Computers & Security 114 (2022) 102602

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Secure deduplication schemes for content delivery in mobile edge computing

Yu Lin*, Yunlong Mao, Yuan Zhang, Sheng Zhong

Department of Computer Science and Technology, Nanjing University, Jiangsu, Nanjing 210023, China

ARTICLE INFO

Article history:
Received 21 June 2021
Revised 9 November 2021
Accepted 2 January 2022
Available online 4 January 2022

Keywords:
Data security
Edge computing
Secure deduplication
Data sharing
Content delivery

ABSTRACT

Since the emergence of the mobile edge computing (MEC) paradigm, data leakage has become a serious threat against edge computing users, thwarting the further applications of MEC. Previous studies concentrating on data storage security and deduplication for conventional cloud computing paradigm cannot be simply adapted to edge computing because a central coordinator (for example, a cloud server) with a global view is not always available in MEC. To tackle this problem, we take the particular properties of MEC into consideration and propose secure data deduplication schemes for three MEC settings (i.e., centralized, semi-distributed and distributed settings). All of our schemes can provide secure data storage, retrieval, sharing, and deduplication. Through theoretical analysis, we prove the security of our schemes against typical attacks in outsourced data storage. Experimental results with a real-world deployment environment have showed that our schemes can guarantee service quality of MEC effectively.

1. Introduction

Mobile edge computing (MEC) is emerging as a promising solution to the resource limitation problem for mobile devices, such as smart phones and many other IoT devices. Compared with mobile cloud computing (MCC), MEC has a much lower delay and supports both centralized and decentralized system management modes (Mao et al., 2017). Among MEC applications, some recent studies such as edge blockchain and edge server aided deep learning have attracted considerable attention. For example, federated learning (FL) has been applied in MEC for mobile users to leverage the computing power of both edge servers and users (Lim et al., 2020). These MEC applications share some notable characteristics, especially for computation offloading and storage offloading. This has enabled edge devices to be able run heavy applications with resource demands beyond their limitations. Moreover, MEC can provide parallel offloading services for multiple users simultaneously with a impalpable response delay, which is an attractive feature for parallel computing applications, such as face recognition (Mao et al., 2018), augmented reality (Ren et al., 2019) and FL (Mao et al., 2020).

However, there are still many open problems to be solved for offloading in MEC, namely user authentication and channel protection (Xiao et al., 2019), among which, data security is one of the most urgent problems. In a typical MEC application, security of data storage in edge servers is important since the majority of network data is related to end users' privacy (Cis, 2017), which will be cached in edge servers. Since users do not want to put their privacy at risk, two problems crop up: which server to store data on and how to store data. The first problem can be solved by introducing a trust evaluation scheme into MEC to rate edge servers (He et al., 2018; Xu et al., 2020; Xu et al., 2018). The second problem can be solved by using secure encryption scheme since social trust evaluation cannot protect privacy against inside adversary. But it is impractical to apply distributed encryption schemes, like threshold encryption (Agrawal et al., 2018; Jarecki et al., 2019), directly in MEC to solve the data storage security requirements because we should not only consider the security assumptions in MEC, but also ensure that both legal data users and sharers can retrieve and decrypt data correctly wherever they will be in the future. For example, distributed encryption schemes can be represented by two algorithms: $Enc(\{sk_i\}, D, r) \to C$ for encryption and $Dec(\{sk_i\}, C) \to D$ for decryption, where D is the input data, r is a random value for semantic security and $\{sk_i\}$ are the secret keys for distributed servers. On the one hand, if r is randomly chosen, the ciphertext of D will also be random for the same input data D, which makes deduplicating data directly on the ciphertext hard. On the other hand, if r is set as a constant value, the distributed servers can perform offline brute-force attack (see Section 3.2) to guess the input data. Besides, MEC applications cannot tolerate the high overhead introduced by heavy cryptographic tools. Thus, se-

* Corresponding authors.
E-mail addresses: mg1933042@smail.nju.edu.cn (Y. Lin), njucsmyl@163.com (Y. Mao), zhangyuan@nju.edu.cn (Y. Zhang), sheng.zhong@gmail.com (S. Zhong).

https://doi.org/10.1016/j.cose.2022.102602
0167-4048/© 2022 Elsevier Ltd. All rights reserved.

curity issues and efficiency of data storage should be taken into consideration in MEC at the same time.

To tackle the data security problem in content delivery for MEC applications, we propose three schemes to meet the demands of different MEC environments, ensuring the security of data storage, sharing and deduplication. 1) A centralized scheme is designed when there exists a central cloud server available for all users to generate keys that are kept secret to the cloud server. When there is no central server, we introduce two improved schemes to solve key generation problem only with the help of edge servers. 2) One semi-distributed scheme is suitable for the situation whereby an edge server set exists to cover the whole network. Covering means that each user is always able to access at least one threshold number of edge server for the key generation. 3) One distributed scheme is designed to apply when no centralized service is provided for users to guarantee the valid key generation for secure deduplication.

Compared with other deduplication schemes, ours provide a data storage service with improved QoS and more comprehensive security guarantees for MEC applications. Meanwhile, we take typical features of MEC into account which are not commonly considered in cloud scenarios:

- **Decentralized Management** The management of MEC tends to be decentralized to solve communication and computation bottleneck of the central server in MCC. Although decentralized management can alleviate these problems, it also introduces various security problems.
- **Restricted Capability of MEC Servers** Unlike the central server in MCC, which usually has unlimited computing and communication capabilities, edge servers in MEC are restricted by the network environment and hardware performance. Therefore, we need to consider reducing the computational and communication overhead as much as possible while solving data security issues in MEC.
- **Service Boundary and User Mobility** Due to the mobility of users, after uploading data to one edge server, users may request their data on another server or share their data to other users in other network locations. So when users download their data, they should be able to find the ciphertext and generate the correct decryption key, regardless of their position.

Compared with convergent encryption (CE) or message-locked encryption (MLE) (Bellare et al., 2013; Douceur et al., 2002), our schemes can resist offline brute-force attacks. We introduce distributed hash table (DHT) (Geambasu et al., 2009), distributed pseudo-random function (DPRF) (Naor et al., 1999a), and oblivious pseudo-random function (OPRF) (Freedman et al., 2005) into our schemes, so users who upload the same data in different locations can generate the same key to encrypt the data. Briefly, this paper makes the following contributions:

- We introduce more comprehensive security requirements for users' data storage and delivery in MEC applications while taking key features of MEC into account.
- We propose three schemes for different application environments (including centralized, semi-distributed and completely distributed) ensuring the security of data storage, sharing and deduplication.
- We prove that our schemes are secure against typical attacks in outsourced data storage through theoretical analysis, and experimental results show that our schemes can guarantee service quality of MEC effectively.

2. Preliminary and related work

2.1. Preliminary

2.1.1. Mobile edge computing

Mobile edge computing (MEC) is a computing paradigm that allocates nearby edge servers for mobile users to provide computing and storage services, such as CONCERT (Liu et al., 2014) and Follow Me Cloud (Taleb et al., 2019). Benefiting from the low propagation delays of MEC, users can offload computation-intensive tasks and abundant data storage to edge servers. In particular, users can upload their data blocks to nearby edge servers with small delays instead of routing data blocks through the core network to a remote central server (Beck et al., 2014). Specifically, user $\mathcal{U}$ communicates with a nearby edge server $E_{\mathcal{N}}$ to upload data D instead of requesting to a cloud server. Cloud server commonly acts as a central manager and a long-term data warehouse. For the users, they interact with $E_{\mathcal{N}}$ directly, and the cloud server becomes transparent if a completely decentralized scheme is adapted. Due to the mobility of $\mathcal{U}$, the nearby edge server for a user may change from $E_{\mathcal{N}}$ to $E_{\mathcal{N}'}$, which means that $\mathcal{U}$ may encrypt and upload D to $E_{\mathcal{N}}$ while requesting it from $E_{\mathcal{N}'}$. The edge cluster should update the data routing between different edge servers and let $E_{\mathcal{N}'}$ send data to the user.

2.1.2. Distributed hash table

Distributed hash table (DHT) (Geambasu et al., 2009) is a way to store data in a peer-to-peer network instead of being kept locally, offering fault tolerance and reliability. In a peer-to-peer network of n nodes, each node takes $1/n$ hash space. In general, DHT can be abstracted as:

- $DHT.Retrieve(L) \to \mathcal{E}$: Taking an access key L as an input, the algorithm outputs an index set $\mathcal{E} = \{I_i\}_n$, which can be used to find the corresponding nodes in the network.

When a user with an access key L wants to store data through DHT, he or she can run the function above and get the index set of target nodes $\mathcal{E} \leftarrow DHT.Retrieve(L)$. After transmitting data to these nodes, the user can keep only the access key instead of the whole index set for the further retrieval.

2.1.3. Distributed pseudo-random function

A distributed pseudo-random function (DPRF) (Naor et al., 1999b) is a function distributed over n authorized parties to evaluate a pseudo-random function f for a given input. For the same input x, each authorized party with a unique secret key generates a pseudo-random share and returns it to the user. The pseudo-random value $f(x)$ can be computed only if the user gets at least t (the setting threshold) shares from the authorized parties. For an authorized party set S of size n, the DPRF consists of the following algorithms:

- $DPRF.Setup(\lambda, n, t) \to (\{sk_i\}_n, pp)$: The algorithm takes a security parameter λ, a party number n and a threshold t as inputs to generate n secret key $\{sk_i\}_n$ and public parameters.
- $DPRF.Eval(sk_i, x, pp) \to z_i$: The evaluation algorithm is executed by an authorized party i with the secret key sk_i to compute a share for the value given by user.
- $DPRF.Comb(\{i, z_i\}_{i \in S}, pp) \to z/\perp$: The algorithm combines the shares $\{z_i\}_{i \in S}$ from the servers in S to generate the final pseudo-random value. If the algorithm is successful, output z, otherwise output $\perp$.

2.1.4. Oblivious pseudo-random function

An Oblivious Pseudo-Random Function (OPRF) (Freedman et al., 2005) allows any sender to perform secure PRF calculation for the receiver using his own key. The receiver evaluates the pseudo-random value corresponding to its private input, while the sender cannot obtain any knowledge of the input. The OPRF is constructed with followings:

1. *Components*: A group $\mathbb{G}$ of order p, two hash functions H with range $\{0,1\}^{\lambda}$ and H' with range $\mathbb{G}$.
2. *Input*: The receiver inputs a value x and the sender inputs a key k.
3. *Output*: The receiver obtains $F_k(x) = H(x, (H'(x))^k)$ and the sender get nothing.

2.1.5. Shamir secret sharing scheme

A $(n,t)-$secret sharing scheme is used to share a secret value such that: 1) There are n shares of the value and 2) Only at least t shares can recover the value. The secret sharing scheme contains the following algorithms:

- $Share(r) \rightarrow \{s_i\}_n$: The algorithm takes a secret value r as input and generates n secret shares $\{s_i\}_n$.
- $Recover(\{s_i\}) \rightarrow r/\perp$: Input a set of secret shares, the algorithm outputs the secret value r if recovery is successful. Otherwise, output $\perp$.

2.1.6. Bilinear pairings

Let $\mathbb{G}$ and $\mathbb{G}_T$ be cyclic multiplicative groups of prime order p, and g be the generator of $\mathbb{G}$. A bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a map satisfying the following properties:

1. Bilinearity: For all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degeneracy: $e(g, g)$ is a generator of $\mathbb{G}_T$, which means $e(g, g) \neq 1$.
3. Computable: There exists an efficient algorithm to compute $e(g, g)$.

$\mathcal{G}$ is the bilinear pairing generating functions defined as: $\mathcal{G}(\lambda) \rightarrow < \mathbb{G}, \mathbb{G}_T, g, p, e >$.

2.2. Related work

2.2.1. Offload strategy

Since MEC servers have limited computing resources, it is critical to schedule the computing resources of the servers efficiently. For example, edge container migration provides users with a data migration service without perception when users move away from the nearby edge servers. To effectively support edge offloading service, docker containers can be used to manage the data migration among the servers in edge computing network without caching redundant data (Ma et al., 2018). Since the placement of containers has a great impact on communication cost, Lv et al. proposed an efficient Communication Aware Worst Fit Decreasing algorithm to balance resource utilization by container migration (Lv et al., 2019). Similarly, when all mobile users offload their tasks on only one edge server, some tasks may not be completed in time. Dai et al. formulate the above joint load balancing and offloading problem as an optimization problem to maximize the utility of each edge server and proposed an effective joint algorithm to solve it (Dai et al., 2018).

2.2.2. Secure deduplication

Data deduplication is a critical technique for secure storage systems to eliminate data redundancy. Several deduplication schemes have been proposed for cloud computing and edge computing scenarios. We compare recent studies on data deduplication in Table 1. Deduplication strategies can be divided into two types: *ciphertext/tag collision check* and *equality test*. The key idea of the first type is to always generate the same ciphertext or tag for one input data. For example, convergent encryption is the most commonly used encryption scheme to support data deduplication by generating an encryption key from the original plaintext data (Douceur et al., 2002). However, CE is vulnerable to the offline brute-force attack. Knowing a candidate data set $\mathbb{D}$, the adversary can recover the data with time complexity of $O(|\mathbb{D}|)$. Equality test is allowing servers to check the existence of replicated data by interacting with data owners. Jiang et al. proposed an efficient equality test scheme with randomized tag generation to avoid brute-force attack based on a tree model (Jiang et al., 2017).

Table 1
Secure data storage schemes.

Scheme	Scenario	Global deduplication	Strategy*
Douceur et al. (2002)	Cloud/Edge	√	Type I
Jiang et al. (2017)	Cloud	√	Type II
Liu et al. (2015)	Cloud	√	Type I
Ni et al. (2018)	Edge	×	Type I
Li et al. (2020)	Edge	×	Type I
Zhang and Chen (2021)	Edge	×	Type II

*Type I: ciphertext/tag collision check, Type II: equality test.

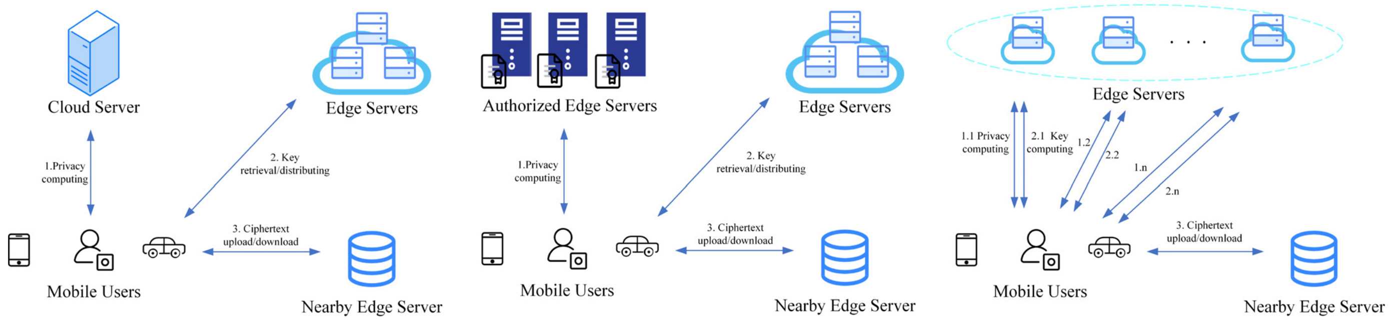
Several cloud-based schemes have introduced novel dedcuplication protocols in consideration of system models. Dekey, proposed by Li et al., achieves secure hash key management with multiple trusted servers by using the ramp secret sharing scheme (RSSS) to realize a deterministic value of secret sharing (Li et al., 2015). ClearBox introduces a third party named gateway as logically centralized entity to attest the deduplication patterns and verify the identity of users for secure data upload and deduplication (Armknecht, 2015). These schemes with additional trusted or semi-trusted servers may not be suitable for practical uses. Liu et al. proposed the first secure data deduplication scheme without the aid of additional servers (Liu et al., 2015). However, their scheme is limited to a centralized cloud storage scenario since it requires previous data users to share the encryption key for the subsequent users.

Recently, edge computing based schemes have extended the deduplication strategies. But, they do not comprehensively consider extra security risks and requirements of deduplication in edge computing scenario, which we will discuss in Section 3.2 and 3.3. Global deduplication means that each replicated data can always be deduplicated regardless of where it was uploaded. Global deduplication can be achieved in a cloud computing system naturally since a central server exists. When each mobile user only interacts with a local edge server, it becomes challenging to recognize replicated data across the edge network. In Fo-SDD scheme (Ni et al., 2018), each edge server provides a key generation service for users within its converage area, and the server can recognize duplicated data by comparing tags. Li et al. proposed an edge-assisted scheme to generate some global consistent keys for data encryption (Li et al., 2020). SHE scheme in Zhang and Chen (2021) offloads equality test phase to edge servers and introduces data similarity evaluation. However, none of these works consider both the distributed network environment and global deduplication in edge computing. Their edge computing based schemes still consider a centralized model, where a cloud server is capable of providing computing services for all users with the help of edge servers. If there is not a global accessed server, these schemes cannot guarantee global deduplication.

3. Problem statement

3.1. System overview

We adopt a typical hierarchical architecture of MEC as shown in Fig. 1 which consists of three entities: mobile user, edge server (ES) and cloud server (CS).



(a) Centralized model (b) Semi-distributed model (c) Distributed model

Fig. 1. System models.

- A **mobile user** $\mathcal{U}$ with limited storage space wants to upload his data to a nearby edge server, and retrieve his data from a nearby edge server (may be different from the server in uploading phase).
- The **Edge Servers (ES)** are distributed in the network to provide temporary data storage and caching service for mobile users. There exist three types of edge servers: 1) Authorized edge servers, 2) indexed edge servers and 3) nearby edge servers. The detailed responsibilities of the authorized and indexed edge servers will be described in Section 4 and 5. And for a nearby edge server, it caches data locally and uploads data to the CS in the data upload phase. In the data download phase, if the nearby edge server has the requested data, it responds with the data directly. Otherwise, it forwards the request to the CS or other edge servers.
- The **Cloud Server (CS)** provides data management and computing service. It controls the data storage and caching location in MEC to handle data requests from edge servers and balances their loads.

The user $\mathcal{U}$ wants to upload his data block D to the nearby edge server $E_{\mathcal{N}}$ for leveraging the computing power and storage space of edge servers. $\mathcal{U}$ may move around in the network and ask another edge server $E'_{\mathcal{N}}$ for its previously uploaded data block D. Besides, users $\mathcal{U}_1$ and $\mathcal{U}_2$ corresponding to the different nearby edge servers $E_{\mathcal{N}_1}$ and $E_{\mathcal{N}_2}$ may upload the same data block D.

Besides, we assume that there exist secure channels between users and the key managing edge servers. It is reasonable since the edge servers that provide the key storage services should be authenticated in actual scenarios (Li et al., 2015; Liu et al., 2017). Note that these secure channels can be achieved by means of a certificate or other authentication methods, thus the construction of secure channels in our schemes is omitted.

3.2. Threat model

There are two kinds of adversaries in the above MEC scenario: 1) inside adversary and 2) outside adversary. ES and CS are both curious-but-honest and regarded as potential inside adversary that have the knowledge of the content in its storage. The inside adversary is a passive adversary who follows the protocol but desires to guess the private data according to information known to it. We assume that an inside adversary can control either less than the threshold number of indexed edge servers, or any number of authorized edge servers and CS in the centralized and semi-distributed schemes. For the distributed schemes, the indexed edge servers cannot collude with each other. Specifically, we consider the inside adversary to be able to run the following attacks:

- **Offline Brute-force Attack**: A passive inside adversary takes his known information sent by users and a set of candidate data as inputs. The adversary calculates the deterministic function, such as hash, of the candidate data offline and verify whether they collide with the known information. Then the adversary can deduce whether or not the candidate data is a user's private data.
- **Integrity Attack**: An inside adversary can tamper with stored data in data flows, then it may forward the tampered data to other servers or mobile users.

The outside adversary is desired to acquire other users' private data illegally or prevent legitimate users from obtaining their data correctly. he or she may eavesdrop the channels between users and servers. The adversary can also carry out the following attacks:

- **Online Brute-force Attack**: An active outside adversary takes the information eavesdropped from public channels and a set of candidate data as inputs. The adversary impersonates a normal user to upload candidate data to ES and obtain the information returned from those ES. If the returned information collides with the eavesdropped information, the adversary can deduce that the candidate data is the private data of users.
- **Poisoning Attack**: An active outside adversary takes the known information eavesdropped from public channels as inputs. The adversary uses the known information to generate a legal data identifier and tag of a tampered data. Then he or she uploads the tampered data with the unmatched identifier to the ES. If a normal user retrieves data with the identifier, he or she will obtain a tampered data from the ES.

3.3. Design goals

By analyzing the demands of mobile users and edge servers, we conclude the following design goals:

Global Deduplication. The replicated data should be recognized and deduplicated across the edge computing network. Mobile users may upload the same data from different areas when interacting with different edge servers, and edge servers should be able to recognize ciphertext of the same data block without decryption and reduce the redundant blocks in local storage.

Secure Deduplication. The server can recognize the same encrypted data and delete the redundant ciphertext without knowing any information about the data while the ciphertext should still be available to all authorized users.

Data Confidentiality. For data confidentiality, unauthorized users and servers cannot obtain the original data or infer the data through brute-force attacks. So both the key and the deterministic function of the original data should be kept secret from both inside and outside adversaries.

Data Validity. The data validity in our schemes will be destroyed by either integrity attack or content poisoning attack. For both kinds of attacks, mobile users will obtain unexpected data.

Thus, a secure storage scheme should ensure that servers can recognize whether the attacks happen.

4. Secure data deduplication solution

In this section, we present our centralized and semi-distributed schemes to realize the secure content delivery with data deduplication. The key idea of the two schemes is to let CS/authorized edges and indexed edges provide computing and key storage services respectively for mobile users with blinded input data. And in the next section, we will discuss how to generate and store data keys without the help(s) of the CS/authorized edges in a distributed situation. Before introducing our schemes, we define some auxiliary functions in our protocols:

1. ***KeyGen***: The user $\mathcal{U}$ runs this algorithm to generate a random symmetric encryption key k for a data block.
2. ***Enc***: Taking a data block D and a key k as inputs, the user $\mathcal{U}$ encrypts D using a symmetric encryption scheme, such as AES, to generate ciphertext C.
3. ***Dec***: Taking a ciphertext C and a key k as inputs, the user $\mathcal{U}$ decrypts C to obtain the original data block D.
4. ***TagGen***: Taking any message as inputs, the function generates a collision-resistant tag T of the message.

4.1. A Centralized Scheme

To make a secure deduplication of the data in server storage, the main problem is identifying the same data encrypted by users. The intuitive method is to encrypt the same data with the same encryption key, while the key transmission between users without knowing the identity of the other party is difficult.

In the centralized model, CS is still accessible and able to provide computing task for all users. The indexed edge servers are indexed by DHT to provide key management service for users. $\mathcal{U}$ wants to upload his data D to the nearby edge server $E_{\mathcal{N}}$. The basic centralized protocol is shown in Fig. 2. The centralized upload protocol consists of three steps: blind tag generation, key sharing or recovery, and data upload.

1. **Blind Tag Generation**. Since it is not secure to let CS know the privacy information of encryption keys, we introduce the blind tag generation into the upload protocol. $\mathcal{U}$ involves a blind tag generation interaction with CS as shown in the steps $1-4$ of Fig. 2.
2. **Key Sharing or Recovery**. To avoid offline brute-force attack, instead of generating an encryption key directly from the data, $\mathcal{U}$ takes the blind tag bt as the access key of DHT to fetch an index list $\mathcal{E} = \{I_1, I_2, \dots\}$ as the indexes of key managers. For the initial user, $\mathcal{U}$ derives a random key k and shares it to the edge servers $\{E_{I_i}\}_{I_i \in \mathcal{E}}$ with the key identifiers $\{kid_i = bt \oplus H_G(h||I_i)\}$ and a threshold number t. For the subsequent user, $\mathcal{U}$ sends $\{kid_i\}$ to $\{E_{I_i}\}_{I_i \in \mathcal{E}}$ to obtain at least t shares and computes the key k using the shares.
3. **Data Upload**. Finally, $\mathcal{U}$ encrypts the data D with k and uploads the ciphertext C to the nearby edge server $E_{\mathcal{N}}$. Note that if $E_{\mathcal{N}}$ already stores C, it will just grant the access of C to $\mathcal{U}$ and not need to save C.

4.2. Semi-distributed scheme

In the semi-distributed scheme, there is no central server for all users to generate the same secure blind tag corresponding to the same data block, but we assume that the authorized edge servers will construct a peer-to-peer network to provide DPRF evaluation service for users. Informally, we define the semi-distributed environment as: there exists a collection containing n authorized edge servers and each user can get the services provided by at least t servers in the collection regardless of the user's location, but no server can be reached by all users. Thus we introduce DPRF into our semi-distributed scheme to generate a blind tag.

4.2.1. Secret key generation

To evaluate DPRF, each authorized server needs to hold a secret key $sk_i = \alpha_i$ as a polynomial part of $sk = \alpha$. We show that each server can generate a correct secret key sk_i in Algorithm 1 without the help of a central server and sk is still kept secret from all authorized servers.

Algorithm 1 Algorithm Secret Key Generation.

```
Input: Server number $|\mathcal{S}| = n$ and threshold $t$
Output: $E_{\mathcal{A}_i}$ outputs $sk_i = \alpha_i$ and $pk = g^{\alpha}$ is published
1: $E_{\mathcal{A}_i}$ generates a random polynomial $P_i(x) = \sum_{j=0}^{t} a_{i,j} \cdot x^{j-1}$ and evaluates $P_i(i)$
2: for $k \in (1, n)$ and $k \neq i$ do
3:     $E_{\mathcal{A}_i}$ computes $P_i(k)$ and sends it to $E_{\mathcal{A}_k}$
4:     $E_{\mathcal{A}_i}$ receives $P_k(i)$ from $E_{\mathcal{A}_k}$
5: $E_{\mathcal{A}_i}$ outputs its secret key as $sk_i = \sum_{k=1}^{n} P_k(i) = \alpha_i$ and publishes it public key $pk_i = g^{sk_i}$
6: The public key can be computed as $pk = \prod_{k=1}^{n} pk_i^{\Delta_{i,\mathcal{S}}(0)} = g^{\alpha}$
```

4.2.2. Blind tag generated with DPRF

We show how to generate the blind tag in a distributed scenarios using DPRF based on DDH assumption.

1. The public parameters are $< \mathbb{G}, \mathbb{G}_T, e, g, p >$ The authorized server $E_{\mathcal{A}_i}$ generate. the secret key $sk_i = P(i)$ by running the key generation algorithm in Algorithm 1 with other authorized servers and the public key $pk = g^{\alpha}$ can be computed by all authorized servers. $H_G(\cdot)$ is a secure collision-resistant hash function: $\{0,1\}^* \to \mathbb{G}$.
2. $\mathcal{U}$ who wants to compute blind tag of the data block D first picks $\beta \xleftarrow{\$} \mathbb{Z}_p^*$, calculates $M \leftarrow H_G(D)$ and sends $\bar{M} = M \times g^{\beta}$ to $\{E_{\mathcal{A}_i}\}_n$.
3. $\mathcal{U}$ receives the share $bt_i = \bar{M}^{sk_i}$, and generates the blind tag: $bt = \prod_{i \in \mathcal{S}} bt_i^{\Delta_{i,\mathcal{S}}(0)} \cdot pk^{-\beta}$, where $\Delta_{i,\mathcal{S}}(x) = \prod_{j \in \mathcal{S}, j \neq i} \frac{x-j}{i-j}$ is the Lagrange coefficient.
4. $\mathcal{U}$ verifies whether $e(g, bt) \overset{?}{=} e(pk, M)$.

We prove the blind tag generation with the DPRF based on DDH assumption is correct and the form of the generated blind tags is the same as those generated by a single cloud server, except that the exponent of bt is changed from the secret key of CS to the public key of DPRF.

Correctness

$$\begin{aligned}
\prod_{i \in \mathcal{S}} bt_i^{\Delta_{i,\mathcal{S}}(0)} \cdot pk^{-\beta} &= \prod_{i \in \mathcal{S}} (M \cdot g^{\beta})^{sk_i \cdot \Delta_{i,\mathcal{S}}(0)} \cdot pk^{-\beta} \\
&= (M \cdot g^{\beta})^{\alpha} \cdot pk^{-\beta} \\
&= bt
\end{aligned}$$

4.2.3. Semi-distributed protocol

As shown in Fig. 3, unlike the centralized scheme, the upload procedure of the semi-distributed scheme mainly contains the following steps:

1. In the setup phase, n authorized edge servers $\{E_{\mathcal{A}_i}\}_n$ will be selected to initialize DPRF using Algorithm 1 and system public parameters will be published.

Setup

Let $\mathbb{G}$ and $\mathbb{G}_T$ be cyclic multiplicative groups of prime order p, g be the generator of $\mathbb{G}$, and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be the bilinear mapping function. $H_G(\cdot)$ is a secure collision-resistant hash function: $\{0,1\}^* \to \mathbb{G}$. CS picks a random number $x \xleftarrow{\$} \mathbb{Z}_P^*$ as the secret key and publish $Y = g^x$ as its public key.

Upload Protocol

1. $\mathcal{U}$ first picks $\beta \xleftarrow{\$} \mathbb{Z}_p^*$, calculates $M \leftarrow H_G(D)$ and sends $\bar{M} = M \times g^\beta$ to CS before uploading data D to a nearby edge server $E_\mathcal{N}$,
2. CS computes $bt' = \bar{M}^x$ and returns $< bt', Y >$ to $\mathcal{U}$.
3. $\mathcal{U}$ calculates $bt = bt' \times Y^{-\beta} = M^x$ as the blind tag and verifies the integrity of bt by the equation: $e(g, bt) \overset{?}{=} e(Y, M)$.
4. $\mathcal{U}$ computes the hash $h \leftarrow H(D)$, uses bt as the access key of DHT to get an edge server index list $\mathcal{E} = \{I_1, I_2, ...\} \leftarrow DHT.Retrieve(bt)$, and generates the key identifiers $\{kid_i = bt \oplus H_G(h||I_i)\}$. Then $\mathcal{U}$ sends $\{kid_i\}$ to the servers corresponding to $\mathcal{E}$ to check whether D has been uploaded, or in other words, whether another user has generated a key for D before.
   - If $\mathcal{U}$ is the initial user uploading D, he or she generates a symmetric key $k \leftarrow \boldsymbol{KeyGen}$ and shares the key: $\{s_i\} \leftarrow SS.Share(k)$. Then $\mathcal{U}$ sends the shares to these servers with kid as the identifier through a secure channel: $\{E_{I_i} :< kid_i, s_i >\}_{I_i \in \mathcal{E}}$.
   - Else, $\mathcal{U}$ uses $\{kid_i\}$ to request the key shares $\{s_i\}$ from edges servers $\{E_{I_i}\}_{I_i \in \mathcal{E}}$ through a secure channel and recovers the key $k \leftarrow Recover(\{s_i\})$.
5. Finally, $\mathcal{U}$ encrypts the data $C \leftarrow \boldsymbol{Enc}(D, k)$, computes the tag $T = (T_1, T_2)$, where $T_1 \leftarrow \boldsymbol{TagGen}(C)$ and $T_2 \leftarrow \boldsymbol{TagGen}(T_1||bt||k)$, and search identifier $h' \leftarrow H(bt||k)$, and uploads $< h', C, T >$ to $E_\mathcal{N}$. Then $\mathcal{U}$ keeps $< bt, h >$ to retrieve C in the future.

Download Protocol

1. $\mathcal{U}$ with $< bt, h, h' >$ obtains a list of server indexes $\mathcal{E} \leftarrow DHT.Retrieve(bt)$, and requests the shares $\{s_i\}$ by sending $\{kid_i = bt \oplus H_G(h||I_i)\}$ to $\{E_{I_i}\}_{I_i \in \mathcal{E}}$.
2. $\mathcal{U}$ recovers the key $k \leftarrow Recover(\{s_i\})$.
3. $\mathcal{U}$ requests the ciphertext C by sending h' to $E_\mathcal{N}$. If h' is not in the storage table of $E_\mathcal{N}$, $E_\mathcal{N}$ will forward the request to other servers. Otherwise, $E_\mathcal{N}$ returns C to $\mathcal{U}$.
4. $\mathcal{U}$ decrypt the ciphertext $D \leftarrow \boldsymbol{Dec}(C, k)$.

Fig. 2. Basic Centralized Protocol.

2. $\mathcal{U}$ who wants to upload data block D to $E_\mathcal{N}$ first communicates with $\{E_{\mathcal{A}_i}\}_n$ to generate and verify a blind tag bt of D.
3. $\mathcal{U}$ uses bt as the access key of DHT to fetch a list of servers $\{E_{I_i}\}_n$. If $\mathcal{U}$ is the initial user uploading D, he or she chooses a random key k and shares it with the key identifiers $\{kid_i = bt \oplus H_G(h||I_i)\}$ to $\{E_{I_i}\}_n$. Otherwise, he or she requests the shares $\{s_i\}$ from $\{E_{I_i}\}_n$ by sending $\{kid_i\}$ and recovers the key $k \leftarrow Recover(\{s_i\})$.
4. Finally, the user encrypts D with the key k to get ciphertext C, generates tag T and search identifier h' of C, then uploads the ciphertext $< h', C, T >$ to $E_\mathcal{N}$.

The download protocol is similar to the centralized scheme because legal users still keep $< bt, h, h' >$ for ciphertext downloading and decryption.

Note that different users with the same data always upload the same ciphertext no matter which neighbor edge server they upload the data to,which makes the edge servers and cloud server more efficient in data deduplication. For example, the users $\mathcal{U}_1$ and $\mathcal{U}_2$ in the Fig. 4 who want to upload the same data D cannot communicate with each other since they are in different network locations. Based on the protocol we designed, the two users can agree on the same encryption key without additional third party trusted servers

Setup Let $\mathbb{G}$ and $\mathbb{G}_T$ be cyclic multiplicative groups of prime order p, g be the generator of $\mathbb{G}$, and $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be the bilinear mapping function. $H_G(\cdot)$ is a secure collision-resistant hash function: $\{0,1\}^* \rightarrow \mathbb{G}$. DPRF is initialized by running Algorithm 1 to evaluate the pseudo-random function $f_\alpha(x) = x^\alpha$ and generate secret keys $\{sk_i\}_n$ for n authorized servers $\{E_{\mathcal{A}_i}\}_n$. Public key of DPRF $pk = g^\alpha$ is also published.

Upload Protocol

1. $\mathcal{U}$ first picks $\beta \overset{\$}{\leftarrow} \mathbb{Z}_p^*$, calculates $M \leftarrow H_G(D)$ and sends $\bar{M} = M \times g^\beta$ to $\{E_{\mathcal{A}_i}\}_n$.
2. $E_{\mathcal{A}_i}$ computes $bt_i' = \bar{M}^{sk_i}$ and returns bt_i' to $\mathcal{U}$.
3. $\mathcal{U}$ calculates $bt = \prod_{i \in \mathcal{S}} bt_i'^{\Delta_{i,\mathcal{S}}(0)} \cdot pk^{-\beta}$ as the blind tag and verifies the integrity of bt by the equation: $e(g, bt) \overset{?}{=} e(pk, M)$.

4/5. The same as the centralized scheme.

Download Protocol

The same as the centralized scheme.

Fig. 3. Semi-distributed Protocol.

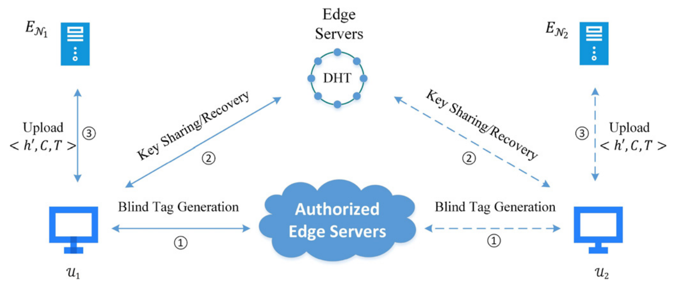


Fig. 4. User $\mathcal{U}_1$ and $\mathcal{U}_2$ want to upload the same data block D to different edge servers in deduplication scheme.

since we treated the existing edge servers as the key generators without revealing any valid information.

5. Distributed data deduplication solution

Considering that it may be difficult for some users to access at least the threashild number of authorized servers to generate a blind tag using DPRF, or it is difficult to select a sufficient number of authorized servers to guarantee the security, we propose a completely distributed solution without both additional central server or central service providers. We divide our upload protocol shown in Fig. 5 into three phases: **key identifier evaluation, key sharing/recovery** and **data encryption**.

Key Identifier Evaluation Unlike the centralized scheme or semi-distributed scheme, different users in the distributed scenarios cannot directly compute a blind tag for data block D in order to not reveal any information about D and avoid offline brute-force attacks. So we propose that the edge servers $\{E_{I_i}\}_{I_i \in \mathcal{E}}$ compute *kid* for $\mathcal{U}$ one by one instead of directly generate it from authorized server(s). Then $\mathcal{U}$ encrypts D and uploads the ciphertext C to $E_{\mathcal{N}}$.

$\mathcal{U}$ uses h as an input to evaluate OPRF between $\mathcal{U}$ and each server E_{I_i} indexed by DHT, then takes the pseudo-random values as the indexes of shares. However, directly evaluating OPRF and uploading the shares cannot defense offline brute-force attack since the servers who own the OPRF keys can also evaluate the pseudo-random indexes to guess users' data. Hence we require that the user should evaluate the OPRF with one server, but upload the result to another server. For security, pseudo-random permutation can be used to determine the mapping between OPRF evaluator and shares storage server. Here we just let the server E_{I_i} generate the OPRF value for the next server $E_{I_{i+1}}$.

Key Sharing or Recovery After generating the pseudo-random indexes, the user can check whether the data has been uploaded before by sending the pseudo-random indexes to the edge servers. If the indexes have been stored in the edge servers $\{E_{I_i}\}_{I_i \in \mathcal{E}}$, it means that the data has been uploaded and the protocol can be aborted. Otherwise, $\mathcal{U}$ picks a random value r, uses the hash value h to hide the secret shares of r and sends them to $\{E_{I_i}\}_{I_i \in \mathcal{E}}$. Otherwise, $\mathcal{U}$ uses h to recover the key from the shares.

Note that due to the distributed network environment, the users may obtain less than t shares. They should also be permitted to upload their secret shares. To distinguish whether different shares correspond to the same secret, each user should tag their shares with a unique value γ before sending it to servers. The servers store the combination of γ and secret shares as shown in Fig. 6.

Data Encryption $\mathcal{U}$ generates the key $k = r \oplus h$ and encrypts the data. Then he or she generates the search identifier h' and tag T of the ciphertext C, and sends $< h', C, T >$ to the edge server. After checking the consistency of C and T, the server stores $< h', C, T >$ and grants the access to $\mathcal{U}$.

Discussion The purpose of using OPRF is to ensure that no deterministic value of data is leaked to an offline brute-force attacker. On the one hand, although the collision resistant hash value h is used to derive an index list $\mathcal{E}$. Because of the scope of the key space in DHT, eavesdropper or edge servers can hardly guess the hash value of the data. On the other hand, the OPRF value of h can be seen by the edge servers, but it is impossible to perform brute-force attack by communicating with other servers to get the correct OPRF value.

The equality check of pseudo-random indexes may increase the overhead as the amount of data increases, the bloom filter or cuckoo hash structure can be applied to reduce the overhead. Since the pseudo-random indexes look like a random value to edge servers, these search methods are secure to perform in the scheme.

6. Analysis

We analyze the security of our schemes to meet the design goals in Section 3.3 and compare our schemes with existing secure deduplication schemes and point out our advantages. Since the first two goals have been satisfied, as explained in our proposed schemes in Section 4 and 5, we only focus on the last two goals in this section.

6.1. Confidentiality

Discussion Before proving the data confidentiality of our schemes, we first explain the strategies for inside adversaries to collide with private data by offline brute-force attacks. In our centralized or semi-distributed scheme, there exist two types of inside adversaries according to our threat model: 1) $\mathcal{A}_1$ controls CS and

Setup

Let $pp = <\mathbb{G}, g, p>$ be the public parameters and H, $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$ be the two public hash functions. Each edge server picks a random key $\mathcal{K}$ to evaluate OPRF.

Upload Protocol

1. $\mathcal{U}$ first calculates the hash $h \leftarrow H(D)$ and uses h as the access key of DHT to get an index list $\mathcal{E} \leftarrow DHT.Retrieve(h)$, where $\mathcal{E} = \{I_0, I_1, I_2, ..., I_n\}$
2. For $0 \le i \le n$, $\mathcal{U}$ sends a request to E_{I_i} to evaluate OPRF using h as input, and obtains the pseudo-random value $kid_{I_i} = F_{\mathcal{K}_{I_i}}(h)$.
3. For $0 \le i' \le n$, let $i = (i'+1)\ mod\ n$. $\mathcal{U}$ sends $kid_{I_{i-1}}$ to E_{I_i}. E_{I_i} searches in the secret share table using $kid_{I_{i-1}}$, and returns the secret shares set $\left\{<\gamma_j^{I_i}, \mathcal{R}(y_j) + s_j^{I_i} \quad mod \quad p, y_j \cdot g^h>\right\}_{j \le J_{I_i}}$ to $\mathcal{U}$ through a secure channel, where J_{I_i} is the number of users who have uploading shares of D to E_{I_i} before.
   - If less than t responses with the same γ are received, $\mathcal{U}$ first picks three random values $r \xleftarrow{\$} \mathbb{Z}_p, \gamma \xleftarrow{\$} \mathbb{Z}_p, y \xleftarrow{\$} \mathbb{G}$, generates the secret shares $\{s^{I_i}\}_n$ of r and then sends $<\gamma, \mathcal{R}(y) + s^{I_i}, y \cdot g^h>$ to the server E_{I_i} through a secure channel.
   - Else, $\mathcal{U}$ parses the response as $<\gamma, u, v>$ and recovers the shares as:

   $$s^{I_i} \leftarrow u - \mathcal{R}(v \cdot g^{-h})$$

   Then use sufficient shares $\{s^{I_1}, s^{I_2}, ..., s^{I_m}\}_{t \le m \le n}$ to recover the secret value: $r \leftarrow Recover(\{s^{I_i}\}_m)$.
4. $\mathcal{U}$ generates the key $k = h \oplus r$ and encryt D with k as: $C \leftarrow \boldsymbol{Enc}(D, k)$. Then he or she computes the tag $T = (T_1, T_2)$, where $T_1 \leftarrow \boldsymbol{TagGen}(C)$ and $T_2 \leftarrow \boldsymbol{TagGen}(T_1 || r)$, the search identifer $h' \leftarrow H'(r||h)$ and uploads $<h', C, T>$ as in centralized protocol.

Download Protocol

1. $\mathcal{U}$ with $<h, \gamma, h'>$ obtains a list of server indexes $\mathcal{E} \leftarrow DHT.Retrieve(h)$, and requests the OPRF indexes $\{kid_{I_i}\}$ from $\{E_{I_i}\}_{I_i \in \mathcal{E}}$. Then he or she uses these indexes to fetch shares set $\left\{<\gamma_j^{I_i}, \mathcal{R}(y_j) + s_j^{I_i} \quad mod \quad p, y_j \cdot g^h>\right\}_{j \le J_{I_i}}$ through a secure channel.
2. $\mathcal{U}$ uses γ as the identifier to check $\gamma \overset{?}{=} \gamma_j^{I_i}$ and recover at least t shares as in the upload protocol. Then he or she recovers the secret $r \leftarrow Recover(\{s^{I_i}\})$, and obtains the decryption key $k = h \oplus r$.
3. $\mathcal{U}$ requests the ciphertext C by sending h' to $E_{\mathcal{N}}$. If h' is not in the storage table of $E_{\mathcal{N}}$, $E_{\mathcal{N}}$ will forward the request to other servers. Otherwise, $E_{\mathcal{N}}$ returns C to $\mathcal{U}$.
4. $\mathcal{U}$ decrypts the ciphertext $D \leftarrow \boldsymbol{Dec}(C, k)$.

Fig. 5. Distributed Protocol.

any number of authorized edge servers, 2) $\mathcal{A}_2$ controls less than the threshold number of indexed edge servers. Both $\mathcal{A}_1$ and $\mathcal{A}_2$ can directly collide with a ciphertext with the form as $<h', C, T>$. Besides, $\mathcal{A}_1$ with the ability to evaluate blind tags can also collide with the index set $\mathcal{E}$ of indexed servers which accessed by the user before uploading the ciphertext. For $\mathcal{A}_2$, it can collide with search identifiers $\{kid_i\}$ of the ciphertext. In the distributed scheme, an inside adversary $\mathcal{A}_3$ can collide with the intermediate results of OPRF and one key identifier kid_i of the ciphertext since indexed edge servers will not collude with each other in the distributed scheme.

Theorem 1. *Our centralized and semi-distributed schemes achieve data confidentiality against offline brute-force attack performed by both the inside under the random oracle model.*

Proof. There exist the following strategies of attacks performed by two types of inside adversaries:

kid	$<\gamma_{D_1}^{I_i}, \mathcal{R}(y_{D_1}) + s_{D_1}^{I_i} \quad mod \quad q, y_{D_1} \cdot g^h>,$
	$<\gamma_{D_2}^{I_i}, \mathcal{R}(y_{D_2}) + s_{D_2}^{I_i} \quad mod \quad q, y_{D_2} \cdot g^h>,$
	$<\gamma_{D_3}^{I_i}, \mathcal{R}(y_{D_3}) + s_{D_3}^{I_i} \quad mod \quad q, y_{D_3} \cdot g^h>$
kid'	$<\gamma_{D'_1}^{I_i}, \mathcal{R}(y_{D'_1}) + s_{D'_1}^{I_i} \quad mod \quad q, y_{D'_1} \cdot g^h>,$
	$<\gamma_{D'_2}^{I_i}, \mathcal{R}(y_{D'_2}) + s_{D'_2}^{I_i} \quad mod \quad q, y_{D'_2} \cdot g^h>$

Fig. 6. Shares table of E_{I_i} after receiving three shares of data blocks D and two shares of D'.

1. π_1: $\mathcal{A}_1$ or $\mathcal{A}_2$ collides with a ciphertext $<h', C, T>$, where $h' = H'(bt||k)$. Let $\mathbb{G}$ and $\mathbb{K}$ be the space of bt and k respectively. If $H'(\cdot)$ is a random oracle, $\mathcal{A}_1$ without the knowledge of k has to collide with both bt and k. Althouth $\mathcal{A}_1$ with the keys to evaluate bt and can reduce the blind tag space from $\mathbb{G}$ to $\mathbb{D}$, where $\mathbb{D}$ is the space of input data, it still has to compute with the time complexity of $O(|\mathbb{K}|)$. For $\mathcal{A}_2$, it has to collide within the space size $|\mathbb{K}| \cdot |\mathbb{G}|$. The collision probability of π_1 can be formulated as:
$$p_{\pi_1} = \max\{p_{\pi_1}^{\mathcal{A}_1}, p_{\pi_1}^{\mathcal{A}_2}\} = \frac{1}{|\mathbb{K}| \cdot |\mathbb{D}|}$$
2. π_2: $\mathcal{A}_1$ collides with an index set $\mathcal{E}$ of indexed servers. Suppose there exist N indexed edge servers $\{E_1, E_2, \ldots, E_N\}$s. $\mathcal{A}_1$ first records the server indexes $\mathcal{E} = \{I_1, I_2, \ldots, I_n\}$ accessed by one user. Then it generates a set of blind tags $\{bt_1, bt_2, \ldots, bt_v\}$ and runs the offline brute-force attack to collide with these server indexes:
$$\{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_v\} \leftarrow DHT.Retrieve(\{bt_1, bt_2, \ldots, bt_v\}),$$
where v is polynomial numbers. If the hash function used in DHT is a random oracle, we can compute the collision probability as:
$$Pr[\mathcal{E} \in \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_v\}] = \frac{v \cdot n!(N-n)!}{N!}$$
and the probability of data equality under collision is:
$$Pr[D = D_i | \mathcal{E} = \mathcal{E}_i] = \frac{N}{|\mathbb{G}|}.$$
So the probability p of $\mathcal{A}_1$ to collide with the original data D is:
$$p_{\pi_2} = Pr[\mathcal{E} \in \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_v\}] \cdot Pr[D = D_i | \mathcal{E} = \mathcal{E}_i] = \frac{v \cdot n!(N-n)!}{|\mathbb{G}|(N-1)!}$$
If $\mathbb{G}$ is chosen with a proper security parameter, it is correct that p_{π_2} is a negligible value.
3. π_3: $\mathcal{A}_2$ collides with search identifiers $\{kid_i\}$ of the ciphertext, where $kid_i = bt \oplus H_G(h||I_i)$. $\mathcal{A}_2$ has to collide with both bt and h, so the probability is
$$p_{\pi_3} = \frac{1}{|\mathbb{G}| \cdot |\mathbb{D}|},$$
which is also a negligible value.

The inside adversaries can collide with input data with the negligible probability $p = \max\{p_{\pi_1}, p_{\pi_2}, p_{\pi_3}\}$, and our centralized and semi-distributed schemes are secure against offline brute-force attack. □

Theorem 2. *Our distributed scheme achieves data confidentiality against offline brute-force attack performed by inside adversaries under the security of OPRF.*

Table 2
AES modes vs. data validity.

AES mode	Poisoning attack	Integrity attack	Semantic security
ECB	Active	Passive	×
Others	Passive	Passive	√

Proof. For the inside adversary $\mathcal{A}_3$, assume that $\mathcal{A}_3$ controls ith indexed server. If OPRF is secure, $\mathcal{A}_3$ cannot know both the input h and the pseudo-random index kid_{I_i}, thus $\mathcal{A}_3$ can only collide with $kid_{I_{i-1}} = F_{\mathcal{K}_{I_{i-1}}}$. Without the knowledge of $\mathcal{K}_{I_{i-1}}$, the collision probability of data with $kid_{I_{i-1}}$ is negligible. □

For the outside adversaries, since they have less known information than inside adversaries, we omit the discussion of the security against offline brute-force attack performed by them. But we will prove that the confidentiality of servers' keys will not be corrupted by the outside adversaries.

Theorem 3. *The keys of CS and authorized edge servers is secure against outside adversaries under the discrete logarithm problem (DLP).*

Proof. Consider the outside adversary $\mathcal{A}$ sends the following queries to CS or authorized ES:
$$\{D_1, D_2, \ldots, D_v\} \xrightarrow{Query} \{bt_1, bt_2, \ldots, bt_v\},$$
where v is a polynomial number and $bt_i = H_G(D_i)^x$, where x is the secret key of CS or the authorized servers. If $\mathcal{A}$ can obtain x from $\{bt_1, bt_2, \ldots, bt_v\}$ and $\{D_1, D_2, \ldots, D_v\}$, then it can compute the DLP, which is known as a hard problem. □

6.2. AES mode and data validity

The auxiliary functions ***Enc***, ***Dec*** of symmetric encryption are used in our schemes for data encryption and decryption. According to the semantic security, AES modes can be generally divided into two types, namely, ECB mode and other modes. We discuss the data validity of our schemes with two types of AES modes separately. Since there is no central trusted server for authentication, it is reasonable to assume that the adversaries do not have the original data which they want to tamper with or poison. Otherwise it is impossible to achieve data validity in a semi-distributed or distributed scenario. We define the *active security* as storage servers being able to recognize the attacks, and *passive security* whereby users can both help storage servers to verify data validity and verify the validity of one downloaded ciphertext independently. We conclude the data validity of our schemes versus AES modes in Table 2.

Active Security. Our schemes achieve active security against content poisoning attack under the ECB mode. Since the ECB mode will always generate the same ciphertext of one input data, each storage server can directly compare ciphertext tags to recognize tampered or poisoned data. Consider an outside adversary trying to poison the search identifier h' and upload store $<h', C_{\mathcal{A}}, T_{\mathcal{A}}>$ to servers, where $C_{\mathcal{A}}$ and $T_{\mathcal{A}}$ are poisoned ciphertext and tag respectively. After receiving both the poisoned and the normal content, the storage servers can directly compare the poisoned ciphertext $C_{\mathcal{A}}$ with the normal ciphertext C to recognize content poisoning attack. Then it can verify which ciphertext is correct with the help of data owners. We will describe this step in the passive security later. Note that although the ECB mode is not semantically secure for offline encryption, it is not a necessary requirement for a storage system with the ability of deduplication, such as (Armknecht, 2015; Bellare et al., 2013; Kwon et al., 2019; Liu et al., 2015; Ni et al., 2018).

Passive Security. Our schemes achieve passive security against poisoning attack under the ECB mode and against integrity attack under any AES modes. We mark both tampered and poisoned content as $< h', C_{\mathcal{A}}, (T_{\mathcal{A},1}, T_{\mathcal{A},2}) >$ which is generated by an adversary $\mathcal{A}$. Since the adversary do not possess the original data D, it cannot generate a valid $T_{\mathcal{A},2}$ which satisfies $T_{\mathcal{A},2} = \boldsymbol{TagGen}(T_{\mathcal{A},1}||bt||k)$ in the centralized and semi-distributed schemes or $T_{\mathcal{A},2} = \boldsymbol{TagGen}(T_{\mathcal{A},1}||r)$ in the distributed scheme. Thus the content verification procedure of $< h', C, (T_1, T_2) >$ for an storage server contains the following steps:

1. The storage server sends $< h', T_1 >$ to all data owners.
2. Each data owner generates k or r as in the download protocol, and computes $T_{\mathcal{O},2} \leftarrow \boldsymbol{TagGen}(T_1||bt||k)$ in the centralized and semi-distributed schemes or $T_{\mathcal{O},2} \leftarrow \boldsymbol{TagGen}(T_1||r)$ in the distributed scheme. Then it returns $T_{\mathcal{O},2}$ to the server.
3. The server verifies whether the majority of the received tags are equal to T_2. If not, the content is considered as poisoned or tampered.

The verification procedure for data owners is similar except that the interaction steps are absent. In addition to assuming that the majority of data owners are normal as above, we can also introduce trust evaluation mechanism to evaluate trust scores of each user for verification (He et al., 2018; Zhao et al., 2020). We do not elaborate on this part since it is not our main contribution.

6.3. Discussion of online brute-force attack

An active outside adversary $\mathcal{A}$ can run an online brute-force attack to guess the data by observing whether deduplication happened. A client-side deduplication scheme reduces bandwidth overhead but cannot prevent online brute-force attack because $\mathcal{A}$ can guess the data according to whether he or she needs to upload. Here we discuss how to protect our schemes running in a server-side deduplication mode against $\mathcal{A}$.

In our centralize and semi-distributed schemes, the online brute-force attack can be performed as the following steps:

1. $\mathcal{A}$ selects data D' from the candidate set and sends the blind tag evaluation request to CS or the authorized servers.
2. Using blind tag bt as the access key of DHT, $\mathcal{A}$ receives the secret shares if D' is the target data. Then $\mathcal{A}$ can recover the key k from the shares.
3. $\mathcal{A}$ derives the search identifier h', requests the ciphertext from the server and decrypts it.

We introduce an additional grouping procedure on the server side before evaluating blind keys for users and modify the blind tag generation phase of our protocols.

Initialization The CS or authorized servers first choose(s) a short hash function $SH(\cdot)$ and generate(s) a set of secret keys $\{x_1, x_2, \ldots, x_{\mathcal{H}}\}$, where $\mathcal{H}$ is the value space of SH. Each key is grouped by a specific short hash value.

Blind Tag Generation $\mathcal{U}$ computes $sh \leftarrow SH(D)$ and sends a blind tag request with sh to the CS or authorized servers. Then the CS or authorized servers use(s) sh as a group identifier to select the secret key x_i for the blind tag evaluation.

To defend against online brute-force attacks, the CS or authorized servers can set a rate limit of SH to limit the frequency of blind tag evaluation of each short hash value. Once receiving the threshold number of the evaluation request over an epoch, the server(s) will reject further request in this epoch. For an adversary with a uniform candidate dataset $\mathcal{D}_{\mathcal{A}}$, the average collision time is:

$$t_c = \frac{1}{2}\left\lfloor \frac{|\mathcal{D}_{\mathcal{A}}|}{\mathcal{H} \cdot R_c} \right\rfloor \cdot \tau, \tag{1}$$

where R_c is the rate limit of SH during an epoch and τ is duration of each epoch. After uploading a dataset $\mathcal{D}$, the probability of a normal upload request being blocked is:

$$r_b = \sum_{i=R_c}^{|\mathcal{D}|} \binom{|\mathcal{D}|}{i} p_b^i (1 - p_b)^{|\mathcal{D}|-i}, \tag{2}$$

where $p_b = \frac{1}{\mathcal{H}}$. Based on Eqs. (1) and (2), we can balance $\mathcal{H}$, R_c and τ to defend from online brute-force attacks while maintaining service quality.

For the distributed scheme, similarly, we can use short hash to limit the OPRF evaluation in the edge servers $\{E_{I_i}\}$.

6.4. Comparison

There are two types of method to realize deduplication: 1) equality test or 2) ciphertext/tag collision check. The first method is more concise in the encryption phase and does not require additional interaction to generate the key. However, it is necessary to check different ciphertexts in the deduplication phase as shown in $\mu R-MLE2$ (Jiang et al., 2017), which leads to an increase in computation cost when the storage data increases.

The second method is more efficient in the deduplication phase, but it is difficult to ensure that users use the same key to encrypt the same data, especially in a distributed network environment. Unlike deduplication scheme proposed in Liu et al. (2015), which involves communication between the current data uploader and a set of data owners in key generation phase, our schemes only require the interaction between the data uploader and multiple edge servers. In addition, their scheme is also limited in achieving effective deduplication in a distributed environment, since it is unpractical for different servers to maintain the same user collection information. Thus their scheme fails to realize global deduplication, and it is hard to perform deduplication between different edge servers. A traditional centralized deduplication architecture such as *SEDS* shown in Nayak and Tripathy (2020) can realize global deduplication with the help of central key servers, which is not practical in actual scenarios. We conclude the advantages in Table 3 and show that our schemes are more secure and efficient to realize data deduplication in MEC.

We compare the computation overhead of our semi-distributed and distributed schemes with the deduplication schemes shown in Liu et al. (2015) and Jiang et al. (2017). Note that since these schemes are introduced in different system models, so we we compare the encryption and deduplication phase separately as shown in Table 4. Because the number of servers is less than the number of data users or records in general, we consider that our schemes achieve better actual performance.

7. Performance evaluation

In this section, we test the performance of Liu et al. (2015), a basic CE scheme as baseline and our schemes under different settings. Note that we choose (Liu et al., 2015) as comparison because its system model, which contains a number of online users for key generation and storage, bears the closest similarity to ours that introduce edge servers for key management. It is reasonable to use the same number of online users of Liu et al. (2015) and edge servers of our schemes respectively for overhead comparison.

We first construct a synthetic dataset to evaluate operation overhead compared with (Liu et al., 2015) since the content of the dataset only affects the deduplication rate and has no impact on the overhead. Thus we can synthesize datasets of different sizes to measure the performance of our schemes more accurately. We then evaluate the deduplication performance of our schemes on a real-world dataset compared with (Liu et al., 2015). The synthetic

Table 3
Compare with other deduplication schemes.

	Our schemes	Liu et al. (2015)	Jiang et al. (2017)	Nayak and Tripathy (2020)
Offline brute-force attack	√	√	×	√
Data validity	√	√	×	×
Global deduplication	√	×	×	√
Without additional servers	√	√	√	×

Table 4
Computation overhead comparison on the user side.

	Encryption	Deduplication
Semi-distributed scheme	$(2Mul + 2Exp) \cdot O(N)$*	$O(1)$
Distributed scheme	$(3Exp + 2Hash) \cdot O(N)$	$O(1)$
Liu's scheme Liu et al. (2015)	$(3Exp + 2Mul + Hash) \cdot O(M)$*	$O(1)$
$\mu R - MLE2(Dynamic)$ Jiang et al. (2017)	$O(1)$	$2Exp + Mul + Hash + (2Hash) \cdot O(Height)$*

*N is the number of edge servers. M is the number of users who run PAKE Liu et al. (2015). *Height* is the record tree height Jiang et al. (2017).

Table 5
Operation overhead on 1KB file.

Scheme	Operation	Time Usage (ms)	Bandwidth Usage (KB)
Cent.	*kid* Gen	5.39	0.32
scheme	Key Sharing	2.89	3.78
	total	8.68	5.13
Semi.	*kid* Gen	19.90	12.73
scheme	Key Sharing	2.89	3.78
	total	23.51	17.53
Dist.	*kid* Gen	76.47	6.46
scheme	Key Sharing	6.95	7.24
	total	87.02	14.72
Liu's	PAKE	115.27	24.92
scheme*	Elgamal Enc	113.48	13.10
Liu et al. (2015)	total	229.59	39.05
CE	Encryption	0.35	0
	total	0.71	1.08

*The overhead of running PAKE 20 times (which is consistent with $n = 20$ in our schemes).

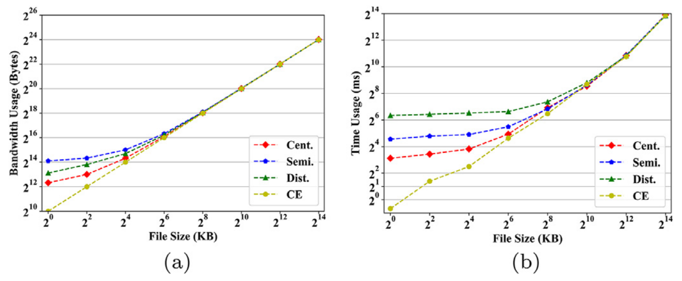


Fig. 7. Time (left) and bandwidth (right) usage vs. file size.

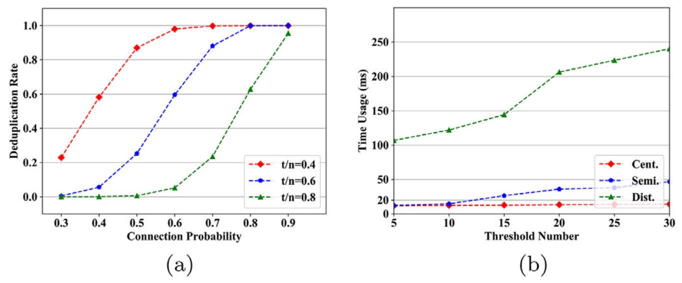


Fig. 8. Deduplication rate under different connection probabilities (left) and time usage under different threshold numbers of servers on 1KB file with $n = 30$ (right).

dataset is generated randomly with some specific data sizes according to the demand of experiments. The real-world dataset is collected by the File systems and Storage Lab at Stony Brook University (Tarasov et al., 2012). We focus on the snapshot of *Fslhomes* dataset (FSL, 2015) in April 2015, which contains 39 students' home directories with an average chunk size of 8KB from a shared network file system. The shared files consist of source code, binaries, office documents, virtual machine images, and other miscellaneous files.

Test setting We implement our schemes based on PBC library and use SHA-256 as the standard hash function. Besides, 256-bits AES with ECB mode is used for data encryption. We ran both the client-side and server-side program of our schemes on the test machine (Intel(R) Xeon(R) Platinum 8260 CPU 2.40GHz). We let both the server number n and threshold t of authorized servers be the same as the indexed servers of DHT, and set $n = 20, t = 12$ when evaluating the performance of our schemes. We bind each authorized edge/cloud server, DHT edge server and nearby edge server to a fixed port to provide evaluation service for users. We set up unordered hash tables on all edge servers to search for the key shares or ciphertext identifiers.

We measure the basic operation overhead of centralized (Cent.), semi-distributed (Semi.) and distributed (Dist.) schemes when uploading $1KB$ data to a nearby edge servers as shown in Table 5. In our centralized and semi-distributed schemes, the blind tag generation consumes the most computing overhead since it incurs the index calculation. In our distributed scheme, the *kid* should be evaluated on all the servers, leading to more computing overhead. It can be seen that both our centralized and semi-distributed are faster than the scheme of Liu et al. (2015). When the number of uploaded files is large, their scheme needs to increase the number of PAKE to ensure the deduplication rate, while the overhead of *kid* evaluation in our distributed scheme only depends on the setting threshold t.

As shown in Fig. 7, we measure the time and bandwidth usage in our schemes and CE scheme when uploading different file sizes to the nearby edge servers. The result shows that all of our schemes realize a similar performance as the CE scheme when the file size grows to more than 1 MB. Besides, considering both the security and performance, the semi-distributed scheme can achieve efficient data sharing with deduplication under the premise of security.

We measure the system performance under different network environments in Fig. 8(a). Since we cannot guarantee that all servers indexed by DHT will be connected to the users, we measure the relationship between deduplication rate and server connection probability under different threshold rates t/n. For example, in our former setting $t = 12$ and $n = 20$, the connection proba-

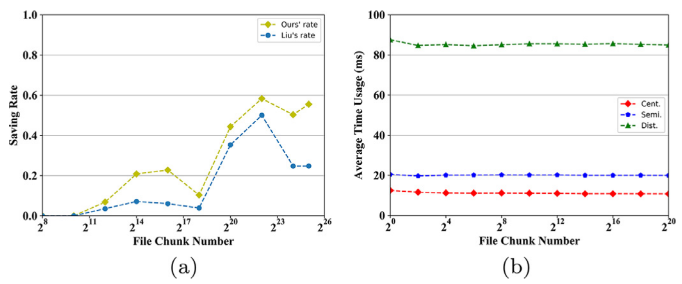


Fig. 9. Storage saving rate (left) and average time usage on key generation (right) evaluated on *Fslhomes* dataset.

bility should be higher than 0.8 to reach nearly 100% deduplication rate. We tested the time usage under different threshold number of edge servers in our schemes. Figure 8(b) shows that the performance of our centralized and semi-distributed schemes are less affected by the threshold number, so they are more suitable for large-scale network environments.

Besides, we test the deduplication performance of our schemes on *Fslhomes* dataset. To evaluate the deduplication performance, we define the storage saving rate as:

$$\rho = 1 - \frac{Server\ storage\ overhead}{Total\ data\ size}$$

To simulate a real mobile network environment, we set up users to upload file chunks to 10 different nearby edge servers. We evaluate both the storage saving rate and key generation time of our schemes on the *Fslhomes* dataset. Figure 9(a) shows that our schemes save more server storage than Liu's scheme. Since our three schemes adopt similar architectures, their deduplication rates are the same. Figure 9(b) shows that the number of uploaded file chunks has little impact on key generation performance in our schemes which ensures their performance stability in mobile scenarios.

8. Conclusion

In this paper, we have introduced multiple data security guarantees for MEC practical uses into three different network environments — centralized, semi-distributed, distributed. Compared with previous schemes, our semi-distributed scheme and distributed scheme can guarantee a high deduplication rate even when users keep their data on multiple servers. Through theoretical analysis, we prove the security of our schemes against typical attacks in outsourced data storage. Experimental results with a real-world deployment environment have showed that our schemes can guarantee service quality of MEC effectively. Since different MEC applications may have specific requirements for the quality of services and other constraints, we are considering to test the feasibility of our schemes in some real-world applications in our future works.

Declaration of Competing Interest

We declare that we have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Yu Lin: Conceptualization, Methodology, Software, Investigation, Investigation, Writing – original draft, Visualization, Supervision. **Yunlong Mao:** Conceptualization, Validation, Resources, Writing – review & editing, Project administration. **Yuan Zhang:** Writing – review & editing. **Sheng Zhong:** Project administration, Funding acquisition, Supervision.

Acknowledgements

The authors would like to thank the anonymous reviewers for the time and efforts they have kindly made. This work was supported in part by National Key R&D Program of China (2020YFB1005900), NSFC-61902176, BK20190294, NSFC-61872179, NSFC-61872176, and the Leading-edge Technology Program of Jiangsu NSF (BK20202001).

References

Agrawal, S., Mohassel, P., Mukherjee, P., Rindal, P., 2018. DiSE: distributed symmetric-key encryption. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. Association for Computing Machinery, New York, NY, USA, p. 19932010.

Armknecht, F., 2015. Transparent data deduplication in the cloud categories and subject descriptors. In: Ccs, pp. 886–900.

Beck, M.T., Werner, M., Feld, S., Schimper, T., 2014. Mobile edge computing: a taxonomy. In: Proc. of the Sixth International Conference on Advances in Future Internet., pp. 48–54.

Bellare, M., Keelveedhi, S., Ristenpart, T., 2013. Message-locked encryption and secure deduplication. In: Johansson, T., Nguyen, P.Q. (Eds.), Advances in Cryptology – EUROCRYPT 2013. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 296–312.

Cisco Visual Networking Index, 2017. Global Mobile Data Traffic Forecastupdate 2016–2021. Technical Report.

Dai, Y., Xu, D., Maharjan, S., Zhang, Y., 2018. Joint load balancing and offloading in vehicular edge computing and networks. IEEE Internet Things J..

Douceur, J.R., Adya, A., Bolosky, W.J., Simon, P., Theimer, M., 2002. Reclaiming space from duplicate files in a serverless distributed file system. In: Proceedings 22nd International Conference on Distributed Computing Systems, pp. 617–624.

Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O., 2005. Keyword search and oblivious pseudorandom functions. In: Theory of Cryptography Conference. Springer, pp. 303–324.

Fsl traces and snapshots public archive, 2015. https://tracer.filesystems.org/traces/fslhomes/2015/.

Geambasu, R., Kohno, T., Levy, A.A., Levy, H.M., 2009. Vanish: increasing data privacy with self-destructing data. In: Monrose, F. (Ed.), 18th USENIX Security Symposium, Montreal, Canada, August 10–14, 2009, Proceedings. USENIX Association, pp. 299–316.

He, Y., Zhao, N., Yin, H., 2018. Integrated networking, caching, and computing for connected vehicles: a deep reinforcement learning approach. IEEE Trans. Veh. Technol. 67 (1), 44–55.

Jarecki, S., Krawczyk, H., Resch, J.K., 2019. Updatable oblivious key management for storage systems. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (Eds.), Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11–15, 2019. ACM, pp. 379–393.

Jiang, T., Chen, X., Wu, Q., Ma, J., Susilo, W., Lou, W., 2017. Secure and efficient cloud data deduplication with randomized tag. IEEE Trans. Inf. Forensics Secur. 12 (3), 532–543.

Kwon, H., Hahn, C., Kang, K., Hur, J., 2019. Secure deduplication with reliable and revocable key management in fog computing. Peer-to-Peer Netw. Appl. 12 (4), 850–864.

Li, J., Chen, X., Huang, X., Tang, S., Xiang, Y., Hassan, M.M., Alelaiwi, A., 2015. Secure distributed deduplication systems with improved reliability. IEEE Trans. Comput. 64 (12), 3569–3579.

Li, J., Su, Z., Guo, D., Choo, K.-K.R., Ji, Y., Pu, H., 2020. Secure data deduplication protocol for edge-assisted mobile crowdsensing services. IEEE Trans. Veh. Technol. 70 (1), 742–753.

Lim, W.Y.B., Luong, N.C., Hoang, D.T., Jiao, Y., Liang, Y.C., Yang, Q., Niyato, D., Miao, C., 2020. Federated learning in mobile edge networks: acomprehensive survey. IEEE Commun. Surv. Tutor. 22 (3), 2031–2063.

Liu, J., Asokan, N., Pinkas, B., 2015. Secure deduplication of encrypted data without additional independent servers. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 874–885.

Liu, J., Wang, J., Tao, X., Shen, J., 2017. Secure similarity-based cloud data deduplication in ubiquitous city. Pervasive Mob. Comput. 41, 231–242.

Liu, J., Zhao, T., Zhou, S., Cheng, Y., Niu, Z., 2014. CONCERT: a cloud-based architecture for next-generation cellular systems. IEEE Wirel. Commun. 21 (6), 14–22.

Lv, L., Zhang, Y., Li, Y., Xu, K., Wang, D., Wang, W., Li, M., Cao, X., Liang, Q., 2019. Communication-aware container placement and reassignment in large-scale internet data centers. IEEE J. Sel. Areas Commun. 37 (3), 540–555.

Ma, L., Yi, S., Carter, N., Li, Q., 2018. Efficient live migration of edge services leveraging container layered storage. IEEE Trans. Mob. Comput..

Mao, Y., Hong, W., Wang, H., Li, Q., Zhong, S., 2020. Privacy-preserving computation offloading for parallel deep neural networks training. IEEE Trans. Parallel Distrib. Syst.. 1–1

Mao, Y., Yi, S., Li, Q., Feng, J., Xu, F., Zhong, S., 2018. A privacy-preserving deep learning approach for face recognition with edge computing. In: Proc. USENIX Workshop Hot Topics Edge Comput.(HotEdge), pp. 1–6.

Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B., 2017. A survey on mobile edge computing: the communication perspective. IEEE Commun. Surv. Tutor. 19 (4), 2322–2358.

Naor, M., Pinkas, B., Reingold, O., 1999. Distributed pseudo-random functions and KDCs. In: International Conference on the Theory and Applications of Cryptographic Techniques. Springer, pp. 327–346.

Naor, M., Pinkas, B., Reingold, O., 1999. Distributed pseudo-random functions and KDCs. In: International Conference on the Theory and Applications of Cryptographic Techniques. Springer, pp. 327–346.

Nayak, S.K., Tripathy, S., 2020. SEDS: secure and efficient server-aided data deduplication scheme for cloud storage. Int. J. Inf. Sec. 19 (2), 229–240.

Ni, J., Zhang, K., Yu, Y., Lin, X., Shen, X.S., 2018. Providing task allocation and secure deduplication for mobile crowdsensing via fog computing. IEEE Trans. Dependable Secure. Comput. 17 (3), 581–594.

Ren, J., He, Y., Huang, G., Yu, G., Cai, Y., Zhang, Z., 2019. An edge-computing based architecture for mobile augmented reality. IEEE Netw. 33 (4), 162–169.

Taleb, T., Ksentini, A., Frangoudis, P.A., 2019. Follow-me cloud: when cloud services follow mobile users. IEEE Trans. Cloud Comput. 7 (2), 369–382.

Tarasov, V., Mudrankit, A., Buik, W., Shilane, P., Kuenning, G., Zadok, E., 2012. Generating realistic datasets for deduplication analysis. In: 2012 {USENIX} Annual Technical Conference ({USENIX}{ATC} 12), pp. 261–272.

Xiao, Y., Jia, Y., Liu, C., Cheng, X., Yu, J., Lv, W., 2019. Edge computing security: state of the art and challenges. Proc. IEEE 107 (8), 1608–1631.

Xu, Q., Su, Z., Lu, R., 2020. Game theory and reinforcement learning based secure edge caching in mobile social networks. IEEE Trans. Inf. Forensics Secur. 15, 3415–3429.

Xu, Q., Su, Z., Zheng, Q., Luo, M., Dong, B., 2018. Secure content delivery with edge nodes to save caching resources for mobile users in green cities. IEEE Trans. Ind. Inf. 14 (6), 2550–2559.

Zhang, Y., Chen, C.P., 2021. Secure heterogeneous data deduplication via fog-assisted mobile crowdsensing in 5G-enabled IIoT. IEEE Trans. Ind. Inf..

Zhao, P., Huang, H., Zhao, X., Huang, D., 2020. P3: Privacy-preserving scheme against poisoning attacks in mobile-edge computing. IEEE Trans. Comput. Social Syst. 7 (3), 818–826.

Yu Lin is pursuing his M.S. degree with the Department of Computer Science and Technology of Nanjing University. He received the B.S. degree in microelectronic science and engineering from Nankai University in 2015. His current research interests include security and privacy.

Yunlong Mao received the BS and PhD degrees in computer science from Nanjing University, Nanjing, China, in 2013 and 2018, respectively. He is currently an assistant researcher with the Department of Computer Science and Technology, Nanjing University. His current research interests include security, privacy, and machine learning.

Yuan Zhang received the B.S. degree in automation from Tianjin University in 2005, the M.S.E. degree in software engineering from Tsinghua University in 2009, and the Ph.D. degree in computer science from the State University of New York at Buffalo in 2013. His current research interests include security, privacy, and economic incentives.

Sheng Zhong received the B.S. and M.S. degrees from Nanjing University in 1996 and 1999, respectively, and the Ph.D. degree from Yale University in 2004, all in computer science. He is interested in security, privacy, and economic incentives.