6: The CSP replaces all Hdr_i and $\mathcal{S}_i$ corresponding to $T(B_i)$ with ${Hdr_i}'$ and $\mathcal{S}_i' = \mathcal{S}_i + \{ID\}$, respectively. At this point, the user successfully adds his identity ID into the receiver set of each block of the file F, which will guarantee that he can decrypt the ciphertext of B_i stored by the CSP later. He just keeps the file pointer of F and does not need to upload the file.

Step-7: This step means that the file to be uploaded does not duplicate and it is necessary to further detect block duplication. The user divides F into a set of blocks $\{B_i\}$ and computes $T(B_i) = TagGen_{CE}(B_i)$ for each block. Then he sends all the tags $\{T(B_i)\}$ to the server.

Step-8: Upon receiving $\{T(B_i)\}$, the CSP checks whether it has stored the same blocks and creates a vector D_B to indicate whether a block duplicates or not. If the same tag exists, it sets $D_B[i] = 1$, else it sets $D_B[i] = 0$ and stores $T(B_i)$. Finally, it returns the vector D_B to the user.

Step-9: The user checks all items of D_B. IF $D_B[i] = 0$, the system proceeds to Step-12. Else if $D_B[i] = 1$, the user runs PoW_B on $T(B_i)$ with the CSP to prove the ownership of B_i.

Step-10: Once all PoW_Bs are finished, the server computes C_i for these duplicating blocks as in Step-4. Then it returns {block pointer of B_i, d_i, Hdr_i, C_i} to the user.

Step-11: Upon receiving these information, the user stores the block pointers of $\{B_i\}$ and interacts with the CSP as from Step-5 to Step-6 to add himself into the receiver sets of these blocks. We just finish the block deduplication of the file to be uploaded.

Step-12: From this step, we deal with the blocks of F that have not been stored by the CSP, namely, the user needs to upload blocks corresponding to $D_B[i] = 0$. For each block, he first randomly chooses $k_i \in Z_q^*$ and computes the convergent key $k_{c_i} = KeyGen_{CE}(B_i)$ used to encrypt the data block B_i. Different from [6], which directly encrypts a data block using the convergent key, we encrypt the data block in the form $c_i = Enc_{CE}(k_{c_i}, B_i||k_i)$. This is to ensure that even though a user occasionally intercepts some blocks stored on the CSP, he can never know whether he owns any of these blocks. We additionally encrypt k_i using convergent key k_{c_i} in the form $d_i = Enc_{SE}(k_{c_i}, k_i)$ to make sure that when a user uploads the same data copy, he can decrypt d_i he receives from the CSP to obtain k_i, so that he can add himself into the receiver set. It is worth mentioning that a user can certainly recover k_i from c_i, but the communication overhead for the CSP to send c_i to a user is much larger than d_i. Since he is the first and only one to upload B_i so far, the user sets $\mathcal{S}_i = \{ID\}$ and calls the algorithm $Enc_{IBBE}(\mathcal{S}_i, PK)$ with k_i to compute the header[4]

$$Hdr_i = (C_{i1}, C_{i2}) = (\omega^{-k_i}, h^{k_i(\gamma + H(ID))}) \tag{6}$$

and the IBBE key $k_{B_i} = \nu^{k_i}$. He uses k_{B_i} to symmetrically encrypt the convergent key k_{c_i}, i.e. $ck_i = Enc_{SE}(k_{B_i}, k_{c_i})$. Finally, the user sends $\{c_i, d_i, ck_i, Hdr_i\}$ along with the file tag $T(F)$ to the CSP.

Step-13: Upon receiving these materials from the user, the CSP sets $\mathcal{S}_i = \{ID\}$ and finally stores $\{T(F), T(B_i), c_i, d_i, ck_i, Hdr_i, \mathcal{S}_i\}$ for each block B_i.

5.3 File Download

When a user wants to download a file F that he stored on the server side, he sends a download request for F to the server. Upon receiving the download request for a file F, the CSP first checks whether the user is eligible to download F. For example, the CSP can check if the user is in receiver sets of blocks making up F. If the user is not eligible, the CSP will return an "abort" signal to inform the user of the failure. Otherwise, the CSP returns $\{c_i, ck_i, Hdr_i, \mathcal{S}_i\}$ corresponding to all blocks $\{B_i\}$ of the file F.

Having received these materials from the CSP, the user performs the following operations in sequence.

(1) Recovers the identity-based broadcast encryption key k_{B_i} using these information with his own identity ID by calling the algorithm $Dec_{IBBE}(\mathcal{S}_i, ID, sk_{ID}, Hdr_i, PK)$.

(2) Calls the decryption algorithm of the SE scheme to compute convergent key $k_{c_i} = Dec_{SE}(k_{B_i}, ck_i)$.

(3) Uses k_{c_i} to decrypt c_i as $Dec_{CE}(k_{c_i}, c_i) = B_i||k_i$, thus the user obtains all the blocks of F.

(4) Finally, the user utilizes $\{B_i\}$ to obtain the original file F.

6 Correctness and Security

6.1 Correctness

We say that the secure deduplication scheme based on the identity-based broadcast encryption (IBBE) is correct only if (1) a data owner can successfully add himself into the receiver set of a block if it duplicates, without leaking the master key of the system and the IBBE key, and (2) the user in the receiver set of a block can successfully recover the block using his secret key. The correctness analysis of our KeyD is given below.

3. According to the IBBE scheme, γ is a secret value unknown to the CSP. But $h, h^{\gamma}, ..., h^{\gamma^m}$ are parts of PK, and $\mathcal{S}_i = \{ID_j | j = 1, ..., s_i\}$ is kept by the CSP. Note that m is the maximal size of the set of receivers and $s_i < m$. So the CSP can compute C_i in Equation 3 by $C_i = h^{\prod_{j=1}^{s_i}(\gamma + H(ID_j))(\gamma + H(ID))} = h^{(\gamma + H(ID_1))(\gamma + H(ID_2))\cdots(\gamma + H(ID_{s_i}))(\gamma + H(ID))} \xrightarrow{ID_{s_i+1} = ID} h^{\gamma^{s_i+1} + \gamma^{s_i} \sum_{j=1}^{s_i+1} H(ID_j) + \gamma^{s_i - 1} \sum_{j=t=1, j \neq t}^{s_i+1} H(ID_j)H(ID_t) + \cdots + \prod_{j=1}^{s_i+1} H(ID_j)} = h^{\gamma^{s_i+1}} \cdot h^{\gamma^{s_i} \sum_{j=1}^{s_i+1} H(ID_j)} \cdot h^{\gamma^{s_i - 1} \sum_{j=t=1, j \neq t}^{s_i+1} H(ID_j)H(ID_t)} \cdot \ldots \cdot h^{\prod_{j=1}^{s_i+1} H(ID_j)}$.

4. The second part $C_{i2} = h^{k_i(\gamma + H(ID))}$ of this header will become the form of $h^{k_i \prod_{j=1}^{s_i}(\gamma + H(ID_j))}$ in Equation 2 as more users upload B_i later, as described in Step-4.

2168-7161 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2869333, IEEE Transactions on Cloud Computing

(1) The process of adding an identity into the receiver set of a block is done through the interaction between the CSP and user, as is shown from Step-4 to Step-6 in Section 5.2. Taking a duplicate block B_i as an example, the uploader hopes that his identity ID will be added into both the set $\mathcal{S}_i$ and the header $Hdr_i = (C_{i1}, C_{i2})$. C_{i1} is independent of ID and $\mathcal{S}_i$ is easy to update, while C_{i2} requires a lot of computation to update. As can be seen from Section 5.1, $(h, h^{\gamma}, \ldots, h^{\gamma^m})$ are public, and $\mathcal{S}_i$ containing identities of all the other owners is stored in the cloud. Therefore, the CSP can compute a $C_i = h^{\prod_{j=1}^{s_i}(\gamma+H(ID_j))(\gamma+H(ID))}$ for the user, without knowing the master key γ. The user then recovers the random number k_i chosen by the first uploader, using B_i owned by him and d_i he received from the CSP. He computes $C'_{i2} = {C_i}^{k_i}$ and performs the verification $e({C_{i2}}', h) \stackrel{?}{=} e(C_{i2}, h^{\gamma+H(ID)})$ based on the bilinear property of bilinear map [28]. The verification could pass only if C_i is correctly computed by the server. Thus the new header Hdr'_i including ID is also correctly computed since C'_{i2} happens to be the new C_{i2}.

(2) When a user downloads $\{c_i, ck_i, Hdr_i, \mathcal{S}_i\}$ from the server, he hopes to recover the corresponding block B_i. The first material that he has to obtain is the IBBE key k_{B_i}. Then he can use k_{B_i} to symmetrically decrypt ck_i to obtain convergent key k_{c_i}, thus can CE-decrypt c_i to obtain B_i. Therefore, whether the user can correctly recover B_i depends mainly on whether he can correctly compute k_{B_i}. According to Section 3.1 in [27], the user with identity ID and the corresponding private key sk_{ID} could call the algorithm $Dec_{IBBE}(\mathcal{S}_i, ID, sk_{ID}, Hdr_i, PK)$ to correctly compute

$$
\begin{aligned}
&[e(C_{i1}, h^{p_{\mathcal{S}(\gamma)}}) \cdot e(sk_{ID}, C_{i2})]^{\frac{1}{\prod_{j=1}^{s'_i} H(ID_j)}} \\
&= [e(\omega^{-k_i}, h^{\frac{1}{\gamma}(\prod_{j=1}^{s'_i}(\gamma+H(ID_j)) - \prod_{j=1}^{s'_i} H(ID_j))}) \cdot \\
&e(g^{\frac{1}{\gamma+H(ID)}}, h^{k_i \prod_{j=1}^{s_i}(\gamma+H(ID_j))})]^{\frac{1}{\prod_{j=1}^{s'_i} H(ID_j)}} \\
&= [e(g^{-k_i\gamma}, h^{\frac{1}{\gamma}(\prod_{j=1}^{s'_i}(\gamma+H(ID_j)) - \prod_{j=1}^{s'_i} H(ID_j))}) \cdot \\
&e(g, h)^{\frac{1}{\gamma+H(ID)} \cdot k_i \prod_{j=1}^{s_i}(\gamma+H(ID_j))}]^{\frac{1}{\prod_{j=1}^{s'_i} H(ID_j)}} \\
&= [e(g, h)^{-k_i(\prod_{j=1}^{s'_i}(\gamma+H(ID_j)) - \prod_{j=1}^{s'_i} H(ID_j)) + k_i \prod_{j=1}^{s'_i}(\gamma+H(ID_j))} \\
&]^{\frac{1}{\prod_{j=1}^{s'_i} H(ID_j)}} \\
&= [e(g, h)^{k_i \prod_{j=1}^{s'_i} H(ID_j)}]^{\frac{1}{\prod_{j=1}^{s'_i} H(ID_j)}} \\
&= e(g, h)^{k_i} = v^{k_i} = k_{B_i}
\end{aligned}
\tag{7}
$$

with $p_{\mathcal{S}(\gamma)} = \frac{1}{\gamma}(\prod_{j=1}^{s'_i}(\gamma + H(ID_j)) - \prod_{j=1}^{s'_i} H(ID_j))$, where $\mathcal{S}'_i$ denotes the set that removing ID from $\mathcal{S}_i$ and $s'_i = |\mathcal{S}'_i|$. It should be noted that computing $h^{p_{\mathcal{S}(\gamma)}}$, namely $h^{\frac{1}{\gamma}(\prod_{j=1}^{s'_i}(\gamma+H(ID_j)) - \prod_{j=1}^{s'_i} H(ID_j))}$ will overburden the user significantly, so this computation can also be finished by the CSP similarly as it computes C_i in Equation 3. Therefore, it can be concluded that in our secure deduplication scheme KeyD, as long as the user's identity ID is included in $\mathcal{S}_i$ and Hdr_i, he can correctly recover B_i.

6.2 Security Analysis

Our scheme is designed to solve the data privacy and key management problems in deduplication in secure storage outsource model. As is described in Section 4.2, the security goal of our scheme is to realize data confidentiality, semantic security of convergent keys and ownership privacy. We will demonstrate that KeyD fulfills the security requirements in the security analysis below.

6.2.1 Data Confidentiality

In our system, a user data block B is encrypted in the form of $Enc_{CE}(k_c, B||k)$ before being outsourced to the cloud server, where k_c is the convergent key and k is randomly chosen by the first user to upload B. Therefore, the confidentiality of data can be achieved if convergent encryption is secure and k_c is well protected against the adversary. The security of the convergent encryption adopted in our scheme has been proved in [3], which guarantees that no more except a controlled amount of information will be deliberately leaked for determining whether or not the plaintexts of two encrypted data files are identical. This means not only the confidentiality of data, but also the security of k. Otherwise, an adversary can easily compute IBBE key $k_B = v^k$ and then symmetrically decrypts encrypted convergent key to obtain k_c. In our KeyD, k_c is encrypted utilizing the identity-based broadcast encryption (IBBE) technique, and its security will be demonstrated in Section 6.2.2. The identities of users who do not own B are not in the receiver set, meaning that they cannot use correct Hdr and $\mathcal{S}$ to compute k_B and recover k_c.

6.2.2 Semantic Security of Convergent Keys

According to the scheme construction in Section 5, convergent keys in our scheme KeyD are encrypted utilizing the Identity-Based Broadcast Encryption (IBBE) keys. The semantic security of convergent keys requires that it is computationally hard for an adversary to distinguish convergent keys according to their corresponding encrypted versions. It also implies that he cannot distinguish the convergent key is encrypted using an IBBE key or a random value in $\mathcal{K}$ by its encryption, where $\mathcal{K}$ is the key space of an IBBE scheme, namely G_T. Formally, The semantic security of convergent keys is defined using the following game between a challenger $\mathcal{C}$ and an attacker $\mathcal{A}$, and the security parameter λ in IBBE scheme is given to both players:

1. $\mathcal{C}$ runs algorithm $Setup_{IBBE}(\lambda, m)$ to generate system master key MSK and public key PK, where m is the maximum number of owners of a data block.

2. $\mathcal{C}$ randomly chooses a data block B and calls the algorithm $KeyGen_{CE}(B)$ to generate the corresponding convergent key k_c. And he also randomly constructs a receiver set $\mathcal{S}$, the number of identities in which is supposed

2168-7161 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2869333, IEEE Transactions on Cloud Computing

to be s. Then $\mathcal{C}$ calls the algorithm $Enc_{IBBE}(\mathcal{S}, PK)$ to generate an IBBE key k_B and randomly selects $b \in \{0,1\}$. He sets $k_b = k_B$ and k_{1-b} to a random value in $\mathcal{K}$. Finally, he computes two encryptions $ck_0 = Enc_{SE}(k_b, k_c)$, $ck_1 = Enc_{SE}(k_{1-b}, k_c)$ and gives B, ck_0, ck_1 to the attacker.

3. The attacker outputs b' and wins the game if $b = b'$.

In other words, the attacker wins if he correctly guesses whether the convergent key k_c of B is encrypted under k_B or a random value in $\mathcal{K}$. And we define attacker $\mathcal{A}$'s advantage in winning the game as $Adv_{\mathcal{A}} = |Pr[b = b'] - 1/2|$.

Since the IBBE scheme has been demonstrated to be $IND - sID - CPA$ secure in [27] using the General Decisional Diffie-Hellman Exponent (GDDHE) framework [29], we can easily obtain the following theorem.

Theorem 1. Convergent keys in KeyD are semantically secure if the IBBE scheme is secure under the GDDHE assumption.

Proof 1. Let the data block is all included in a dictionary $\mathbb{S}$. Suppose we have a semantic security attacker $\mathcal{A}$ for which $Adv_{\mathcal{A}} > \varepsilon$. We build an attacker $\mathcal{B}$ that breaks the $IND - sID - CPA$ security of IBBE where $Adv_{\mathcal{B}} > \varepsilon/(C^s_{s'} \cdot |\mathbb{S}|)$.

The reduction is immediate: $\mathcal{B}$ is given a block B and two encryptions ck_0, ck_1 (one of them is an encryption under an IBBE key and the other is under a random value in $\mathcal{K}$) of B's convergent key k_c, with the IBBE receiver set $\mathcal{S}$ of B, where $|\mathcal{S}| = s$. $\mathcal{B}$ randomly chooses another $s' - s$ identities and combine them with $\mathcal{S}$ to construct a "whole" receiver set $\mathcal{S}'$. $\mathcal{B}$ sends $\mathcal{S}'$ and s to $\mathcal{A}$ who then responds with a data block B_i and a receiver set $\mathcal{S}_i$ on which $\mathcal{A}$ wishes to be challenged. If $B_i \neq B$ or $\mathcal{S}_i \neq \mathcal{S}$ algorithm $\mathcal{B}$ reports failure and aborts. Otherwise, $\mathcal{B}$ sends the challenge B, ck_0, ck_1 to $\mathcal{A}$ who then responds with a $b' \in \{0,1\}$. The algorithm outputs b' as its response to the semantic security challenge. We can observe that $b' = b$ if algorithm $\mathcal{B}$ did not abort and $\mathcal{A}$'s response was correct. This probability is at least $1/2 + \varepsilon/(C^s_{s'} \cdot |\mathbb{S}|)$. Hence $Adv_B > \varepsilon/(C^s_{s'} \cdot |\mathbb{S}|)$.

6.2.3 Ownership Privacy

Users' ownership should be private to all other users, no matter they are malicious or honest users sharing identical data. In our KeyD scheme, the CSP is the only entity that knows the ownership of a data copy besides its owners. When a user with identity ID uploads a new data block B, he is apparently the first uploader, namely the IBBE broadcaster. The CSP will set the receiver set $\mathcal{S} = \{ID\}$ after finishing the whole uploading process. Note that ID is sent to the CSP as early as the upload request is launched, but not with the encrypted data. Else if B is existing on the cloud-side, the user will be informed of the duplication. At this time, the receiver set $\mathcal{S}$ and broadcaster header Hdr of B need to be updated. The former can be easily accomplished by the CSP setting the new receiver set $\mathcal{S}' = \mathcal{S} + \{ID\}$. And the latter is computed through the user's interaction with the CSP. Specifically, the CSP computes $C = h^{\prod_{j=1}^{s}(\gamma+H(ID_j))(\gamma+H(ID))}$ and sends C and the old Hdr to the user, then the user computes the new $Hdr' = C^k$ and returns Hdr' to the CSP. It can be seen from the above analysis that $\mathcal{S}$ will not appear in any communication. Therefore, the user cannot know who share(s) B with him during ownership updating process, and the malicious users cannot violate users' ownership privacy. On the other hand, deriving identities from Hdr or C is equivalent to solving the discrete logarithm problem for all users.

7 Performance Analysis

In this section, we evaluate the performance of our secure deduplication scheme KeyD in terms of the storage cost and the overhead of computation and communication. Since Li's Dekey [6], Wen's SKC and CKS [9], and our scheme all focus on the convergent key management problem of baseline approach in [6], we will give a comprehensive comparison between these schemes through theoretical and piratical analysis. In terms of the storage cost, we mainly focus on the cost that a data owner should pay for a block that he stores in the cloud. For communication and computation overhead, we will lay emphasis on the overhead for a data owner to upload a block. Some notations that will be used in the analysis of storage cost and communication overhead are listed in TABLE I. In order to facilitate comparison, we assume the default sizes of data file and data block are 10MB and 4 KB respectively. Since the scheme we proposed is a universal deduplication protocol, meaning which can generally handle all datasets with redundancy, we did not specify a dataset in the experiment. We use the hash function SHA-256 to generate a convergent key of size 32 bytes for each block, and also set the size of a tag to be 256 bits for the clearness of comparison. Moreover, we adopt the symmetric-key encryption algorithm AES-256 in Cipher-Block Chaining (CBC) mode as the default algorithm to encrypt convergent key as in [6]. For simplicity, we utilize PBC library [30] and choose type A paring with 80-bit security level to conduct the experiment on Linux Debian 4.6.0 with Intel Core i7-4790 CPU @ 3.60GHz and 20G memory. Finally, we use Amazon Web Service (AWS) as the cloud server and Amazon Elastic File System (EFS) [31] as the file storage system, which exactly takes 4 KB as the block size and can be accessed easily.

TABLE 1
Notations

$\|Tag\|$	Size of a file tag or a block tag
$\|c\|$	Size of an encrypted block
$\|ck\|$	Size of an encrypted convergent key
$\|G_1\|$	Size of an element in group G_1 or G_2
$\|Z_q^*\|$	Size of an element in group Z_q^*
m	Number of owners of a block
n, k, r	Three parameters in the Ramp secret sharing scheme (RSSS)

7.1 Storage Cost

In our KeyD, the CSP stores $\{T(B_i), c_i, d_i, ck_i, Hdr_i\}$ for each block B_i. Here d_i is the symmetric encryption of $k_i \in Z_q^*$ using the convergent key k_{c_i}, so its length is $O(|ck|)$. Hdr_i consists of C_{i1} and C_{i2}, which are elements in groups G_1 and G_2 respectively. Therefore, our total storage cost is $|Tag| + |c| + 2|ck| + 2|G_1|$. In Li's Dekey [6], they utilize (n, k, r)-RSSS (where $n > k > r \geq 0$) to generate n shares

2168-7161 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2869333, IEEE Transactions on Cloud Computing

from a convergent key, so their storage cost on the S-CSP and n KM-CSP is more than $|c| + 2n|Tag| + 2|ck|$, where $n \geq 3$ in their scheme. In Wen's SKC [9], the CSP stores $\{T(B_i), C_{h_i}, C_{b_i}\}$ for each block B_i, and the trusted GW has to store an $O(|q|)$-long session key K and a secret prime $r_i \in Z_q^*$. Therefore, the total storage cost of SKC is $|Tag| + |ck| + |c| + 2|Z_q^*|$. In their improved scheme CKS, the CSP just needs to store $T(B_i)$ and C_{b_i} for each B_i, but every GW needs to store m key shares $C_{i,j}$, which length is $O(|Z_q^*|)$. Then the actual storage cost of CKS is $|Tag| + |c| + m|Z_q^*|$, indicating that the storage cost that every owner needs to pay for B_i increases with the number of owners of B_i.

The storage cost for a data block of four schemes is listed in Table 2. According to our experiment hypothesis, $|Tag| = |ck| = 256$ bits, $|G_1| = 512$ bits and $|Z_q^*| = 160$ bits. Therefore, the storage cost of Dekey is more than our KeyD since the number of key-management servers n is at least 3. And the storage cost of SKC is just a little less than KeyD. The storage cost of Wen's improved scheme CKS is related to the number of data owners, making the comparison between CKS and KeyD not so intuitive. We suppose that each duplicate block is owned by m users, and define the duplication percentage P_D as the ratio of the number of duplicate blocks in a data file to the total number of blocks in that. The influence of P_D and m on the storage cost for a data file is shown in Fig.2. It can be seen from the figure that the storage cost in the CKS scheme increases with m, and we can compute that only when $m = 9.6$ (namely $m <= 9$) is the storage cost of CKS less than KeyD. Suppose that a 1GB data file owned by 1000 or 10000 users duplicates, their storage cost is much more than ours.

7.2 Communication Overhead

With regard to communication overhead, we consider two conditions, namely the block duplicates or not. In KeyD, if a data owner uploads a duplicated block B_i, he needs to interact with the CSP to add his identity ID into B_i's receiver set $\mathcal{S}_i$. If B_i is unique, the user just encrypts the block B_i and corresponding convergent key k_{c_i} and then uploads them. In Dekey, if B_i duplicates, the user just proves to the S-CSP and n KM-CSP his ownership and obtains secret shares, instead of sending anything else. But he should first send $T(B_i)$ to every KM-CSP and $T_j(F)$ to the j-th KM-CSP. If B_i is a new block, the user should compute n secret shares for the corresponding convergent key and then sends them to the S-CSP and n KM-CSP respectively. In SKC, if B_i duplicates, the trusted GW (a powerful user) will compute $x_j = (k_j)^{r_n} \bmod q$ for other $m - 1$ owners and broadcasts all of them to every owner. The owner will use the new common session key to encrypt the convergent key and sends it to the GW, who will further inform the CSS to replace the encrypted convergent key. If B_i does not duplicate, the session key will be established only between this user and the GW. In Wen's CKS, if B_i duplicates in two local communication networks (LCNs), four users in two LCNs need to exchange their key shares with the help of two GWs. Else the data block will be encrypted and sent to the GW, who will then forward it to the CSS. The concrete communication head of each scheme is given in Table 2.

TABLE 3
Comparison of Communication Overhead (Byte)

	duplicate	not duplicate
Dekey [6]	$64n$	$> 4160 + 96n$
SKC [9]	148	8424
CKS [9]	$384 + 40m$	8416
KeyD	384	4264

Combining the communication overhead in Table 2 and our experiment hypothesis, we can obtain Table 3. It can be easily concluded that our KeyD is better than Li's Dekey. This is because that the number of Key-Management Cloud Service Providers n in their scheme is at least 3. Their communication overhead for uploading a new block is at least 4.34KB (when $n = 3$) and increases with n, while ours is always 4.16KB. And only when $n <= 5$ is their communication overhead for uploading a duplicate block less than ours. With regard to SKC and CKS, their common disadvantage is that their communication overhead for uploading a new block is much more than ours. After all, the size of Tag, ck, element in G_1 and Z_q^* is much smaller compared with that of c. Take SKC as an example, their communication overhead for uploading a data file of 10MB is as small as us only when the duplication percentage P_D reaches 94.6%, computed by $\frac{10MB}{4kB} \times P_D \times 148B + \frac{10MB}{4kB} \times (1 - P_D) \times 8424B = \frac{10MB}{4kB} \times P_D \times 384B + \frac{10MB}{4kB} \times (1 - P_D) \times 4264B$. It means that 94.6% blocks of the data file to be uploaded have the same copies stored in the cloud, almost the whole file is duplicate.

7.3 Computation Overhead

We evaluate the computation overhead on client-side, based on whether the block to be uploaded duplicates or not. Let **Exp** denote one exponentiation in group G_1, **Pair** be one pairing operation on $e : G_1 \times G_2 \to G_T$, **Mul** represent one multiplication in group G_1, and **Add** indicate a modular addition in group Z_q^*.

In our KeyD, if the user is informed "block duplicates" when he uploads block B_i, he needs to interact with the CSP to compute a new header Hdr_i. If B_i is a new block, the user just encrypts it and the corresponding convergent key. And the user also needs to compute a header Hdr_i to include himself in the receiver set of B_i. If block duplicates in Li's Dekey, the user does not need to compute anything. However, if the block is a new one, the user needs to call (n, k, r)-RSSS to compute n secret shares for convergent key k_{c_i}. In SKC, if B_i duplicates, the GW and the user will interact with each other to negotiate a new common session key, which will be used to encrypt the convergent key. Otherwise, the user just computes a session key sharing with the GW, and uses this session key to encrypt convergent key and sends it to the GW. While in their second scheme CKS, when a user wants to upload a duplicate block, there are four users in two local communication networks needing to re-compute their secret shares. Else if the block is a new one, the user only needs to compute his own share. The specific computation overhead of four schemes is listed in Table 2.

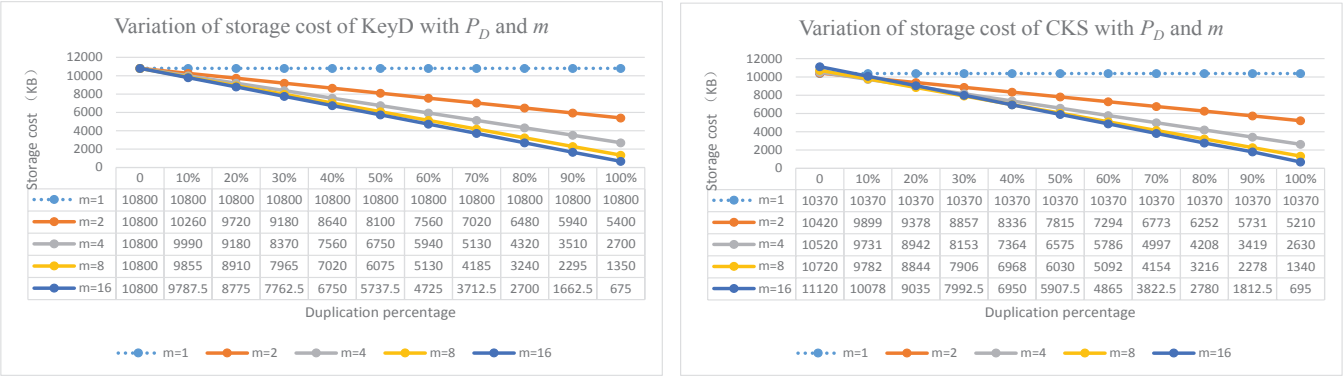
Our experiment results of the time required by a modular addition in Z_q*, multiplication and exponentiation in G_1,

2168-7161 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2869333, IEEE Transactions on Cloud Computing

TABLE 2
Performance Comparison

	Storage Cost	Communication Overhead		Computation Overhead	
		duplicate	not duplicate	duplicate	not duplicate
Dekey [6]	$> 2n\|Tag\| + \|c\| + 2\|ck\|$	$2n\|Tag\|$	$> 3n\|Tag\| + \|c\| + 2\|ck\|$	0	$n[(k-2)\mathbf{Exp} + (k-1)\mathbf{Mul} + (k-1)\mathbf{Add}]$
SKC [9]	$\|Tag\| + \|c\| + \|ck\| + 2\|Z_q^*\|$	$2\|Tag\| + 2\|ck\| + \|Z_q^*\|$	$4\|Tag\| + 2\|c\| + 2\|ck\| + 2\|Z_q^*\|$	$\mathbf{Exp} + (m-1)\mathbf{Mul}$	**Exp+Mul**
CKS [9]	$\|Tag\| + \|c\| + m\|Z_q^*\|$	$2\|Tag\| + (2m+16)\|Z_q^*\|$	$7\|Tag\| + 2\|c\|$	$4\mathbf{Exp} + 8\mathbf{Mul}$	**Exp+Mul**
KeyD	$\|Tag\| + \|c\| + 2\|ck\| + 2\|G_1\|$	$\|Tag\| + \|ck\| + 5\|G_1\|$	$2\|Tag\| + \|c\| + 2\|ck\| + 2\|G_1\|$	$2\mathbf{Exp} + 2\mathbf{Pair} + \mathbf{Mul}$	$4\mathbf{Exp} + \mathbf{Mul}$



(a) (b)

Fig. 2. Influence of duplication percentage and number of data owners on the storage cost. Because "$m = 1$" means that the data file is owned only by one user, the storage cost is independent of the duplication percentage and the "variation" is described by a dashed line.

and paring on $e : G_1 \times G_2 \rightarrow G_T$ are 2.72 us, 3.17 us, 1.08 ms and 0.75 ms, respectively. Therefore, the computation overhead caused by addition and multiplication is negligible compared to that by exponentiation and pairing operation. And a comparison of computation overhead of four schemes is given in Fig.3. Since the total computation time of scheme Dekey is related to n and k, we illustrate it in Fig.3.(a) and the other three schemes in Fig.3.(b). It can be seen from Fig.3 that computation overhead of Dekey increases with both n and k, and only when both of them are small is their computation overhead comparable with ours. Wen's first scheme SKC is better than ours, while their improved scheme CKS is not so practical. According to the figure, it is obvious that the computation overhead of CKS increases with duplication percentage P_D. It means that more blocks of the file that user wants to upload duplicate, should he spend more time in computation, which is apparently not a desirable scheme. Our computation overhead of uploading a duplicate file may be intuitively considered to be close to that of CKS, according to the column of computation overhead (duplicate) in Table 2 and the rightmost side of KeyD and CKS variation curves in Fig.3.(b). However, we can simplify multiple pairing operations as only one due to the bilinear property of bilinear map. Thus multiple verifications on client-side as described by Eq.8 can be realized by computing only twice bilinear paring as Eq.9, and our computation cost can be significantly reduced with the help of this batch verification. It can be seen from the variation curves of KeyD and KeyD (Batch verification) in Fig.3.(b) that the reduction of computation cost grows with duplication percentage P_D. In the most extreme case when the whole file is duplicate, the computation cost can be reduced from 9.37s to 5.55s, with a reduction of more than 40%.

$$\begin{cases} e(C_{12}', h) \stackrel{?}{=} e(C_{12}, h^{\gamma+H(ID)}) \\ e(C_{22}', h) \stackrel{?}{=} e(C_{22}, h^{\gamma+H(ID)}) \\ \vdots \\ e(C_{i2}', h) \stackrel{?}{=} e(C_{i2}, h^{\gamma+H(ID)}) \end{cases} \tag{8}$$

$$e(C_{12}'C_{22}' \cdots C_{i2}', h) \stackrel{?}{=} e(C_{12}C_{22} \cdots C_{i2}, h^{\gamma+H(ID)}) \tag{9}$$

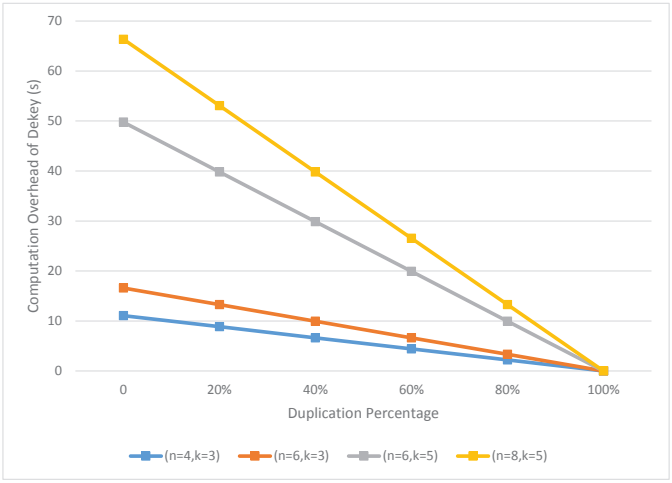
According to the analyses above, KeyD is a little inferior compared to Wen's SKC in storage cost and computation overhead, but much better than their improved scheme CKS and Li's Dekey. For communication overhead, our scheme performs better than other three schemes taking both duplicate and unique data upload into consideration. Besides, both SKC and CKS need trusted gateways to assist users in key agreement, with user interactions involved. And n key management servers are introduced in Dekey to manage convergent keys. Note that the space complexity of storage cost and communication overhead are all $O(n)$ in all four schemes if we use n to denote the file size, which can be easily seen from Fig.2 and Table 3. Besides, the computation overhead of four schemes has a relationship of at most twenty times according to Fig.3. In other words, the space complexity of storage cost and communication overhead of four schemes are in the same order of magnitude, so is the time complexity of computation overhead of them. Therefore, we do not emphasize the order of the complexity of storage cost, communication overhead, and computation overhead in this paper.

8 Conclusion and Future Work

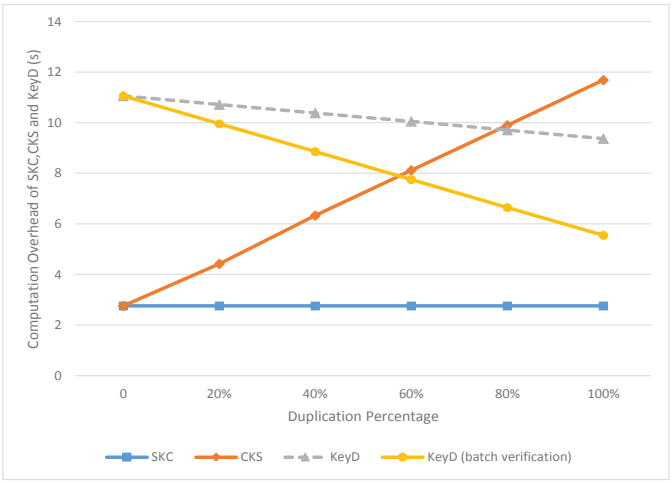
In this paper, we propose a secure client-side deduplication scheme KeyD to effectively manage convergent keys. Data

2168-7161 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2869333, IEEE Transactions on Cloud Computing



(a) Dekey



(b) SKC, CKS and KeyD

Fig. 3. Comparison of Computation Overhead

deduplication in our design is achieved by interactions between data owners and the Cloud Service Provider (CSP), without participation of other trusted third parties or Key Management Cloud Service Providers. The security analysis shows that our KeyD ensures the confidentiality of data and security of convergent keys, and well protects the user ownership privacy at the same time. Experimental results demonstrate that the security of our scheme is not at the expense of the performance. For our future work, we will try to seek ways to protect the identity privacy of data owners, which is not considered in our scheme.

ACKNOWLEDGMENTS

This work was supported in part by the National Information Security Special Projects of the National Development and Reform Commission of China under Grant (2012)1424, in part by the National Natural Science Foundation of China under Grant 61272481 and Grant 61572460, in part by the National Key R&D Program of China under Grant 2016YFB0800703, and in part by the Open Project Program of the State Key Laboratory of Information Security under Grant 2017-ZD-01.

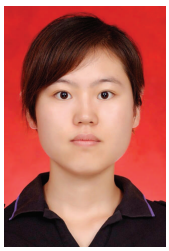
REFERENCES

[1] *Amazon Web Services*, [Online]. Available: https://aws.amazon.com/cn/.
[2] D.A. Sarma, X. Dong, and A. Halevy, *Bootstrapping pay-as-you-go data integration systems*[C]. ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, Bc, Canada, June. DBLP, 2008:861-874.
[3] J.R. Douceur, A. Adya, W.J. Bolosky, D. Simon, and M. Theimer, *Reclaiming Space from Duplicate Files in a Serverless Distributed File System*[C]. Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on. IEEE, 2002: 617-624.
[4] S. Ghemawat, H. Gobioff, and S. Leung, *The Google File System*[M]. SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles, 2003, 37(5): 29-43.
[5] D. Borthakur, *HDFS architecture guide*[J]. Hadoop Apache Project, 2008, 53.
[6] J. Li, X. Chen, M. Li, J. Li, P.P.C. Lee, and W. Lou, *Secure Deduplication with Efficient and Reliable Convergent Key Management*[J]. IEEE transactions on parallel and distributed systems, 2014, 25(6): 1615-1625.
[7] G.R. Blakley and C.A. Meadows, *Security of Ramp Schemes*[C]. Crypto. 1984, 84: 242-268.
[8] A.D. Santis and B. Masucci, *Multiple Ramp Schemes*[J]. IEEE Transactions on Information Theory, 1999, 45(5): 1720-1728.
[9] 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*[J]. IEEE transactions on computational social systems, 2015, 2(4): 137-147.
[10] W. Leesakul, P. Townend, and J. Xu, *Dynamic Data Deduplication in Cloud Storage*[C]. IEEE, International Symposium on Service Oriented System Engineering. IEEE, 2014:320-325.
[11] F. Rashid, A. Miri, and I. Woungang, *A secure data deduplication framework for cloud environments*[C]. Tenth International Conference on Privacy, Security and Trust. IEEE Computer Society, 2012:81-87.
[12] M. Bellare, S. Keelveedhi, and T. Ristenpart, *Message-Locked Encryption and Secure Deduplication*[C]. Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2013: 296-312.
[13] M. Abadi, D. Boneh, I. Mironov, A. Raghunathan, and G. Segev, *Message-Locked Encryption for Lock-Dependent Messages*[M]. Advances in CryptologyCCRYPTO 2013. Springer, Berlin, Heidelberg, 2013: 374-391.
[14] M.W. Storer, K. Greenan, D.D.E. Long, and E.L. Miller, *Secure Data Deduplication*[C]. Proceedings of the 4th ACM international workshop on Storage security and survivability. ACM, 2008: 1-10.
[15] P. Anderson and L. Zhang, *Fast and Secure Laptop Backups with Encrypted De-duplication*[C]. LISA. 2010.
[16] P. Puzio, R. Molva, M. Onen, and S. Loureiro, *ClouDedup: Secure Deduplication with Encrypted Data for Cloud Storage*[C]. Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on. IEEE, 2013, 1: 363-370.
[17] T. Jiang, X. Chen, Q. Wu, J. Ma, W. Susilo, and W. Lou, *Secure and Efficient Cloud Data Deduplication with Randomized Tag*[J]. IEEE Transactions on Information Forensics and Security, 2017, 12(3): 532-543.
[18] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, *Proofs of Ownership in Remote Storage Systems*[C]. Proceedings of the 18th ACM conference on Computer and communications security. Acm, 2011: 491-500.
[19] R.D. Pietro and A. Sorniotti, *Boosting Efficiency and Security in Proof of Ownership for Deduplication*[C]. Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security. ACM, 2012: 81-82.
[20] W.K. Ng, Y. Wen, and H. Zhu, *Private Data Deduplication Protocols in Cloud Storage*[C]. Proceedings of the 27th Annual ACM Symposium on Applied Computing. ACM, 2012: 441-446.
[21] J. Xu, E.C. Chang, and J. Zhou, *Weak Leakage-Resilient Client-Side Deduplication of Encrypted Data in Cloud Storage*[C]. Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. ACM, 2013: 195-206.
[22] J. Stanek, A. Sorniotti, E. Androulaki, and L. Kencl, *A Secure Data Deduplication Scheme for Cloud Storage*[C]. International Conference on Financial Cryptography and Data Security. Springer, Berlin, Heidelberg, 2014: 99-118.
[23] J. Li, Y.K. Li, X. Chen, P.P.C. Lee, and W. Lou, *A Hybrid Cloud Approach for Secure Authorized Deduplication*[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(5): 1206-1216.

2168-7161 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2869333, IEEE Transactions on Cloud Computing

[24] J. Liu, N. Asokan, and B. Pinkas, *Secure Deduplication of Encrypted Data without Additional Independent Servers*. Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, 2015: 874-885.
[25] M. Bellare, S. Keelveedhi, and T. Ristenpart, *DupLESS: Server-Aided Encryption for Deduplicated Storage*[J]. IACR Cryptology ePrint Archive, 2013, 2013: 429.
[26] M. Miao, J. Wang, H. Li, and X. Chen, *Secure Multi-Server-Aided Data Deduplication in Cloud Computing*[J]. Pervasive and Mobile Computing, 2015, 24: 129-137.
[27] C. Delerablée, *Identity-Based Broadcast Encryption with Constant Size Ciphertexts and Private Keys*[C]. International Conference on the Theory and Application of Cryptology and Information Security. Springer, Berlin, Heidelberg, 2007: 200-215.
[28] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, *Aggregate and Verifiably Encrypted Signatures from Bilinear Maps*[C]. Eurocrypt. 2003, 2656: 416-432.
[29] D. Boneh, X. Boyen, and E.J. Goh, *Hierarchical Identity Based Encryption with Constant Size Ciphertext*[C]. Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2005: 440-456.
[30] *PBC Library-The Pairing-Based Cryptography Library*, [Online]. Available: https://crypto.stanford.edu/pbc/.
[31] *Amazon Elastic File System (EFS)-Scalable, reliable, and elastic file storage for the AWS Cloud*, [Online]. Available: https://aws.amazon.com/efs/?nc1=h_ls.



Ling Liu received the B.S. degree from the School of Telecommunications Engineering, Xidian University, in 2013. She is currently pursuing her Ph.D. degree at the School of Cyber Engineering, Xidian University. Her research interests include cloud computing security, network and system security, and applied cryptography.

Yuqing Zhang received his Ph.D. degree in Cryptography from Xidian University, China. Dr. Zhang is a Professor in the School of Cyber Engineering at Xidian University, and the Director of the National Computer Network Intrusion Protection Center at University of Chinese Academy of Sciences. His research interests include network and system security, and applied cryptography. He has published more than 100 research papers in international journals and conferences, such as ACM CCS, IEEE TPDS, and IEEE TDSC. His research has been sponsored by NSFC, Huawei, Qihu360 and Google. He has served as program chair for more than 5 international workshops (e.g., SMCN-2017), and PC member for more than 10 international conferences in networking and security, such as IEEE Globecom 16/17, IEEE CNS 17, and IFIP DBSec 17.

Xuejun Li received her Ph.D. degree from Northwestern Polytechnical University, China, in 2003. In the same year, she joined the State Key Laboratory of Information Security in the University of Chinese Academy of Sciences as a postdoctor and finished her postdoctoral work in August 2005. Dr. Li is an Associate Professor in the School of Cyber Engineering at Xidian University. She had been a visiting scholar at Wayne State University in the United States from September 2012 to September 2013. Her main research interests are cryptography and information security, including elliptic curve cryptography, identity cryptography, cryptographic protocol design and analysis.

2168-7161 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.