Towards Secure Data Distribution Systems in Mobile Cloud Computing

Jiang Zhang, Zhenfeng Zhang, and Hui Guo

Abstract—Though the electronic technologies have undergone fast developments in recent years, mobile devices such as smartphones are still comparatively weak in contrast to desktops in terms of computational capability, storage etc, and are not able to meet the increasing demands from mobile users. By integrating mobile computing and cloud computing, mobile cloud computing (MCC) greatly extends the boundary of the mobile applications, but it also inherits many challenges in cloud computing, e.g., data privacy and data integrity. In this paper, we leverage several cryptographic primitives such as a new type-based proxy re-encryption to design a secure and efficient data distribution system in MCC, which provides data privacy, data integrity, data authentication, and flexible data distribution with access control. Compared to traditional cloud-based data storage systems, our system is a lightweight and easily deployable solution for mobile users in MCC since no trusted third parties are involved and each mobile user only has to keep short secret keys consisting of three group elements for all cryptographic operations. Finally, we present extensive performance analysis and empirical studies to demonstrate the security, scalability, and efficiency of our proposed system.

Index Terms—Mobile cloud computing, secure data distribution, data integrity, access control, proxy re-encryption

1 Introduction

In cloud computing, many computing resources are provided as services over the internet. One of the main services provided by clouds is storage (e.g., Simple Storage Services—Amazon S3 [1]), which allows users to store their enormous amount of data to the remote clouds without bothering the complex management of storage hardware. Outsourcing big data to clouds provides many benefits, e.g., low costs, good reliability and availability, but the data security issues such as privacy and integrity brought by third party's cloud systems have been the major concerns for users utilizing such services. According to the surveys [25], [31], more than 90% of US consumers wants to be asked to give permission for their data to be shared, and 88% of all potential consumers are worried about the privacy of their data. Since the data is stored and managed in the cloud, the data security highly depends on the IT management of the cloud services providers, and any security loophole in the cloud system might damage the security of the users' private data, e.g., [30], [38].

Nowadays, it becomes very common and popular to access cloud services by using mobile devices. By a recent study [2], cloud applications will account for 90% of total mobile data traffic by 2018. To offload storage to the cloud, there are many existing storage services for mobile devices, such as Dropbox, Box, iCloud, Google Drive, and Skydrive [17]. Since mobile cloud computing (MCC) integrates mobile computing and cloud computing, all the above security issues in cloud computing are inherited in MCC with the extra resource limited mobile devices [28]. For example, in September 2014, a loophole in iCloud causes many private photos of iphone users to be downloaded by hackers [27]. Thus, it is very necessary to design a framework to ensure the security of private data in cloud storages. Besides, the limitations of mobile devices such as low storage, unpredictable Internet connectivity and less energy also require a lightweight solution in MCC that provides security with minimum communication and processing overhead, which may make MCC services significantly different from cloud services for desktops [28], [33]. Recently, Huang et al. [22] gave four principles for shifting mobile clouds from the traditional Internet clouds, where the first principle requires that MCC applications should be designed in such a way that a user can control their own data with strong privacy and security protection.

The above security challenges in MCC and the current state of cloud storage systems such as [18], [40], [49], [53], [55] suggest that it is not an easy task to design an efficient and secure data distribution system in MCC. However, we observe that most mobile devices are currently in personal use, which usually does not need a data distribution system as powerful as that for business desktops. For example, in contrast to enterprise users in cloud storage who have to share hundreds of gigabytes of data among thousands of enterprise employees, a smartphone user usually shares many megabytes of photos with small number of friends. This fact makes it possible to design a lightweight data distribution system. In addition, as the developments of electronic technologies, the capability of mobile devices is greatly enhanced. One should not overlook the tradeoff between the energy consumption on the mobile devices and the expense of using cloud resources while designing a security framework [28].

In this work, we present an efficient data distribution system in MCC, which allows mobile users to securely store their data in the cloud storage, and flexibly share their

- *Jiang Zhang and Hui Guo are with State Key Laboratory of Cryptology, P.O. Box 5159, Beijing, 100878, China. E-mail: jiangzhang09@gmail.com, guohui@tca.iscas.ac.cn.*
- *Zhenfeng Zhang is with Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, 100190, China. E-mail: zfzhang@tca.iscas.ac.cn.*

data with friends. We leverage several cryptographic primitives to achieve data privacy, data integrity, dynamical data modification and deletion, and flexible data distribution. Concretely, we first design an efficient type-based proxy re-encryption (TB-PRE), which allows a mobile user with a single secret key to keep the data privacy, and flexibly share his data with friends under permission. We also use the BLS signature [12] to both protect the integrity of the data and provide authentication to the data. By combining the BLS signature with Merkle hash tree (MHT), our system allows the data owner to dynamically modify and delete his data, and enables the data consumer to efficiently authenticate the data owner's identity. In addition, our data distribution system is designed without any trusted third party (TTP) such that a mobile user can fully control his own data, and easily share it with friends. Actually, establishing a TTP might be a little troublesome when a smartphone user simply wants to share some pictures with his close friends.

Our contributions are summarized as follows:

- We present a new type-based proxy re-encryption (TB-PRE) with provable security, which allows mobile data owners to classify their data, and share their data in a fine-grained way. Compared with previous TB-PRE schemes in the literature, our scheme has a good overall performance.
- We design a data distribution system in MCC without TTP, which has the following useful features:
  - The data owner can arbitrarily classify his data into different categories and encrypt them under a single public key. The data owner can also check the integrity of the encrypted data, and dynamically modify or delete the encrypted data in the cloud.
  - The data owner can permit data consumers to access his private data by the category of the data. More specifically, the data owner can allow different data consumers to access different set of data categories.
  - The data consumer only has to interact with the data owner one time to get access permission for each data category. In other words, the data owner is not required to be online during the data distribution phase after granting the access permission. [1]
  - Any user without permission to access a particular data category of the data owner cannot read the data in the category, while a permitted data consumer can read the data, authenticate the data owner's identity, and check the integrity of the data in the category.
  - All the users only have to keep secret keys consisting of three group elements to perform all cryptographic operations (e.g., at most 96 bytes for 128-bit security), which is independent of the number of total users in the system.
- We give an extensive performance analysis and a proof-of-concept implementation to demonstrate the efficiency of our system in terms of storage, communication and computation overheads on both sides of the cloud server and the mobile devices.

1. Due to the limitations of mobile devices such as low storage, unpredictable Internet connectivity and less energy, it is somewhat restrictive to require the data owner to be online all the time. In fact, one of our main goals is to reduce the number of interactions between the data owner and the data consumers. However, in a system which allows dynamic data operations, no interactions after the access permission may cause some inherent problems. In remark 3, we will give more discussions on this.

The rest of this paper is organized as follows. Section 2 gives some discussions with related work. Section 3 introduces some preliminaries. In Section 4, we describe the system model, the security model and the design goals of our data distribution system in MCC. A new TB-PRE scheme is proposed in Section 5. We give our data distribution system in Section 6, and present an extensive performance analysis in Section 7. Finally, we conclude in Section 8.

2 Related Work

There are a lot of works focusing on designing systems that delegate the storage to a remote cloud. The first branch of research tried to ensure the integrity of the data in the cloud [5], [14], [26], [42], [46], such that a dishonest cloud server cannot prove the possession of the owner's data if the data is broken (e.g., due to system errors) or has been discarded on purpose (e.g., for economic benefits). To reduce the heavy data auditing or check computation of the data owner, a third trusted auditor is usually introduced [47], [51]. However, such a solution seems simply transfer the trust from the cloud to a third (trusted) auditor. Besides, some of the works [5], [53], [57] considered the data privacy with respect to the third auditor, but they usually do not keep the confidentiality of the data against the cloud (i.e., the owner's data is simply stored without privacy protection against the cloud services providers, e.g., [52], [53], [55]).

Encryption is a common approach to achieve data privacy, but encryption alone cannot solve the problem [33]. Actually, traditional data encryption scheme limits the data access in the sense that it only allows the user with corresponding decryption key to read the data. In order to realize data-sharing with access control over the private data in the cloud, advanced cryptographic encryptions such as broadcast encryption (BE) [19], [32], attribute-based encryption (ABE) [20], [43] and proxy re-encryption (PRE) [10], [13], [35] have been employed in the design of cloud storage systems.

Since broadcast encryption (BE) usually involves a group manager, and needs to determine the group of the data consumers before the data is uploaded to the cloud, the solution of using BE to ensure secure data-sharing usually cannot support dynamic data distribution [9]. This is not applicable to the case that the data owner does not know the information of the potential data consumers when he uploads the data to the cloud. Besides, if the group manager is a third party (i.e., not the data owner himself), this solution may incur the key escrow problem since the group manager can read the data of all the group members, e.g., [37].

In recent years, many researchers have resorted to attribute-based encryption (ABE) to enforce fine-grained access control in cloud storage [20], [24], [41], [49]. On the one hand, those systems support dynamic and flexible access control based on the attributes of the data consumers. But on the other hand, an ABE system relying on a third party authority to issue secret keys for all system users usually suffers from the key escrow problem [34]. The use of multi-authorities ABE, e.g., [15], can somehow ease the worry of this problem, but this general method cannot completely solve the key escrow issue. By letting the data owner himself be the attribute authority, the above key escrow problem can be avoided [56], but the number of secret keys of each data consumer in such a solution is usually linear to the number of the data owners. This might be a big bottleneck in open networks where each user may potentially be a data owner. Actually, the above problems as well as the inefficiency of existing ABEs in terms of computation and communication overheads have become the major obstacle in deploying ABE in real applications.

Proxy re-encryption (PRE) is another solution to securely share data on public clouds [6], [23], [36], [50]. A significant benefit brought by PRE is that each user only has to keep a single secret key and does not suffer from the key escrow problem. Concretely, in a PRE system, a data owner Alice can generate a re-encryption key to help Bob transform a ciphertext under her own public key to ciphertext of the same message under Bob's public key, such that Bob can decrypt it by using his own secret key to obtain the original message. However, the access control provided by a traditional PRE (including ID-based PRE [16], [21]) is in an "all or nothing" manner [45]. Namely, a user with a corresponding re-encryption key can read all the data of the data owner, but a user without the re-encryption key cannot read any private data of the data owner. This is not applicable to the case that the data owner only wants to share part of his data with others. In order to realize flexible access control, a new variant of PRE, Type-based PRE (TB-PRE), was introduced in [35], [45], [48], [54]. In contrast to traditional PRE, TB-PRE allows the data owner to classify his data into different categories [6], and share the data of a particular category with data consumers without affecting the privacy of the data in other categories.

In this paper, we use TB-PRE to enforce access control for mobile devices in cloud computing due to the following three reasons: 1) Theoretically, TB-PRE is less powerful than ABE in terms of access control, but it is enough for many applications where the data is naturally classified into different categories for different users. For example, users may share different types of photos/articles with different friends in social networks; 2) The state of the art suggests that TB-PRE can be more efficient than ABE, and thus is more appealing to devices with limited capacity; 3) A TB-PRE system does not suffer from the key escrow problem, and each user in the system only has to keep a single pair of public and secret keys of his own.

To the best of our knowledge, our work is the first to use TB-PRE in designing a concrete secure data distribution system in MCC. In fact, several TB-PREs were proposed in the literature [35], [45], [48], [54], but no concrete data distribution system using TB-PRE was given. In order to obtain a practical data distribution system in MCC, we first design an efficient TB-PRE scheme, and then carefully combines it with several other cryptographic primitives such as signatures to obtain a secure and efficient data distribution system with fine-grained access control in MCC. Since we use TB-PRE as a building block, one can instantiate our system with other TB-PRE schemes in the literature. However, detail comparisons show that our TB-PRE scheme has the highest overall performance. We also give an extensive performance analysis and a proof-of-concept implementation to demonstrate the efficiency of our system in terms of storage, communication and computation overheads on both sides of the cloud server and the mobile devices, which indicates that it could be very practical.

3 Preliminaries

If x is a string, $|x|$ denotes its length, and if S is a set, $|S|$ denotes its size. Denote $x\|y$ as the bit concatenation of two strings $x, y \in \{0,1\}^*$. We use 1^k to denote the string of k ones for some positive integer k. We use the notation $\leftarrow$ to denote randomly choosing an element from some set (distribution) or indicate the output of some algorithm. For example, $s \leftarrow S$ means that we randomly choose an element s from the set (distribution) S, and $z \leftarrow \mathcal{A}(x, y, \dots)$ means that the output of algorithm $\mathcal{A}$ with inputs $x, y, \dots$, is z.

We will use bilinear groups, the BLS signature [12], as well as Merkle Hash Tree [39] in this work. Since those primitives are standard in the literature, we defer the details to the supplemental material for space reason.

4 Problem Statement

4.1 System Model

There are three main network entities in our data distribution system, namely, the cloud, the data owner and the data consumer as illustrated in Fig. 1.

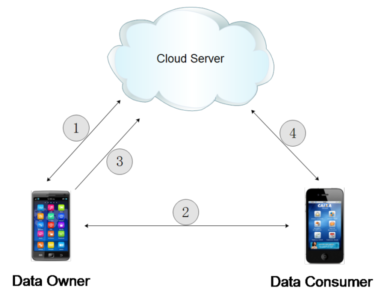

Fig. 1. System Model of Data Distribution System (①: perform data operations such as upload and modification; ②: ask for data access permission; ③: upload permissions to help data distribution; ④ distribute data to the consumer)

- The data owner is a mobile user who stores his private data in the cloud (by different categories), and allows the data consumer to access his private data (of some category) from the cloud.
- The cloud is an entity who provides storage services and is responsible to help the data owner to distribute the private data (belonging to some particular category) to the data consumer.
- The data consumer is an entity who first gets data access permission (of some data category) from the data owner (and this only happens once per data

category), and then access the data owner's private data from the cloud.

In mobile cloud computing (MCC), the data produced in mobile devices such as smartphones usually contains many personally sensitive data, e.g., location information. The data owner may want to protect his private data from the cloud services providers such that the compromise of the cloud servers will not damage the privacy of the data. Besides, the private data of the data owner might belong to different categories, which possibly has different potential data consumers. This requires a fine-grained access control based on the categories of the private data. For example, the private data shared by the data owner might be significantly different for his family members and workmates.

Due to the limited storage and computation capability, the mobile data owner would like to store his private data to the cloud without backup in the local storage of the device. In this case, it is very necessary for the data owner to check the integrity of his data when retrieving it from the cloud. For some applications, it is also required for the data consumer to check the integrity of the data or authenticate the data owner's identity. Besides, the data owner might want to perform data operations (such as modification and deletion) on his private data storing in the cloud.

Another limitation in MCC is the unpredictable Internet connectivity, which requires that the mobile users should perform the data operations without involving too many interactions between the entities. This is one of the reasons why we do not resort to a trusted third party (TTP) to relax the computation overhead of the mobile users, because it may potentially increase the interactions between system users. Besides, it is preferable for some applications to achieve data distribution without interactions between the data owner and the data consumer. For example, a social network user might want to access the photos shared by his friends who are currently offline.

4.2 Security Model

In this paper, we consider two types of adversaries against our data distribution system: the dishonest cloud and the malicious data consumers. For a dishonest cloud, it might want to compromise the privacy of the data without knowing by the data owner, e.g., employing a data mining on the users' private data to find user's preferences for its own (commercial) interests. The cloud might also want to break the integrity of the data, e.g., hiding data error/loss incidents [7], [26], [44] from the data owner for protecting its reputation, or discarding rarely accessed data for saving storage resources [5], [47]. In addition, the cloud might ignore data operations such as data modification for saving computation resources [9].

For malicious data consumers, the direct target is to access the private data without obtaining the access permission from the data owner. This includes malicious data consumers either without gaining any data access permission, or having gained the permission to some particular data categories but trying to access data belonging to other categories. Besides, the dishonest cloud and the malicious data consumers can collude to launch the above attacks. We emphasize that the cloud colluding with any permitted data consumer for a specific data category to access the data belonging to that category is not considered as an attack, since this is allowed by the functionality of any data distribution system. However, the cloud colluding with a permitted data consumer for one data category to access any data belonging to other categories is an attack, and thus should be prevented.

4.3 Design Goals

In this subsection, we present the main goals of the proposed data distribution system:

- Data confidentiality: Any user (including the cloud) without access permission cannot collude to read the private data of the data owner.
- Data integrity: Both the data owner and the data consumer can check the integrity of the data.
- Data authentication: The data consumer can authenticate the data owner's identity.
- Dynamic data operation: The data owner can perform data modification and deletion operations without affecting the confidentiality, integrity and authentication properties.
- Fine-grained data distribution: The data consumer can read the private data belonging to a category if and only if he obtains the access permission to that specific data category from the data owner.
- Lightweight: Both the data owner and the data consumer should perform all the operations with small storage, communication and computation overhead. For example, the data owner is not necessarily online all the time, and the data consumer only has to interact with the data owner once per data category (to get the corresponding access permission).

5 A New Type-based Proxy Re-encryption

We first briefly review the definition of type-based proxy re-encryption (TB-PRE), and then give our new construction.

5.1 Definition

A single-hop unidirectional TB-PRE scheme consists of the following algorithms [35], [48]:

- $\texttt{Setup}(1^k)$: Taking a security parameter 1^k as input, the setup algorithm outputs a public parameter $param$, which specifies the plaintext space $\mathcal{P}$ and the type space $\mathcal{T}$.
- $\texttt{KeyGen}(param, i)$: Taking a parameter $param$ and a user identity i as inputs, the key generation algorithm outputs a pair of public key and secret key (pk_i, sk_i) for user i.
- $\texttt{ReKeyGen}(sk_i, pk_j, t)$: Taking a secret key sk_i of user i, a public key pk_j of user j, and a type $t \in \mathcal{T}$ as inputs, the re-encryption key generation algorithm outputs a unidirectional re-encryption key $rk_{i \to j,t}$.
- $\texttt{Enc}(pk_i, t, m)$: Taking a public key pk_i of user i, a type $t \in \mathcal{T}$ and a message $m \in \mathcal{P}$ as inputs, the encryption algorithm outputs a ciphertext C_i.
- $\texttt{ReEnc}(rk_{i \to j,t}, C_i)$: Taking a re-encryption key $rk_{i \to j,t}$ and a ciphertext C_i under pk_i as inputs,

the re-encryption algorithm outputs a re-encrypted ciphertext C_j under pk_j.

- $\mathtt{Dec}(sk_i, C_i)$: Taking a secret key sk_i of user i, and a ciphertext C_i under pk_i as inputs, the decryption algorithm outputs a message $m \in \mathcal{P}$ or an error symbol $\perp$ indicating the failure of the decryption.

For correctness, we require that for any key pairs (pk_i, sk_i) of user i and (pk_j, sk_j) of user j, a re-encryption key $rk_{i\to j,t}$, a type $t \in \mathcal{T}$, a message m, a ciphertext $C_i = \mathtt{Enc}(pk_i, t, m)$, and a re-encrypted ciphertext $C_j = \mathtt{ReEnc}(rk_{i\to j,t}, C_i)$, the following condition holds with overwhelming probability.

$$\mathtt{Dec}(sk_i, C_i) = \mathtt{Dec}(sk_j, C_j) = m.$$

5.2 Our Construction

Let k be the security parameter and $l = 2k$, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be groups of prime order p, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear pairing. Let $\mathcal{H}_0 : \{0,1\}^* \to \mathbb{Z}_p^*$, $\mathcal{H}_1 : \{0,1\}^l \to \mathbb{Z}_p$, $\mathcal{H}_2 : \mathbb{G}_T \to \{0,1\}^l$, and $\mathcal{H}_3 : \mathbb{G}_1 \times \{0,1\}^* \to \mathbb{G}_2$ be families of hash functions. The information flow of our scheme between the involved parties is given in Fig. 2, followed with a full description of each operation.

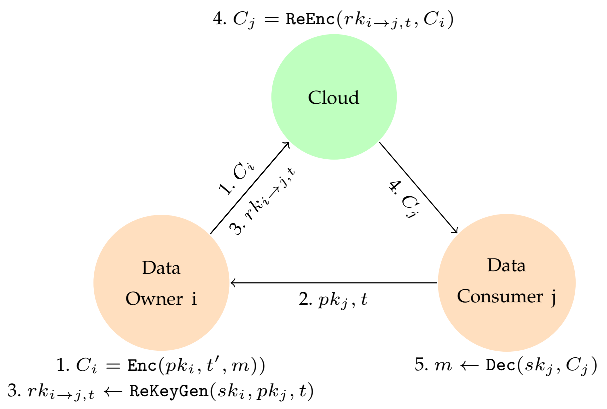
4. $C_j = \mathtt{ReEnc}(rk_{i\to j,t}, C_i)$
Cloud
1. C_i
3. $rk_{i\to j,t}$
4. C_j
Data Owner i
2. pk_j, t
Data Consumer j
1. $C_i = \mathtt{Enc}(pk_i, t', m))$
3. $rk_{i\to j,t} \leftarrow \mathtt{ReKeyGen}(sk_i, pk_j, t)$
5. $m \leftarrow \mathtt{Dec}(sk_j, C_j)$

Fig. 2. The information flow of our TB-PRE between the involved parties

- $\mathtt{Setup}(1^k)$: Given the security parameter 1^k, uniformly choose $g \leftarrow \mathbb{G}_1, h \leftarrow \mathbb{G}_2$, $H_0 \leftarrow \mathcal{H}_0$, $H_1 \leftarrow \mathcal{H}_1$, $H_2 \leftarrow \mathcal{H}_2$ and $H_3 \leftarrow \mathcal{H}_3$ at random, and compute $Y = e(g, h)$. Finally, set the message space $\mathcal{P} = \{0,1\}^k$, and the type space $\mathcal{T} = \{0,1\}^*$, and return the system parameter $param = (g, h, Y, H_0, H_1, H_2, H_3, \mathcal{P}, \mathcal{T})$.
- $\mathtt{KeyGen}(param, i)$: Given the system parameter $param$ and a user identity i, uniformly choose $\alpha_i, \beta_i \leftarrow \mathbb{Z}_p$ at random, compute the public key components $pk_{i,1} = g^{\alpha_i}, pk_{i,2} = h^{\beta_i}$, and set the secret key components $sk_{i,1} = \alpha_i, sk_{i,2} = \beta_i$. Finally, return the public and secret key pairs $(pk_i, sk_i) = ((pk_{i,1}, pk_{i,2}), (sk_{i,1}, sk_{i,2}))$.
- $\mathtt{ReKeyGen}(sk_i, pk_j, t)$: Given a secret key sk_i, a public key pk_j, and a type t, compute $rk_{i\to j,t} = pk_{j,2}^{\frac{1}{H_0(t)+sk_{i,1}}}$. Return the re-encryption key $rk_{i\to j,t}$.
- $\mathtt{Enc}(pk_i, t', m)$: Given a public key pk_i, a message $m \in \mathcal{P}$, and a type t', uniformly choose $\rho \leftarrow \{0,1\}^k$ at random, compute $r \leftarrow H_1(m\|\rho)$, $c_0 = (g^{H_0(t')}pk_{i,1})^r, c_1 = H_2(Y^r) \oplus (m\|\rho)$ and $c_2 = H_3(c_0, c_1, t')^r$. Finally, return the ciphertext $C_i = (t', c_0, c_1, c_2)$.
- $\mathtt{ReEnc}(rk_{i\to j,t}, C_i)$: Given a re-encryption key $rk_{i\to j,t}$, a ciphertext $C_i = (t, c_0, c_1, c_2)$, the algorithm checks if $e(c_0, H_3(c_0, c_1, t)) = e(g^{H_0(t)}pk_{i,1}, c_2)$ holds. If not, it returns $\perp$ and report failures, else computes $c_0' = e(c_0, rk_{i\to j,t}) = e(g, pk_{j,2})^r$, and returns the re-encrypted ciphertext $C_j = (c_0', c_1)$.
- $\mathtt{Dec}(sk_i, C_i)$: Given a secret key sk_i and a ciphertext C_i, the algorithm distinguishes the following cases:
  - If C_i is an original ciphertext, parse $C_i = (t, c_0, c_1, c_2)$, and check if $e(c_0, H_3(c_0, c_1, t)) = e(g^{H_0(t)}pk_{i,1}, c_2)$ holds. If not, return $\perp$, else compute $K = e(c_0, h^{\frac{1}{H_0(t)+sk_{i,1}}})$, and $m\|\rho = c_1 \oplus H_2(K)$. Finally, return m if $c_2 = H_3(c_0, c_1, t)^{H_1(m\|\rho)}$, else return $\perp$.
  - If C_i is a re-encrypted ciphertext, parse $C_i = (c_0, c_1)$, and compute $K = c_0^{1/sk_{i,2}}$, and $m\|\rho = c_1 \oplus H_2(K)$. Finally, return m if $c_0 = e(g, pk_{j,2})^{H_1(m\|\rho)}$, else return $\perp$.

In the following, we will briefly show the correctness and the security of our scheme. Then, we give an optimization and a comparison of our scheme with existing TB-PREs.

5.2.1 *Correctness*

An original ciphertext C_i under pk_i and type t has a form of $(c_0 = (g^{H_0(t)}pk_{i,1})^r, c_1 = H_2(Y^r) \oplus (m\|\rho), c_2 = H_3(c_0, c_1, t)^r)$ where $r = H_1(m\|\rho)$ and $pk_{i,1} = g^{sk_{i,1}}$, we always have $e(c_0, H_3(c_0, c_1, t)) = e(g^{H_0(t)}pk_{i,1}, c_2)$ and $K = e(c_0, h^{\frac{1}{H_0(t)+sk_{i,1}}}) = e((g^{H_0(t)}pk_{i,1})^r, h^{\frac{1}{H_0(t)+sk_{i,1}}}) = e(g, h)^r = Y^r$. Thus, the decryption of C_i is correct.

In addition, given a re-encryption key $rk_{i\to j,t}$, an original ciphertext C_i can be transformed into a re-encrypted ciphertext C_j under pk_j with a form of $(c_0 = e(g, pk_{j,2})^r, c_1 = H_2(Y^r) \oplus (m\|\rho)$, where $r = H_1(m\|\rho)$. Thus, we have $K = c_0^{1/sk_{j,2}} = e(g, pk_{j,2}^{1/sk_{j,2}})^r = Y^r$ for the re-encrypted ciphertext. This shows that the decryption of $C_j$is correct.

5.2.2 *Security*

The security of a TB-PRE scheme requires that the cloud with the re-encryption keys and a malicious user without permissions (or with access permission to data type t) cannot collude to compromise the privacy of the encrypted data (with data type $t' \neq t$). We prove that the above construction is CCA-secure against any dishonest cloud and malicious users under the q-BDHI assumption [11] in Section 3 of the supplemental material, please refer to it for details.

5.2.3 *Optimization*

One pairing can be eliminated in the decryption of an original ciphertext by randomly choosing $x \leftarrow \mathbb{Z}_p$ and computing $K = e(c_0, h^{\frac{1}{H_0(t)+sk_{i,1}}} \cdot H_3(c_0, c_1, t)^x)/e((h^{H_0(t)}pk_{i,1})^x, c_2)$ without computing the check $e(c_0, H_3(c_0, c_1, t)) = e(h^{H_0(t)}pk_{i,1}, c_2)$. This is possible since if the check does not

TABLE 1
Storage and computation overhead comparison among related TB-PRE schemes (where we roughly have $t_v \approx t_{dec} \geq t_p \gg t_{me} > t_s \geq t_e$, $t_{enc} \geq 2t_e$, and $|vk| \approx |\sigma| \geq |\mathbb{G}| \geq 2k$, $|\mathbb{G}_T| \approx |C_{pke}| \geq 2|\mathbb{G}|$ for system security parameter k. Besides, we use the same symmetric pairing group notations, i.e., $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$, for simplicity.)

			Our Construction	Tang [48]	Weng et al. [54]	LV11 [35]	Seo et al. [45]
Storage	RK		$\|\mathbb{G}\|$	$\|C_{pke}\| + \|\mathbb{G}\|$	$2\|\mathbb{G}\|$	$2\|\mathbb{G}\|$	$\|\mathbb{G}\|$
	C_1		$2\|\mathbb{G}\| + k$	$\|\mathbb{G}_T\| + 2\|\mathbb{G}\|$	$\|\mathbb{G}_T\| + 2\|\mathbb{G}\|$	$\|vk\| + 3\|\mathbb{G}\| + \|\sigma\|$	$\|vk\| + 2\|\mathbb{G}\| + \|\sigma\|$
	C_2		$\|\mathbb{G}_T\| + k$	$2\|C_{pke}\| + \|\mathbb{G}_T\| + \|\mathbb{G}\|$	$\|\mathbb{G}_T\| + 2\|\mathbb{G}\|$	$\|vk\| + 7\|\mathbb{G}\| + \|\sigma\|$	$\|vk\| + 4\|\mathbb{G}\| + \|\sigma\|$
Computation	Enc		$t_{me} + 2t_e$	$t_p + 3t_e$	$t_p + 3t_e$	$2t_{me} + t_s + 2t_e$	$2t_{me} + t_s + t_e$
	ReEnc		$3t_p + t_e$	$3t_p + t_{enc}$	$3t_p$	$t_v + 2t_p + 2t_{me} + 6t_e$	$t_v + 2t_p + t_{me} + 4t_e$
	Dec	C_1	$2t_p + 2t_{me} + t_e$	$3t_p + 2t_e$	$3t_p + 2t_e$	$t_v + 3t_p + 2t_{me} + t_e$	$t_v + 3t_p + 3t_e$
		C_2	$2t_e$	$2t_{dec} + t_p + t_e$	$t_p + 3t_e$	$t_v + 9t_p + 2t_e$	$t_v + 4t_p + 2t_e$

hold, one will obtain a value $K' \neq K$ uniformly distributed over $\mathbb{G}_T$, and thus it is unlikely to pass the second check $c_2 = H_3(c_0, c_1, t)^{H_1(m\|\rho)}$ since $c_1 = H_2(K) \oplus (m\|\rho)$.

5.2.4 Comparison

We give a comparison of our scheme with previous TB-PRE schemes [35], [45], [48], [54] in Table 1, which shows that our construction has small storage and computation overhead. For the notations, let k be the security parameter. We use $|\cdot|$ as the bit-length function, e.g., $|\mathbb{G}|$ equals the least bit-length used for representing the element in the cyclic group $\mathbb{G}$, which is usually greater than $2k$ for solving the discrete-log (DL) problem with complexity at least 2^k. We use RK, C_1 and C_2 to denote the re-encryption key, the original ciphertext, and the re-encrypted ciphertext, respectively. Moreover, the storage overhead of the ciphertexts does not take account of the length required for the messages and types since they are invariable for all schemes. Let vk and σ denote the verification key and the signature of the one-time signature scheme used in [35] and [45]. We also denote t_p, t_e, t_{me}, t_s and t_v as the time for computing a bilinear pairing, an exponentiation, a multi-exponentiation in the bilinear group, a signing algorithm and a verification algorithm, respectively. Besides, since a CCA-secure public key encryption (PKE) scheme is required in Tang's scheme [48], we use C_{pke} to denote the ciphertext, and use t_{enc}, t_{dec} to denote the time used for encryption and decryption of the PKE scheme. The values for $t_s, t_v, t_{enc}, t_{dec}$, and the lengths of vk, σ, C_{pke} might be various (depending on the choices the underlying signature and PKE schemes). Considering that all the proposed scheme are using pairing groups, for an intuition it is safe to assume that $t_v \approx t_{dec} \geq t_p \gg t_{me} > t_s \geq t_e$, $t_{enc} \geq 2t_e$, and $|vk| \approx |\sigma| \geq |\mathbb{G}| \geq 2k$, $|\mathbb{G}_T| \approx |C_{pke}| \geq 2|\mathbb{G}|$ (e.g., instantiating with the efficient pairing-based BLS signature [12] and Kiltz's PKE [29]).

6 Our Data Distribution System in Mobile Cloud Computing

6.1 Overview

In our system, the data owner will classify its own data into different categories, e.g., by the content/subject of the data or by the groups of potential data consumers. To protect data privacy and ensure fine-grained access control on the data, the data owner protects his data in each category with a unique type (e.g., a short description of the category) by using the TB-PRE scheme in Section 5.2. For efficiency, a secure symmetric encryption, e.g., AES, is employed to encrypt the data under a uniformly and randomly chosen secret key for each data category, and the TB-PRE scheme is then used to encrypt the symmetric secret key. Namely, all the files in each category are encrypted under a unique symmetric secret key, and a TB-PRE ciphertext of the symmetric secret key is associated with the data category.

Since each data category may contain many data files, the data owner also constructs a Merkle Hash Tree (MHT) for each data category (where the leaf nodes of the MHT are the hashes of the data files in the category), and only stores the metadata consisting of the root of the MHT corresponding to each category at his own local storage for checking data integrity and performing dynamic data operations. The use of MHT enables the data owner to achieve minimum overhead when performing dynamic data operations such as modification without affecting the integrity check of other data files. Namely, it is not necessary to download all the data files in the category when the data owner only wants to modify a single one or a small portion of them. Besides, the BLS signature scheme is used to authenticate the roots of the MHT such that the data consumer cloud also check the integrity of the data files as well as authenticate the identity of the data owner.

6.2 Details of System Operations

Let $\Pi_{pre} = (\texttt{Setup}, \texttt{KeyGen}, \texttt{ReKeyGen}, \texttt{Enc}, \texttt{ReEnc}, \texttt{Dec})$ be the TB-PRE scheme proposed in Section 5.2. Let $\Pi_{sig} = (\texttt{KeyGen}, \texttt{Sign}, \texttt{Verfify})$ be the BLS signature [12], and $\Pi_{sym} = (\texttt{E}, \texttt{D})$ be the symmetric encryption (e.g., AES). Let H be the cryptographic hash function (e.g., SHA-256 [4]) which will be used to construct MHTs in our system. We assume that each user i in the system has a BLS signing key ssk_i (stored in the local storage) corresponding to a certificated verification key vk_i (which is public to other entities) at a certificate authority (CA). Now, we describe the detail system operations of our data distribution system. For simplicity, an encryption of a data file F will be denoted by using the same letter F with a tilde on its head, i.e., $\widetilde{F}$.

System Setup. Let k be the security parameter, the system generates the system parameters $params \leftarrow$

$\Pi_{pre}.\texttt{Setup}(1^k)$, and makes it public to all users. Then, each user i in the system computes $(pk_i, sk_i) \leftarrow \Pi_{pre}.\texttt{KeyGen}(params, i)$, uploads the public key pk_i to the cloud and keeps the keys sk_i secret in his local storage. (Recall that each user i also has a pair of BLS verification and signing key (vk_i, ssk_i) by assumption.)

Data Upload. If the data owner with identity i wants to upload a set of data files $\{F_n\}$ of category t to the cloud, he distinguishes the following two cases:

- If t is a new data category, the data owner i performs the following operations.
  1) Uniformly choose a random symmetric secret key K and encrypt the data files $\{F_n\}$ by computing $\widetilde{F}_n \leftarrow \Pi_{sym}.\texttt{E}(K, F_n)$. Then, assign each encrypted data file with a unique file identifier Fid (i.e., a counter starting from 1) for later accessing the files.
  2) Construct an MHT by using the leaf node labels $\{H(\widetilde{F}_n)\}$. Let h_r be the root label of the MHT. Store the metadata, which consists of the category name t, the root label h_r, and the max file identifier Fid in the local file L.

     Metadata in file L storing in the local storage:

Category Name	Root	Max Fid

  3) Compute an encryption of the symmetric key $\widetilde{K} = \Pi_{pre}.\texttt{Enc}(pk_i, t, K)$, and an authentication $auth = \Pi_{sig}.\texttt{Sign}(ssk_i, t\|h_r)$ on the concatenation of the category name and the root label h_r.

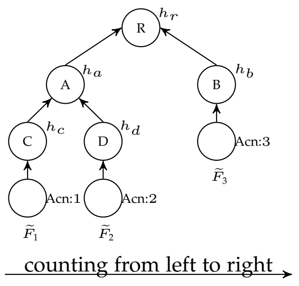

Fig. 3. An example of MHT with assigned access numbers.

  4) Upload $(\widetilde{K}, auth, \{\widetilde{F}_n\})$ to the cloud. Then, the cloud assigns each encrypted data file with an access number (Acn) by using the numbers counting the leaves of the MHT from left to right, and store the access number, the file identifier, the depth of the leaf node and the hash of each encrypted file in the MHT in the file T. Fig. 3 gives an example of an MHT with assigned Acns for three encrypted files.

     Metadata in file T storing in the cloud:

Acn	Fid	Depth	Hash

- If t is an existing data category in the cloud, the data owner i performs the following operations.
  1) Request the cloud to return the hash label h' of a leaf node at the lowest depth in the MHT, and the augmented authentication information AAI of $\widetilde{F}$. [2] For example, $h' = h_a$ and $AAI = (\text{'1'}, h_b)$ will be returned if the data owner i uploads new data files to a data category t having an MHT as in the left subfigure (a) of Fig. 4.

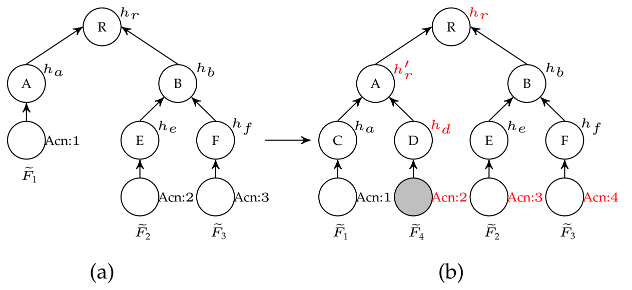

Fig. 4. Uploading a new encrypted file $\widetilde{F}_4$ to a category having an MHT as in the left subfigure (a).

  2) Retrieve the root label h_r and the max file identifier Fid corresponding to category t from the local file L, and check the correctness of h' by using h_r and AAI. If the check fails, report an error and abort.
  3) Download $\widetilde{K}$ corresponding to category t, and decrypt $K \leftarrow \Pi_{pre}.\texttt{Dec}(sk_i, \widetilde{K})$. Then, encrypt the data files $\{F_n\}$ by computing $\widetilde{F}_n \leftarrow \Pi_{sym}.\texttt{E}(K, F_n)$, and assign each encrypted data file with file identifier starting from the next integer after Fid. Finally, update the max Fid corresponding to category t in the local file L.
  4) Construct an MHT by using the leaf node labels h' and $\{H(\widetilde{F}_n)\}$. Let h'_r be the root label of the constructed MHT, compute and update the root label h_r corresponding to category t in the local file L by using h'_r and AAI. For example, we have that $h'_r = H(h_a\|h_d)$ and the new root label $h_r = H(h'_r\|h_b)$ if the data owner i uploads a new data file to a category that has an MHT as in the left subfigure (a) of Fig. 4.
  5) Compute a new authentication $auth = \Pi_{sig}.\texttt{Sign}(ssk_i, t\|h_r)$ on the concatenation of the category name and the new root label h_r.
  6) Upload $(auth, \{\widetilde{F}_n\})$ to the cloud. The cloud inserts the access number, the file identifier, and the depth of the leaf node, and the hash of each new uploaded data files in the file T, and update the access numbers of other affected records in the file T accordingly. Fig. 4 (b) gives an example of updated access numbers

2. If the cloud does not store the AAI for each encrypted data file, it can use the records in T to compute the augmented authentication information AAI.

after uploading a new data file to a category having an MHT as in the left subfigure (a).

Data Retrieval. If the data owner i wants to retrieve a set of data files of category t with file identifiers $F = \{fid_n\}$ from the cloud, he performs the following operations:

1) Request the cloud to return a set S of encrypted data files with identifiers in $F = \{fid_n\}$, and the augmented authentication information AAI for the encrypted data files in S. Then, retrieve the root label h_r corresponding to category t from the local file L, and check the integrity of those files by using h_r and AAI. If the check fails, report an error and abort. For example, if the data owner i wants to download the encrypted data files $\widetilde{F}_2$ and $\widetilde{F}_4$ in the right subfigure (b) of Fig. 4, i.e., $S = \{\widetilde{F}_2, \widetilde{F}_4\}$ and $AAI = \{(\text{‘}10\text{’}, h_f, h_r'), (\text{‘}01\text{’}, h_a, h_b)\}$,[3] the integrity check can be done by verifying whether $h_r = H(H(h_a \| H(\widetilde{F}_4)) \| H(H(\widetilde{F}_2) \| h_f))$ holds.
2) Download $\widetilde{K}$ corresponding to category t, and compute $K \leftarrow \Pi_{pre}.\texttt{Dec}(sk_i, \widetilde{K})$. Then, decrypt the files in S by using K to obtain the original data files.

Data Modification. If the data owner i wants to modify a set of data files of category t with file identifiers $F = \{fid_n\}$, he performs the following operations:

1) Request the cloud to return a set S of encrypted data files with file identifiers in $F = \{fid_n\}$, and the augmented authentication information AAI for the encrypted data files in S. Then, retrieve the root label h_r corresponding to category t from the local file L, and check the integrity of those encrypted files by using h_r and AAI. If the check fails, report an error and abort.
2) Download $\widetilde{K}$ corresponding to category t, and compute $K \leftarrow \Pi_{pre}.\texttt{Dec}(sk_i, \widetilde{K})$. Then, decrypt the files in S by using K to obtain the original data files, and perform modifications on those files to obtain a set of new data files $\{F_n\}$.
3) Encrypt the new data files $\{F_n\}$ by computing $\widetilde{F}_n \leftarrow \Pi_{sym}.\texttt{E}(K, F_n)$, and recompute the MHT root label h_r by using $\{H(\widetilde{F}_n)\}$ and AAI. Then, update the MHT root label in the local file L by using h_r.
4) Compute a new authentication $auth = \Pi_{sig}.\texttt{Sign}(ssk_i, t \| h_r)$ on the concatenation of the category name and the new root label h_r. Then, upload $(auth, \{\widetilde{F}_n\})$ to the cloud, and discard the old authentication and the encrypted data files in the cloud.

Data Deletion. If the data owner i wants to delete the whole category t in the cloud, he simply requests the cloud to discard all the data files related to category t, and then deletes the record corresponding to category t in the local file L. Otherwise, if the data owner i wants to delete a set of data files of category t with file identifiers $F = \{fid_n\}$ in the cloud, he performs the following operations:

1) Request the cloud to return a set S of encrypted data files with file identifiers in $F = \{fid_n\}$, and the augmented authentication information AAI for the encrypted data files in S. Then, retrieve the root label h_r corresponding to category t from the local file L, and check the integrity of those encrypted files by using h_r and AAI. If the check fails, report an error and abort.
2) Download $\widetilde{K}$ corresponding to category t, and compute $K \leftarrow \Pi_{pre}.\texttt{Dec}(sk_i, \widetilde{K})$. Then, decrypt the files in S by using K to obtain the original data files, and check whether those are the desired files that have to be deleted. If no, report an error and abort. Otherwise, use AAI to recompute the root label h_r of the new MHT by removing the nodes corresponding to the encrypted files in S. Finally, update the root label corresponding to category t in the local file L by using h_r.
3) Compute a new authentication $auth = \Pi_{sig}.\texttt{Sign}(ssk_i, t \| h_r)$ on the concatenation of the category name and the new root label h_r, and upload $auth$ to the cloud.

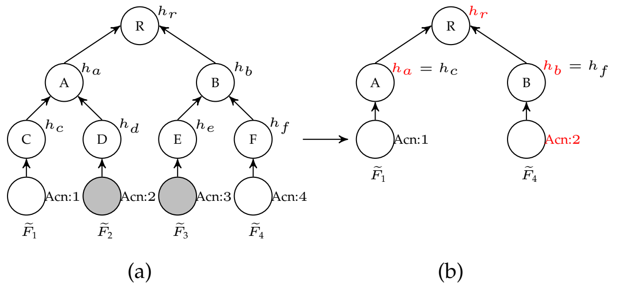

Fig. 5. Deleting the encrypted files $\widetilde{F}_2$ and $\widetilde{F}_3$ in a category having an MHT as in the left subfigure (a).

4) Request the cloud to discard the old authentication together with the encrypted data files in S, and update the access numbers and the depths for the left data files accordingly. Fig .5 gives an example that the encrypted data files $\widetilde{F}_2$ and $\widetilde{F}_3$ are deleted in a category that has an MHT as in the left subfigure (a), where the subfigure (b) shows the updated hash labels and access numbers for the left two files.

Data Permission. If a data consumer j wants to access the data of a category t belonging to the data owner i, user j first generates an access request $req = (t, pk_j, \sigma)$ by computing $\sigma = \Pi_{sig}.\texttt{Sign}(ssk_j, i \| t \| pk_j)$, where pk_j is the TB-PRE public key of user j. Then, he sends req to the data owner i. After receiving the request $req = (t, pk_j, \sigma)$, user i first checks the validity of the signature σ by computing $\Pi_{sig}.\texttt{Verify}(vk_j, i \| t \| pk_j, \sigma)$. If the check fails or t is not a valid category name, user i rejects the request. Otherwise, the data owner i performs the following operations:

1) Prepare an access permission information $per = (t, j, rk_{i \to j,t})$ by computing the re-encryption key $rk_{i \to j,t} \leftarrow \Pi_{pre}.\texttt{ReKeyGen}(sk_i, pk_j, t)$.
2) Send $per = (t, j, rk_{i \to j,t})$ to the cloud, and request the cloud to store the metadata, which consists of

3. Note that here we simply concatenate the augmented authentication information for each data block, but it is possible to compress it by using $AAI = (\text{‘}10\text{’}, h_f, \text{‘}01\text{’}, h_a)$, which saves two hash labels.

the user identity (Uid) j, and the re-encryption key $rk_{i\to j,t}$, in the permission file P for category t.

Metadata in the permission file P:

Uid	Re-encryption key

3) Compute the confirmation information $con = \sigma$ by computing $\sigma = \Pi_{sig}.\texttt{Sign}(ssk_i, t\|j\|pk_j)$, and send con to the data consumer j.

After receiving the confirmation con, user j accepts it if and only if con is a valid signature on $t\|j\|pk_j$.

Remark 1 (On the Access Permission). In practice, the data owner may directly grant the access permission of his private data to some users, e.g., his friends, without interacting with them in advance. In such a case, the data owner only has to perform the first two steps.

Data Distribution. If a permitted data consumer j wants to access a set of data files of category t with file identifiers $F = \{fid_n\}$, belonging to the data owner i, he performs the following operations:

1) Request the cloud to return the authentication $auth$ and the MHT root label h_r corresponding to category t, a set S of encrypted data files with file identifiers in $F = \{fid_n\}$, and the augmented authentication information AAI for the encrypted data files in S.
2) Check the validity of authentication $auth$ by computing $\Pi_{sig}.\texttt{Verify}(vk_i, t\|h_r, auth)$. If the check fails, report an error and abort. Then, check the integrity of those encrypted files by using h_r and AAI. If the check fails, report an error and abort.
3) Request the cloud to compute $\widetilde{K}' \leftarrow \Pi_{pre}.\texttt{ReEnc}(rk_{i\to j,t}, \widetilde{K})$ by first retrieving $\widetilde{K}$ and the re-encryption key $rk_{i\to j,t}$ (note that $rk_{i\to j,t}$ must exist for a permitted consumer j in the permission file P) corresponding to category t.
4) Download $\widetilde{K}'$, and compute $K \leftarrow \Pi_{pre}.\texttt{Dec}(sk_j, \widetilde{K}')$. Then, decrypt the files in S by using K to obtain the original data files.

Remark 2 (On the Error Reports). In the system, both the data owner and the data consumer will report errors if something is incorrect. In practice, those errors might be introduced by the unreliable wireless network. It is preferable to request a retransmission after an error occurs, and only consider it as a system failure/malicious behavior after several trails (e.g., three times).

6.3 Security Analysis

In our system, the private data in a category t of the data owner i is encrypted under a uniformly and randomly chosen symmetric secret key K by using the symmetric encryption algorithm (e.g., AES). This means that any user (including the cloud) without the knowledge of the secret key K is unable to read the private data of the data owner. In addition, different symmetric secret keys are chosen for different categories, and all the secret keys are independently encrypted under the data owner i's TB-PRE public key pk_i associated with their corresponding category names (i.e., the category name is used as the type input of the TB-PRE encryption algorithm). By the functionality and the security of the proposed TB-PRE scheme in Section 5, we have that: 1) no user without the data owner's permission can obtain the symmetric secret keys (thus cannot access any private data of the data owner); and 2) a user with the permission to read the data in the category t can obtain the corresponding symmetric secret key K, but cannot get the secret keys corresponding to other data categories (that he is not permitted to access). Namely, our data distribution system allows fine-grained data distribution in the sense that the data owner can classify his data into different categories and share them based on the data classifications to different data consumers.

Besides, the encrypted data files in a category are organized by using Merkle hash tree (MHT) associated with the cryptographic hash function H [4], i.e., the leaf nodes consists of the hashes of the encrypted data files. Since the data owner keeps the MHT root label of each category in the local file L, it is infeasible for a malicious user including the cloud to break the integrity of the encrypted data files and cheat the data owner by returning any malicious generated file that is not claimed by the data owner. In other words, our data distribution system ensures that the latest private data of the data owner is actually stored in the cloud with the integrity guarantee.

Since the data consumer does not have the MHT root label h_r corresponding to the data category t that he is allowed to access, the integrity of the encrypted data on the side of the data consumer cannot be guaranteed only by the property of the MHT if the data consumer simply obtains the root label h_r from the (malicious) cloud. In our system, the cloud is required to return h_r together with an authentication $auth$, which is a signature of the data owner on the concatenation of the category name t and h_r, to the data consumer. This makes it infeasible for a malicious cloud to output a false root label h_r that is not generated by the data owner, and is accepted by the data consumer (Otherwise, the cloud should be a valid attacker against the BLS signature). Besides, the authentication $auth$ associated with each data category also allows the data consumer to authenticate the data owner's identity. In all, we have shown that the proposed data distribution system is secure in the security model given in Section 4.2.

Remark 3 (On the Number of Interactions). By considering the limitations of mobile devices, we design our data distribution system such that no interactions between the data owner and the data consumer are needed after the data permission phase, and the data consumer is not required to store any metadata corresponding to the data category. In particular, the cloud is designed to return both the MHT root label h_r and a corresponding authentication $auth$ to the data consumer, and there is no interaction between the data owner and the data consumer after each (dynamic) data operation. This gives a possibility for a malicious cloud to cheat the data consumer by ignoring the updates and replaying an old version of the root label and authentication pair $(h_r, auth)$ (as well as the corresponding data). In the following, we give a further discussion on this issue.

- First, we clarify that this kind of attacks is meaningless to the cloud. Note that the data owner always keeps the latest MHT root label in his local file L, the replay attack cannot work for the data owner. This means that the latest encrypted data files of the data owner are actually stored in the cloud. In other words, the cloud has to pay additional computation and storage cost for the above attacks almost without gaining any benefit.
- Second, from the point of techniques, one can solve this problem by asking the data owner to perform more computation while still keep the feature of no interactions after the data permission phase. For example, the data owner can simply add a time stamp as the input of the BLS signature when generating the authentication $auth$, and update the $auth$ after a reasonable period of time (depending on applications), e.g., once per week/month. In this case, the data consumer can check the freshness of the encrypted data files by comparing the time stamp contained in the authentication $auth$ with his local system time.

Remark 4 (On the Lazy Access Revocations). In some applications, the data owner may just want to temporarily share his private data with some data consumer during a specific time period. In our system, the data owner can classify his private data by considering the permitted access deadline as an option. For example, the data owner may set a category name t: "Photos in Europe–shared before Jan 1th, 2015" for allowing the data consumer to access his private photos on a journey in Europe. After the deadline has passed, he updates the category name and encrypts the symmetric encryption K by using the TB-PRE scheme with the new category name, and grant access permission to the non-revoked data consumers. In such a case, a data consumer with permission to a previous expired category name might not be able to access the data associated with the new category name. This could achieve some kind of lazy revocation on the access permission of the data consumer.

7 Performances Analysis

In this section, we analyze the storage overhead of all users and the communication overhead of each operation, and estimate the computation overhead of our data distribution system in a proof-of-concept implementation. In Table 2, we introduce several variables for our performance analysis.

TABLE 2
Variable and Description

Variable	Description
N_t	The number of data categories of each data owner
N_c	The number of data consumers for each data category
N_f	The number of data files in each data category

Letting the system security parameter $k = 128$. To achieve this security level, we use the Barreto-Naehrig (BN) curve [8] over base field F_p-256 for both the proposed TB-PRE scheme and the BLS signature. This means that the group elements in $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ and $\mathbb{Z}_p$ can be represented in 128 bytes, 33 bytes (in the LSB compressed form), 384 bytes and 32 bytes, respectively. Since we use SHA-256 as our secure cryptographic hash function H, the length of a hash label is 32 bytes. Besides, we also assume that both the number of the data files in each data category, and the number of users in the system are at most 2^{32} (which means that the access number, the max file identifier, as well as the user identity can be represented in 4 bytes), and that the length of the category names is at most 64 bytes.

TABLE 3
Storage and computation overhead comparison between two TB-PRE schemes in Experiments (an average of 100 runs of each operation)

			Our TB-PRE	Weng et al. [54]
Storage	RK		33 bytes	66 bytes
	C_1		193 bytes	561 bytes
	C_2		416 bytes	561 bytes
Computation	Enc		4.64 ms	14.80 ms
	ReEnc		16.39 ms	16.37 ms
	Dec	C_1	18.33 ms	20.14 ms
		C_2	3.99 ms	10.61 ms

Since we use TB-PRE scheme as a building block, one can instantiate our system with any other TB-PRE schemes in [35], [45], [48], [54]. However, the comparison among related TB-PRE schemes in Table 1 indicates that our scheme has good overall performance. For a concrete comparison, we further implement the TB-PRE scheme in [54], which seems to the most competitive one in the literature. We compare the performance between the scheme in [54] and ours, with the same choice of parameters and on a Lenovo X1 Carbon laptop (equipped with a Win 10 system, 2.5 GHz Intel i7 CPU and 8GB memory). Interestingly, Table 1 shows that the re-encryption operation in [54] might be more efficient than ours, but the experiment results demonstrate that they are actually very close and comparable. This is because the computational performance is dominated by the costly pairing operations. A detail comparison is given in Table 3, which again confirms that our TB-PRE scheme has better overall performance.

7.1 Storage

In our system, both the data owner and the data consumer have to store a pair of public and secret keys for the TB-PRE scheme (about 225 bytes including an element in $\mathbb{G}_1$, an element in $\mathbb{G}_2$, and two elements in $\mathbb{Z}_p$), and a pair of verification and signing keys for the BLS signature (about 160 bytes including an element in $\mathbb{G}_1$, and an element in $\mathbb{Z}_p$). This in all needs 385 bytes storage. In addition, the data owner has to store a local file L which consists of the category names, and their corresponding MHT root labels and max file identifiers. In our settings, each entry in L needs 100 bytes, and thus the total size of L is $100N_t$ bytes for an average N_t data categories of each data owner. In all, the storage overheads of each data owner and each data consumer are $385 + 100N_t$ bytes and 385 bytes, respectively.

For each data owner, the cloud has to store the data owner's TB-PRE public key and BLS verification key (about

TABLE 4
Absolute overhead and percentage of the cloud when storing 1 GB data with average file size 100 kB.

$N_t(N_f)$ \ N_c	50	100	150	200	250
10 (1000)	0.421 MB (0.042%)	0.44 MB (0.044%)	0.458 MB (0.046%)	0.477 MB (0.047%)	0.495 MB (0.05%)
100 (100)	0.608 MB (0.061%)	0.793 MB (0.079%)	0.978 MB (0.098%)	1.163 MB (0.116%)	1.348 MB (0.135%)
1000 (10)	2.476 MB (0.248%)	4.326 MB (0.433%)	6.176 MB (0.618 %)	8.026 MB (0.803%)	9.876 MB (0.988 %)
10000 (1)	21.16 MB (2.116%)	39.66 MB (3.966%)	58.16 MB (5.816 %)	76.66 MB (7.666%)	95.16 MB (9.516 %)

289 bytes). For each data category of the data owner, the cloud has to store a TB-PRE ciphertext (about 193 bytes), an authentication $auth$ (about 33 bytes), a permission file P and a file T. By our settings, each entry in the files P and T needs about 37 and 40 bytes,[4] respectively. This means that for each data category, the files P and T have sizes $37N_c$ and $40N_f$ bytes, respectively, where N_c and N_f are the average numbers of data consumers and data files for each data category. Since there are N_t data categories for each data owner, the total storage overhead in the cloud for each data owner is $(226 + 37N_c + 40N_f)N_t + 289$ bytes.

For concreteness, considering a system with $N_t = 20$ categories for each data owner, $N_f = 100$ data files and $N_c = 50$ data consumers for each data category, the overhead of the data owner, the data consumer, and the cloud is about 2.385 kilobyte (kB), 0.385 kB, and 121.809 kB, respectively. This is about 0.0012%, 0.0002% and 0.0609% of the total storage needed for storing the N_f data files with an average file size 100 kB (i.e., about 200 megabyte (MB)).

Now, we fix that each data owner has data files in a total size of 1 gigabyte (GB), and the size of each data file is 100 kB. In other words, there are 1×10^4 data files of each data owner. In the following, we study the overhead percentage of the cloud and the data owner in our system with respect to the total data size for different choices of N_t and N_c, which are both highly related to the scalability of our data distribution system. Actually, a larger N_t means a more fine-grained access control over the total number of the data files (e.g., each category only has a single data file in an extreme case). Besides, a larger N_c for each data category indicates a more flexible data distribution that the system allows.

In Table. 4, we give the absolute and percentage overhead of the cloud for different choices of (N_t, N_c). As shown in Table 4, the overhead of the cloud in the extreme case, where each data category only has one data file but has 250 data consumers (i.e., $(N_t, N_c) = (10^4, 250)$), is about 9.516% of the total data size. This seems relatively large compared to that for the setting $(N_t, N_c) = (10^4, 50)$. However, for such a choice there are 2.5×10^6 data access permissions for the 1 GB data, and the average overhead for each data access permission is low (i.e., only about 38 bytes for each access permission to a 100 kB data file). Besides, the number of data consumers for each data category is usually inversely proportional to the number N_t of the data categories. This is because for a fixed number of total data files, the access control becomes strict as N_t increases, and the number of data consumers for each data category is expected to reduce. Thus, the figures at the left bottom corner (i.e., 5×10^5 data access permissions for the 1 GB data) may occur more often than that at the right bottom corner (i.e., 2.5×10^6 data access permissions for the 1 GB data).

4. Note that the access numbers do not have to be stored because they are simply sequential integer numbers from 1.

TABLE 5
Absolute overhead and percentage of the data owner when storing 1 GB data with average file size 100 kB.

$N_t(N_f)$ \ N_c	50	250
100 (100)	10.385 kB (0.001 %)	10.385 kB (0.001 %)
1000 (10)	100.385 kB (0.01 %)	100.385 kB (0.01 %)
10000 (1)	1000.385 kB (0.1 %)	1000.385 kB (0.1 %)

Table. 5 shows the absolute storage overhead of the data owner for different choices of (N_t, N_c). For example, the figures at the left top corner give the absolute overhead and percentage of the data owner when each category contains 100 data files and has 50 data consumers, while the figures at the right bottom corner give the absolute overhead and percentage of the data owner when each category only contains one data file (i.e., achieving the most fine-grained access control over the data files) and has 250 data consumers. The results in Table 5 indicates that the overhead of the data owner is very low and independent of the number N_c of the data consumers.

7.2 Communication

In this subsection, we show the communication overhead for the system operations of our data distribution system. Without loss of generality, we assume that the transmission of the affected data files is necessary for all data distribution systems, and all other transmissions are considered as communication overheads introduced by specific data distribution system.

In order to upload N' new data files to an existing data category in the cloud, the data owner should first get the hash label corresponding to the leaf node at the lowest depth and its corresponding augmented authentication information (about $32 \log N_f + 37$ bytes for a category with N_f data files).[5] In addition, the data owner has to download a TB-PRE ciphertext (about 193 bytes), and will upload a

5. In our implementation, we use one byte for specifying the length of the AAI, and four bytes for representing the description bit-string. In this setting, we can handle an MHT with less than 2^{32} leaf nodes, and the length of AAI for each leaf node is about $32 \log N_f + 5$ bytes.

new authentication $auth$ (about 33 bytes) to the cloud. Thus, the total communication overhead is about $32 \log N_f + 263$ bytes. For concreteness, in a case that the data owner only uploads one data file with size 100 kB to a data category with 1×10^3 data files in the cloud, the communication overhead is about 583 bytes, and 0.583% of the transmission of the 100 kB data file.

TABLE 6
The communication overhead (in bytes) of each operation on N' new data files to an existing data category with N_f data files (where Upload* denotes uploading data files to a new data category)

Operations	Communication overhead
Upload*	226
Upload	$32 \log N_f + 263$
Retrieval	$32N' \log N_f + 5N' + 193$
Modification	$32N' \log N_f + 5N' + 226$
Deletion	$32N' \log N_f + 5N' + 226$
Permission	< 260
Distribution	$32N' \log N_f + 5N' + 449$

By almost the same analysis as for the upload operation, the communication overheads for the upload operation to a new category, the modification, deletion, permission and data distribution operations are given in Table 6, where N' denotes the number of data files that the corresponding data operation affects. Note that the overhead of the upload is independent of the number N' of the new data files, lazy upload operations can be done in real applications to reduce average communication overhead (e.g., only upload the data files when an upload for a large number of data files is required, or an emergent upload occurs, or a specific period of time has passed).

For concreteness, we also give the percentage overhead of the operations on N' data files with average file size 100 kB (i.e., the necessary data transmission is about $N' * 100$ kB) for a category with $N_f = 1 \times 10^3$ data files in Fig. 6. As shown in the figure, the percentage overhead with respect to the necessary data transmission approaches a constant 0.33% as N' grows. This indicates that our system has a good scalability in terms of the communication overhead.

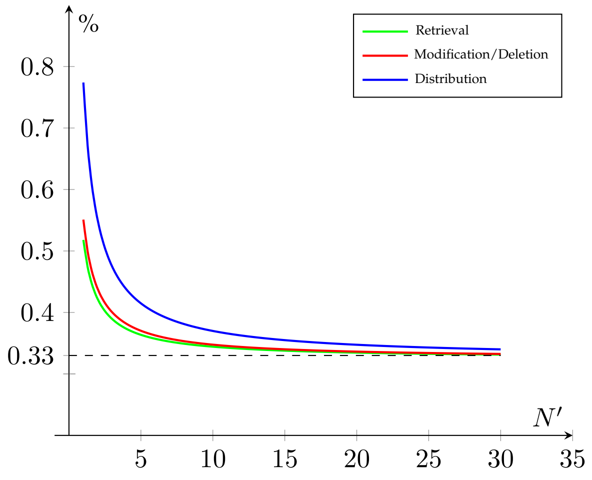

Fig. 6. Percentage overhead of each operation on N' new data files to an existing data category of $N_f = 1 \times 10^3$ data files with average file size 100 kB.

7.3 Computation

We conducted a proof-concept-implementation of our data distribution system by using the MIRACL Crypto SDK [3]. We choose Barreto-Naehrig (BN) curve [8] over base field F_p-256 for our TB-PRE scheme and the BLS signature for 128-bit security. We use AES and SHA-256, which are both implemented in the MIRACL Crypto SDK, as our symmetric encryption algorithm and hash function also for 128-bit security. The whole implementation is written in C++ and has about 3400 lines code.

To better examine the computation overhead of our data distribution system and avoid the influence of the bandwidth, we simply run the program on a single computer and simulate the storage of the data owner, the cloud and the data consumer by using three document folders. In other words, the upload and download operations are just moving files between the data owner's/consumer's folders and the cloud's folder. The computer in the simulation runs Linux Ubuntu 12.04 system, and equips a 2.83 GHz Intel Core2 CPU and 4GB memory.

In our experiment, we first let the data owner upload $N_f = 1 \times 10^3$ data files to a category in the cloud, and then perform dynamic operations such as retrieval and modification on the created data category. The computation overhead for uploading the 1×10^3 data files with average file size 100 kB to a new category is about 2842.43 ms for the data owner, and 1421.77 ms for the cloud. This means that the throughput of the upload operation (to a new category) is about 35.2 MB/s. As for each permission operation, the time for the data owner is about 0.7 ms. In other words, the data owner is capable of issuing as many as 1429 data access permissions in a second.

In Table 7, we give the average overhead for other operations on N' new data files (also, with average file size 100 kB) to the created data category. The timing is the average of 10 runs of the program. As shown in the table, the throughput of all the operations increases as N' grows. For example, the throughput for the deletion and the distribution changes from about 3.8 MB/s for $N' = 1$ to about 32 MB/s for $N' = 80$. Since the modification operation has to first retrieve the data files from the cloud, and then upload the modified data files to the cloud. The timing is relatively large, but the throughput of the modification still has about 16.4 MB/s for $N' = 80$. Again, the figures in Table 7 indicate that lazy operations can be done to reduce the average computation overhead. In all, the experiment results given above show that our data distribution system cloud be practical for mobile users in cloud computing.

8 Conclusion

We propose a practical data distribution system in mobile cloud computing, which does not involve any trusted third party and provides several useful properties including data privacy, data integrity, data authentication, dynamic data modifications and deletions, as well as fine-grained access control. Our system leverages a new efficient and provably secure type-based proxy re-encryption scheme, Merkle hash

TABLE 7
The average computation overhead (in million second, ms.) of each operation on N' new data files to an existing data category of $N_f = 1 \times 10^3$ data files with average file size 100 kB (where $t_{\text{cl}}, t_{\text{ow}}, t_{\text{co}}$ denote the computation overhead of the cloud, the data owner and the data consumer, respectively)

Operation \ N'	1	20	40	60	80
Upload (t_{cl}, t_{ow})	(3.35 ms, 26.44 ms)	(30.25 ms, 81.66 ms)	(58.25 ms, 139.71 ms)	(86.14 ms, 196.44 ms)	(144.54 ms, 254.69 ms)
Retrieval (t_{cl}, t_{ow})	(1.89 ms, 25.29 ms)	(2.27 ms, 82.35 ms)	(2.7 ms, 138.15 ms)	(2.93 ms, 193.84 ms)	(3.66 ms, 252.21 ms)
Modification (t_{cl}, t_{ow})	(1.89 ms, 29.1 ms)	(2.27 ms, 138.3 ms)	(2.7 ms, 255.26 ms)	(2.93 ms, 368.63 ms)	(3.66 ms, 488.92 ms)
Deletion (t_{cl}, t_{ow})	(6.45 ms, 26.32 ms)	(16.67 ms, 80.44 ms)	(19.22 ms, 138.73 ms)	(16.52 ms, 193.79 ms)	(21.41 ms, 251.86 ms)
Distribution (t_{cl}, t_{co})	(23.64 ms, 22.72 ms)	(23.66 ms, 76.75 ms)	(24.25 ms, 134.18 ms)	(24.35 ms, 190.38 ms)	(25.21 ms, 249.93 ms)

tree, as well as the BLS signature to ensure the security. An extensive performance analysis and a proof-of-concept implementation show that our data distribution is practical.

Acknowledgments

This work is sponsored by the National Basic Research Program of China under Grant No. 2013CB338003, and the National Natural Science Foundation of China under Grant No. 61170278, 61602046, 91118006.

References

[1] Amazon S3, http://aws.amazon.com/s3/.

[2] Cisco visual networking index: Global mobile data traffic forecast update, 2013–2018, February 5, 2014. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html.

[3] MIRACL Crypto SDK, https://certivox.com/.

[4] Secure Hash Standard (SHS), National Institute of Standards and Technology (NIST), FIPS PUB 180-4, http://csrc.nist.gov/publications/PubsFIPS.html.

[5] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 598–609, New York, NY, USA, 2007. ACM.

[6] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9:1–30, February 2006.

[7] Lakshmi N. Bairavasundaram, Garth R. Goodson, Shankar Pasupathy, and Jiri Schindler. An analysis of latent sector errors in disk drives. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '07, pages 289–300, New York, NY, USA, 2007. ACM.

[8] PauloS.L.M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer Berlin Heidelberg, 2006.

[9] A. Barsoum and A. Hasan. Enabling dynamic data and indirect mutual trust for cloud computing storage systems. *Parallel and Distributed Systems, IEEE Transactions on*, 24(12):2375–2385, Dec 2013.

[10] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *Advances in Cryptology - EUROCRYPT'98*, volume 1403 of *LNCS*, pages 127–144. Springer Berlin / Heidelberg, 1998.

[11] Dan Boneh and Xavier Boyen. Efficient selective identity-based encryption without random oracles. *Journal of Cryptology (JoC)*, 24(4):659–693, 2011. early version in Eurocrypt 2004.

[12] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer Berlin Heidelberg, 2001.

[13] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 185–194, New York, NY, USA, 2007. ACM.

[14] Ee-Chien Chang and Jia Xu. Remote integrity check with dishonest storage server. In Sushil Jajodia and Javier Lopez, editors, *Computer Security - ESORICS 2008*, volume 5283 of *Lecture Notes in Computer Science*, pages 223–237. Springer Berlin Heidelberg, 2008.

[15] Melissa Chase and Sherman S.M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 121–130, New York, NY, USA, 2009. ACM.

[16] Cheng-Kang Chu and Wen-Guey Tzeng. Identity-based proxy re-encryption without random oracles. In Juan Garay, Arjen Lenstra, Masahiro Mambo, and Ren Peralta, editors, *Information Security*, volume 4779 of *LNCS*, pages 189–202. Springer Berlin / Heidelberg, 2007.

[17] Adrian Covert. "Google Drive, iCloud, Dropbox and more compared: Whats the best cloud option?", 2012. http://gizmodo.com/5904739.

[18] Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 213–222, New York, NY, USA, 2009. ACM.

[19] Amos Fiat and Moni Naor. Broadcast encryption. In DouglasR. Stinson, editor, *Advances in Cryptology ł CRYPTO 93*, volume 773 of *Lecture Notes in Computer Science*, pages 480–491. Springer Berlin Heidelberg, 1994.

[20] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 89–98, New York, NY, USA, 2006. ACM.

[21] Matthew Green and Giuseppe Ateniese. Identity-based proxy re-encryption. In Jonathan Katz and Moti Yung, editors, *Applied Cryptography and Network Security - ACNS 2007*, volume 4521 of *LNCS*, pages 288–306. Springer Berlin / Heidelberg, 2007.

[22] Dijiang Huang, Tianyi Xing, and Huijun Wu. Mobile cloud computing service models: a user-centric approach. *Network, IEEE*, 27(5):6–11, September 2013.

[23] Nguyen Thanh Hung, Do Hoang Giang, Ng Wee Keong, and Huafei Zhu. Cloud-enabled data sharing model. In *Intelligence and Security Informatics (ISI), 2012 IEEE International Conference on*, pages 1–6, 2012.

[24] Junbeom Hur and Dong Kun Noh. Attribute-based access control with efficient revocation in data outsourcing systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1214–1221, 2011.

[25] F. R. Institute. Personal data in the cloud: A global survey of consumer attitudes [online], 2010. Available:http://www.fujitsu.com/downloads/SOL/fai/reports/fujitsu_personal-data-in-the-cloud.pdf.

[26] Ari Juels and Burton S. Kaliski, Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 584–597, New York, NY, USA, 2007. ACM.

[27] Ryan Kalember. "celebrity photo hack: Ten takeaways for enterprises", September 11, 2014. https://www.watchdox.com/en/blog/celebrity-photo-hack-ten-takeaways-enterprises/.

[28] Abdul Nasir Khan, M.L. Mat Kiah, Samee U. Khan, and Sajjad A. Madani. Towards secure mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(5):1278 – 1299, 2013. Special section: Hybrid Cloud Computing.

[29] Eike Kiltz. Chosen-ciphertext secure key-encapsulation based on gap hashed diffie-hellman. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography – PKC 2007*, volume 4450 of *Lecture Notes in Computer Science*, pages 282–297. Springer Berlin Heidelberg, 2007.

[30] Jason Kincaid. "Google privacy blunder shares your docs without permission", 2009. http://techcrunch.com/2009/03/07/huge-google-privacy-blunder-shares-your-docs-without-permission.

[31] KMPG. From hype to future: Kpmgs 2010 cloud computing survey [online], 2010. Available:http://www.kpmg.com/ES/es/ActualidadyNovedades/ArticulosyPublicaciones/Documents/2010-Cloud-Computing-Survey.pdf.

[32] Noam Kogan, Yuval Shavitt, and Avishai Wool. A practical revocation scheme for broadcast encryption using smartcards. *ACM Trans. Inf. Syst. Secur.*, 9(3):325–351, August 2006.

[33] Karthik Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, 2010.

[34] Ming Li, Shucheng Yu, Yao Zheng, Kui Ren, and Wenjing Lou. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. *Parallel and Distributed Systems, IEEE Transactions on*, 24(1):131–143, Jan 2013.

[35] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. *Information Theory, IEEE Transactions on*, 57(3):1786 –1802, march 2011.

[36] Qin Liu, Guojun Wang, and Jie Wu. Clock-based proxy re-encryption scheme in unreliable clouds. In *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pages 304–305, 2012.

[37] Xuefeng Liu, Yuqing Zhang, Boyang Wang, and Jingbo Yan. Mona: Secure multi-owner data sharing for dynamic groups in the cloud. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1182–1191, 2013.

[38] Robert McMillan. "Hacker: I broke into twitter", 2009. http://www.pcworld.com/article/164182.

[39] Ralph C. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, Los Alamitos, CA, USA, 1980.

[40] M. Nabeel, Ning Shang, and E. Bertino. Privacy preserving policy-based content sharing in public clouds. *Knowledge and Data Engineering, IEEE Transactions on*, 25(11):2602–2614, Nov 2013.

[41] Shivaramakrishnan Narayan, Martin Gagné, and Reihaneh Safavi-Naini. Privacy preserving ehr system using attribute-based infrastructure. In *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop*, CCSW '10, pages 47–52, New York, NY, USA, 2010. ACM.

[42] Alina Oprea, Michael K. Reiter, and Ke Yang. Space-efficient block storage integrity. In *In Proc. of NDSS 05*, 2005.

[43] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 557–557. Springer Berlin / Heidelberg, 2005.

[44] Bianca Schroeder and Garth A Gibson. Disk failures in the real world: What does an MTTF of 1, 000, 000 hours mean to you? In *FAST*, volume 7, pages 1–16, 2007.

[45] Jae Woo Seo, Dae Hyun Yum, and Pil Joong Lee. Proxy-invisible cca-secure type-based proxy re-encryption without random oracles. *Theoretical Computer Science*, (0):–, 2012.

[46] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 90–107. Springer Berlin Heidelberg, 2008.

[47] Mehul A Shah, Mary Baker, Jeffrey C Mogul, and Ram Swaminathan. Auditing to keep online storage services honest. In *HotOS*, 2007.

[48] Qiang Tang. Type-based proxy re-encryption and its construction. In DipanwitaRoy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *Progress in Cryptology - INDOCRYPT 2008*, volume 5365 of *Lecture Notes in Computer Science*, pages 130–144. Springer Berlin Heidelberg, 2008.

[49] Yang Tang, P.P.C. Lee, J.C.S. Lui, and R. Perlman. Secure overlay cloud storage with access control and assured deletion. *Dependable and Secure Computing, IEEE Transactions on*, 9(6):903–916, Nov 2012.

[50] D.H. Tran, Hai-Long Nguyen, Wei Zhao, and Wee Keong Ng. Towards security in sharing data on cloud-based social networks. In *Information, Communications and Signal Processing (ICICS) 2011 8th International Conference on*, pages 1–5, 2011.

[51] Cong Wang, Qian Wang, Kui Ren, Ning Cao, and Wenjing Lou. Toward secure and dependable storage services in cloud computing. *Services Computing, IEEE Transactions on*, 5(2):220–232, April 2012.

[52] Huaqun Wang. Proxy provable data possession in public clouds. *Services Computing, IEEE Transactions on*, 6(4):551–559, Oct 2013.

[53] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou, and Jin Li. Enabling public auditability and data dynamics for storage security in cloud computing. *Parallel and Distributed Systems, IEEE Transactions on*, 22(5):847–859, May 2011.

[54] Jian Weng, Yanjiang Yang, Qiang Tang, Robert Deng, and Feng Bao. Efficient conditional proxy re-encryption with chosen-ciphertext security. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Ardagna, editors, *Information Security*, volume 5735 of *LNCS*, pages 151–166. Springer Berlin / Heidelberg, 2009.

[55] Kan Yang and Xiaohua Jia. An efficient and secure dynamic auditing protocol for data storage in cloud computing. *Parallel and Distributed Systems, IEEE Transactions on*, 24(9):1717–1726, Sept 2013.

[56] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9, March 2010.

[57] Yan Zhu, Hongxin Hu, Gail-Joon Ahn, and Mengyang Yu. Cooperative provable data possession for integrity verification in multicloud storage. *Parallel and Distributed Systems, IEEE Transactions on*, 23(12):2231–2244, Dec 2012.

PLACE PHOTO HERE

Jiang Zhang received the BS degree from Sun Yat-sen University in 2009 and the PhD degree from Institute of Software, Chinese Academy of Sciences in 2015. He is now an assistant researcher in State Key Laboratory of Cryptology. His research focuses on trusted computing, applied cryptography, data security and privacy in cloud computing.

PLACE PHOTO HERE

Zhenfeng Zhang received the PhD degree from Academy of mathematics and Systems Science, Chinese Academy of Sciences in 2001. He is currently a professor and PhD supervisor in Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences. His research interests include trusted computing, applied cryptography and information security.

PLACE PHOTO HERE

Hui Guo received the BS degree from Ji Lin University in 2010 and the PhD degree from Institute of Software, Chinese Academy of Sciences in 2016. He is now an assistant researcher in State Key Laboratory of Cryptology. Her research interests include cryptography and information security, proxy re-encryption and obfuscation.