Privacy-Preserving Data Integrity Verification in Mobile Edge Computing

Wei Tong*, Bingbing Jiang*, Fengyuan Xu*, Qun Li†, and Sheng Zhong*
*State Key Laboratory for Novel Software Technology, Nanjing University, China
†Department of Computer Science, College of William and Mary, USA
Email: weitong@outlook.com, njubing.jiang@gmail.com, †liqun@cs.wm.edu, {fengyuan.xu, zhongsheng}@nju.edu.cn

***Abstract*—Mobile edge computing (MEC) is proposed as an extension of cloud computing in the scenarios where the end devices desire better services in terms of response time. Edge nodes are deployed at the proximity of the end devices, and it can pre-download parts of data stored in the cloud so that the end devices can access these data with low latency. However, because the edges are usually owned by individuals and small organizations, which have limited operation capacities for maintaining the machines, the data on the edges is easily corrupted (due to external attacks or internal hardware failures). Therefore, it is essential to verify data integrity in the MEC. We propose two Integrity Checking protocols for MEC, called ICE-basic and ICE-batch, which are designed for the cases where the user wants to check data integrity on a single edge or multiple edges, respectively. Based on the concept of provable data possession and the technique of private information retrieval, our protocol allows a third party verifier to check the data integrity on the edges without violating users' data privacy and query pattern privacy. We rigorously prove the security and privacy guarantees of the protocols. Furthermore, we have implemented a proof-of-concept system that runs ICE, and extensive experiments are conducted. The theoretical analysis and experimental results demonstrate the proposed protocols are efficient both in computation and communication.**

***Keywords*-Mobile edge computing, data integrity, privacy-preserving**

I. Introduction

Cloud computing has made tremendous success in the past decade, which provides powerful, scalable, and reliable computing and storage for average users and enterprises. However, when it comes to the mobile computing, the clouds cannot meet the requirements of mobility support and low response time for many mobile applications, due to the large geographic distances between the clouds and the end devices. Mobile edge computing (MEC) [1] has been proposed as a new architecture for dealing with the issue. It pushes the computation capability and data storage from cloud to the edge nodes which are at the proximity of the end devices, such that the end devices can enjoy the computing and data access services with low latency.

When outsourcing data to the remote servers (*e.g.*, cloud), the users cannot fully control their data because they no longer possess these data locally, and the data integrity is a major concern in this circumstance. For the cloud computing, the concern has been addressed by lots of previous works in the last decade, *e.g.*, [2]–[8]. We find that mobile edge computing also faces the similar problem regarding the data integrity, and it requires that the integrity verification is not only performed on the back-end cloud but also the edge nodes. In the MEC, the edges pre-download a part of the data and services from the back-end cloud so that it can provide data access for end devices in time. Though the data stored in the back-end cloud is intact, the pre-downloaded data on the edges may also be corrupted. (For example, the edge services usually belong to individuals or small organizations, which may have limited IT operation techniques.)

Furthermore, the edges are expected to deploy a delayed write-back strategy for communication efficiency. Thus, the edges usually will not write-back the updated data blocks to the back-end cloud immediately, even though the end devices may frequently update the data blocks on the edges. In this case, the back-end cloud has no ability to recover the updated data for users if the data on the edges is corrupted, and the users will permanently lose their data. Therefore, data integrity verification, *i.e.*,checking whether the data is removed or tampered, is not only important on back-end cloud, but also essential for edges.

As mentioned above, many works have been proposed for integrity verification. However, the previous approaches may not work well when the data integrity problem comes to the edge computing. First, previous works implicitly assume that the verifier knows what data the remote server should possess. For the edge computing, it is inevitable to reveal information of which parts of data the edges have pre-downloaded to the verifier, but this information may imply users' query pattern privacy. Second, edge computing has different trust model compared with other architectures. Unlike Internet cache, in which the service provider has the responsibility to protect the data integrity on cache servers, the remote cloud and edge nodes may belong to different entities in the edge computing, and the cloud has no responsibility for verifying the data integrity on edges.

We identify two major challenges in verifying the data integrity on the edges. The first challenge is that the edge nodes have pre-downloaded a subset of whole data, and the user does not know what part of data the edge nodes have possessed, which is also a unique challenge for edge computing. The other major challenge is that a user may access the edges with multiple devices (*e.g.*, laptop, phone, sensors) at different time. Therefore, the users need a verifier

to maintain the data integrity tags and perform the verifications [7]. Furthermore, in this process, users' and edges' privacy information ought not to be revealed to the verifier.

In this paper, we study the integrity checking problem for a general-purpose data storage service in edge computing and proposed protocols for it. The *homomorphic verifiable tags (HVT)* [3], [8] is adopted to achieve the integrity verification without downloading the data stored in the edges. By incorporating the *private information retrieval (PIR)* [9], [10] and the *tag repacking* technique, the protocols preserve the data privacy and users' query pattern against the third party verifiers. To sum up, the proposed protocols have three desirable properties. First, it can verify the data integrity on the edge nodes without downloading the data from them. Second, both the pre-download strategy of the edge and the users' query pattern are preserved against the third party verifier. Third, the verification is efficient, which meets the requirement of response time of mobile computing.

In our basic protocol, ICE-basic, we consider the case where the user connects to a single edge node, and needs to verify the data integrity on one edge at each time. Furthermore, we attend to the scenario that a user may want to verify the integrity of data on multiple edges at the same time and have proposed an extended protocol, ICE-batch for efficient batch verification of multiple edges.

Our contributions can be summarized as follows:

- First, to our best knowledge, we are the first to study the problem of data integrity verification in edge computing. We consider a case where the users need third party verifiers to check the integrity of data blocks that have been pre-downloaded on the edge nodes in a privacy-preserving manner.
- Second, we propose a remote data integrity checking protocol for edge storage, and an extended protocol is also proposed for efficient batch verification of multiple edges. The proposed protocols are privacy-preserving against semi-honest verifier, secure against untrusted edge nodes, and efficient in terms of the requirements of edge computing.
- Third, we analyze the correctness and security of the proposed protocols, and rigorously prove that it is secure against the untrusted edges and private against the third-party verifiers.
- We have implemented a proof-of-concept system and conducted extensive experiments to evaluate the proposed protocols. The results show that our protocols are efficient.

The rest of this paper is organized as follows. We present the technical preliminaries in Sec. II. In Sec. III and Sec. IV, we present the ICE-basic protocol and the corresponding analysis, respectively. The ICE-batch protocol is presented in Sec. V. In Sec. VI, the evaluation results will be presented. In Sec. VII, we review the related works. Finally, we conclude in Sec. VIII.

II. Preliminaries

A. Edge computing and System Model

We consider the data-based services that leverage the architecture of the edge computing (*e.g.*, QoS-based video access [11], location-based information retrieval, and machine learning based applications [12], [13]) . In such applications, when a user wants to fetch some data from the cloud, it sends the request via the edges. If the edges hold the data that the user queries, they can directly deliver the data without downloading from the cloud, and thus the latency is reduced. For providing better services, the edges will pre-download a subset of data from the cloud based on the users' queries. In terms of the data integrity, the uniqueness of the edge-cloud architecture is two-fold. First, the back-end cloud and the edges are usually belong to different entities, which means the cloud has no obligation to ensure the data integrity on edges, and the users ought to verify. Second, the edges are deployed at the proximity of the users. Their computation and storage capabilities are more powerful than end users, but cannot match the back-end cloud. Therefore, the edges can only pre-download parts of data from the cloud.

Our work aims to provide general-purpose data integrity verification in edge computing. Specifically, we consider an edge-cloud system with several edge nodes and a back-end cloud that are connected through WAN-based connections. The edge nodes (*e.g.*, Wi-Fi access points, enterprise servers) are deployed at the proximity of the end devices (*e.g.*, mobile phones, sensors, PCs), and the end devices are directly connected to the edge nodes [1]. The end devices intend to check the integrity of data in the edge nodes. A third party auditor is usually required to perform the integrity check, due to the limit of computing and storage capabilities and the fragility of the end devices. To sum up, there are four types of entities in such an edge-cloud system:

- *User*: A User is an end device, which queries the data and requests for the data integrity verification on edges.
- *Cloud Service Provider (CSP)*: We assume that the CSP has unlimited computing and storage capabilities. Denote by F the file that is stored in the cloud, which includes n data blocks: $b_1, b_2, \ldots, b_n$, and L the length of the file.
- *Edge*: A edge node could be an enterprise server, a base station, and etc. These edges can be reached by nearby end devices with low latency. For a edge edge_j, it pre-downloads a subset of data blocks $F_j = \{m_{j,1}, m_{j,2}, \ldots, m_{j,n_j}\}$ from the back-end cloud, and denoted by S_j the indexes of these blocks in F.
- *Third Party Auditor (TPA)*: The TPAs have powerful computing and storage capabilities, which maintain the data integrity tags and perform the verifications.

B. Trust Model

This work focuses on the data integrity checking in edges. There are many previous works (*e.g.*, [3], [5], [8]) having

studied the problem of providing data integrity checking in the cloud, and we assume that the data stored in the back-end cloud is secure in our work.

We assume that the edge nodes are untrusted. One of the benefit of edge computing is that the edge nodes are often deployed at the proximity of the users, and even owned by the users (*e.g.*, PC or routers). However, a major concern is that average users and small organizations may have limited IT operation techniques to maintain the machines. Thus, these edge nodes may suffer internal failures and are easily attacked by an external adversary, and data corruption often happens. Therefore, the user cannot get the correct data even the data in the cloud is intact.

For the auditor, which is often a cloud service, and we assume that it is semi-honest, who is interested in getting users' data, query pattern, and edge nodes' strategies (*i.e.*, which part of data is pre-downloaded) so that it can pry the users' sensitive information by analyzing the collected information. Specifically, the auditor may try to require the subsets of indexes (*i.e.*, S_j) for performing the verification on edges, because it seems to be an indispensable prerequisite to know what the edges have stored before checking them. However, in a edge-powered data storage application, which pre-downloads the data based on the users' queries, such indexes may reveal users' query patterns. Query pattern is an essential type of sensitive information, and it may reveal a user's preference, locations, and other privacy in a certain type of applications (*e.g.*, video applications, location-based information retrieval).

C. Design Goals

To ensure the security for edge-cloud system under the aforementioned adversary model, we aim to design mechanism for verifying data with the following goals:

- Edge storage correctness: ensuring that the edge nodes cannot cheat to pass the audit.
- Public audibility: the user can require a TPA to verify the integrity of data in the cloud and edges nodes.
- Privacy-preserving: when performing the integrity checking, the TPA cannot get other private information (including retrieved data, query patterns, and edges' strategies) from both the user and the edge.
- Efficient: allowing the TPA and the user to perform verification with reasonable overheads.

III. Integrity Checking in Edge Computing

Our goal is to design a protocol that allows users to verify data integrity on edge nodes without downloading the data and knowing which part of data the edge nodes have pre-downloaded. The proposed approach is based the provable data possession (PDP) schemes [3], [6], [8] and private information retrieval (PIR) techniques [9], [10]. Below we first address the technical challenges in designing a integrity checking protocol in edge computing. Then we describe the details of the proposed protocol and the private tag retrieval that we have adopted within it.

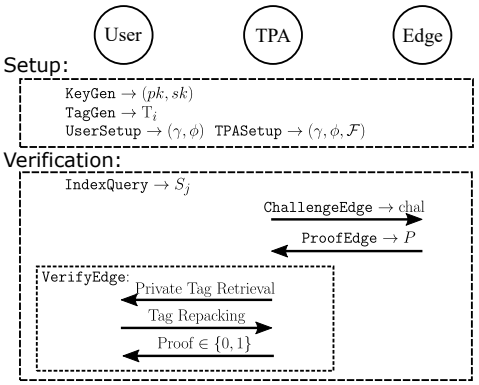


Figure 1. Information flow between User, Edge, and TPA.

As we have mentioned, the data stored by the edges may be tampered with or removed due to external attacks or internal failures. Compared with the traditional cloud architecture, the difficult of solving the data integrity verification problem in the edge architecture is twofold. On the one hand, the computation and storage capacities of the end devices are limited, and they many need third party auditors to store the tags and assist in verifying the integrity. One the other hand, if the integrity verification is performed by the third party auditors, it seems inevitable to reveal which subset of data that the edge node has pre-downloaded to the TPA.

A. ICE *Protocol*

Basically, in the proposed protocol, the user generates a tag for each data block before storing it to the remote servers (*i.e.*, back-end cloud or edges), and these tags will be stored by the TPA. Upon receiving a data integrity checking request from the user, the TPA generates a $\mathbf{challenge}$ randomly based on user's public key and sends it to the edge. The edge node is required to prove that the data is not tampered or removed by computing a $\mathbf{proof}$ based on the received $\mathbf{challenge}$ and the stored data. When receiving the $\mathbf{proof}$, the TPA can verify the data integrity on the edge based on the tags, the $\mathbf{challenge}$, and the $\mathbf{proof}$, and the verification can be done in a privacy-preserving way by resorting to the private tag retrieval scheme in Sec. III-B.

The interactions between User, Edge, and TPA in the protocol are shown in Fig. 1. The protocol consists of 8 components: KeyGen, TagGen, UserSetup, TPASetup, IndexQuery, ChallengeEdge, ProofEdge, and VerifyEdge.

$\mathbf{KeyGen}(1^K) \rightarrow (pk, sk)$. Given the parameter K, this probabilistic algorithm generates a public key pk and a secret key sk for the user. Let $pk = (N, g)$, and $sk = (p, q)$, where $N = pq$ is a publicly known RSA modulus; g is a generator, which satisfies $g = b^2$, where $\gcd(b \pm 1, N) = 1$; p and q are two large primes, which satisfies that $p = 2p' + 1$, $q = 2q' + 1$, and p' and q' are also primes.

$\mathbf{TagGen}(sk, pk, b_i) \rightarrow \mathrm{T}_i$. Given pk, and the data block b_i, this operation generates a verification tag T_i for b_i. The tags will be stored by the TPA for performing integrity verification. Denote the set of tags by $\mathcal{T} = \{\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_n\}$. For each data block b_i, the tag is computed as

$$\mathrm{T}_i = (g^{b_i}) \mod N.$$

$\mathbf{UserSetup}(n) \rightarrow (\gamma, \phi)$. Given the number of data blocks n, this operation returns a parameter γ for private tag retrieval from the TPA and an embedding ϕ of the n indexes of data blocks into points in a Hamming space $\{0,1\}^\gamma$ with weight 3. In the proposed protocol, we set $\gamma = \lceil (6n)^{1/3} \rceil + 2$ base on the private retrieval requirement [10].

$\mathbf{TPASetup}(n, \mathcal{T}) \rightarrow (\gamma, \phi, \mathcal{F})$. Given the n and the generated tags $\mathcal{T}$, this operation also returns γ and ϕ as UserSetup. In addition, it returns a polynomial representation of the tags for private tag retrieval. The set of tags can be represented by a bit matrix

$$\mathbf{T} = \begin{pmatrix} \mathbf{t}_1 \\ \vdots \\ \mathbf{t}_n \end{pmatrix} = \begin{pmatrix} t_{11} & \ldots & t_{1K} \\ \vdots & \ddots & \vdots \\ t_{n1} & \ldots & t_{nK} \end{pmatrix},$$

where $\mathbf{t}_i = (t_{i1} t_{i2} \cdots t_{iK})$ is a bit vector that represents tag T_i. Denoted by $\mathbf{t}'_\pi = (t_{1\pi}\ t_{2\pi}\ \cdots\ t_{n\pi})$, for $\pi = 1, 2, \ldots, K$, and the set of polynomials is denoted by $\mathcal{F} = \{F_1, F_2, \ldots, F_K\}$. For each bit vector $\mathbf{t}'_\pi$, the constructed polynomial is

$$F_\pi(x_0, x_1, \ldots, x_{\gamma-1}) = \sum_{i=1}^{n} t_{i\pi} \prod_{j:\phi(i)[j]=1} x_j. \qquad (1)$$

$\mathbf{IndexQuery}(\mathrm{edge}_j) \rightarrow S_j$. Given the ID of the edge_j, this operation returns a set of indexes S_j of data blocks that have been pre-downloaded by edge_j. In addition, the set of data blocks that have been pre-downloaded is denoted by F_j.

$\mathbf{ChallengeEdge}(pk) \rightarrow \mathrm{chal}$. This operation generates a challenge chal by the TPA for requesting an integrity proof of F_j from the edge. The verifier generates a random key $e \in [1, 2^K - 1]$ and a random element $s \in \mathbb{Z}_N^*$, and sends challenge $\mathrm{chal} = (e, g_s)$, where $g_s = g^s \mod N$.

$\mathbf{ProofEdge}(F_j, \mathrm{chal}, \tilde{s}) \rightarrow P$. The edge computes a response to prove the integrity based on the challenge. It first generates a sequence of coefficients for the subset of blocks that it possesses based on the received random key e, denoted by $a_1, a_2, \ldots, a_{n_j}$, then computes

$$P = (g_s)^{\sum_{k=1}^{n_j} a_k m_{j,k} \tilde{s}} \mod N,$$

where $\tilde{s}$ is a random element in $\mathbb{Z}_N^*$, which is generated by the user and shared to the edge,and recall that $\{m_{j,k}\}_{k=1,2,\ldots,n_j}$ are the data blocks on the edge_j.

$\mathbf{VerifyEdge}(pk, \mathcal{T}, chal, P, S_j) \rightarrow \{0, 1\}$. This process consists of the following steps:

1) Based on the set of indexes S_j, the user get the corresponding set of tags $\mathcal{T}_j$ from the TPA without revealing the indexes in S_j. The details of the private tag retrieval will be described in Sec. III-B.
2) For $\mathrm{T}_{j,k} \in \mathcal{T}_j$, the user calculates

   $$\tilde{\mathrm{T}}_{j,k} = \mathrm{T}_{j,k}^{\tilde{s}} \mod N,$$

   where $\tilde{s}$ is a random element in $\mathbb{Z}_N^*$. The user then sends the regenerated set of tags $\tilde{\mathcal{T}}_j = \{\tilde{T}_{j,1}, \tilde{\mathrm{T}}_{j,2}, \ldots, \tilde{\mathrm{T}}_{j,n_j}\}$ to the TPA. If a data block $m_{j,k}$ is updated by the user in the current session, the user replace the corresponding regenerated tag by

   $$\tilde{\mathrm{T}}'_{j,k} = g^{m'_{j,k}\tilde{s}} \mod N,$$

   where $m'_{j,k}$ is the updated data block.
3) The TPA generates the coefficients $a_1, a_2, \ldots, a_{n_i}$ based on the random key e, and each coefficient has a length of d bits. Then it computes

   $$R = \prod_{k=1}^{n_j} (\tilde{\mathrm{T}}_{j,k}^{a_k} \mod N) \mod N,$$
   $$\tilde{P} = R^s \mod N.$$

   And then the TPA checks whether P equals to $\tilde{P}$. If $P = \tilde{P}$, return 1; otherwise, return 0.

B. Private Tag Retrieval

In the tag retrieval process, the privacy of users' access pattern should be protected, which means that the subset of indexes cannot be revealed to the TPAs. A trivial approach to achieve the goal is to retrieve all the tags and save the desirable tags corresponding to the subset of indexes. However, such an approach brings large communication overheads, which makes the verification unpractical.

We find that the private tag retrieval is a kind of private information retrieval (PIR), and there are two major lines of studies having been done extensively in the previous works, information-theoretic PIR (*e.g.*, [9], [10]) and computational PIR (*e.g.*, [14], [15]). The computational PIR may require large computation overheads for users and auditors. We adopt an geometric approach-based information-theoretic PIR scheme [9], [10] to realize our design. The private tag retrieval requires that there are two TPAs, which store the tags of data blocks separately, and which will not collude with each other. It is easy to realize for the real-world edge computing systems. The verification is usually a kind of cloud service, and the system only needs to rent two distinct TPAs that are deployed at different clouds.

The overview of the scheme

Basically, based on the PIR scheme [9], the user first encodes it its query to protect the actual indexes of tags that are expected to retrieved; then the TPAs response to the received user queries based on the stored tags (which are encoded by

polynomials); the user decodes the responses sent by the TPAs and get the retrieved tags. Specifically, the private tag retrieval consists of the following steps:

► Tag Query. The user encodes its query to protect the actual indexes of tags that are expected to retrieved. Denote by $\{Q_\tau\}_{\tau=0,1}$ is the query by the user. For each index of data blocks, the user first generates a random element $\mathbf{z}_l \in \mathbb{F}_4^\gamma$ uniformly, and then appends $\phi(j_l) + t_\tau \mathbf{z}_l$ (for $\tau = 0, 1$) to the query, where ϕ is the embedding in UserSetup of the protocol.

► Tag Response. The TPAs response to the received user queries based on the stored tags (which are represented by polynomials). For query of each tag $\mathbf{q}_l \in Q_\tau$, the auditor generates the response $(v_{l,\pi}, \mathbf{g}_{l,\pi})_{\pi=1,2,\ldots,K}$, where

$$v_{l,\pi} \leftarrow \texttt{eval}(M_\pi, \mathbf{q}_l);$$
$$\mathbf{g}_{l,\pi} \leftarrow (\texttt{eval}(M_\pi^1, \mathbf{q}_l) \ \cdots \ \texttt{eval}(M_\pi^\gamma, \mathbf{q}_l)),$$

where M_π is a matrix representation of the polynomial $F_\pi(x_0, x_1, \ldots, x_{\gamma-1})$ that stores the π-th bits of all the tags in the TPA. $M_\pi^1, M_\pi^2, \ldots, M_\pi^\gamma$ are matrices represent the partial derivatives of the polynomial, and `eval` is an evaluation that performs query on corresponding matrix. The details of matrix representation of the polynomials will be described later.

► Tag Decoding. The user decodes the responses sent by the TPAs and get the retrieved tags. For each queried index of tag, the user computes the π-th bit of the tag by corresponding value $v_{l,\pi}^\tau$ ($\tau = 0, 1$) and partial derivatives $\mathbf{g}_{l,\pi}^\tau$ ($\tau = 0, 1$) responded by TPAs. First of all, the user computes a vector $\mathrm{u}_{l,\pi} \leftarrow (v_{l,\pi}^0 \ \langle \mathbf{g}_{l,\pi}^0, \mathbf{z}_l \rangle \ v_{l,\pi}^1 \ \langle \mathbf{g}_{l,\pi}^1, \mathbf{z}_l \rangle)^\mathsf{T}$, where $\langle \cdot, \cdot \rangle$ means the inner product of two vectors. And then get the π-th bit of the tag by $T_{j_l}[\pi] \leftarrow (M^{-1}\mathrm{u}_{l,\pi})[0]$, where M is a decoding matrix introduced by [10].

For more details, a formal description of the private tag retrieval is shown in Alg. 1.

Algorithm 1: Privately fetching tags from TPAs

Input: S_j, γ, ϕ, $\mathcal{F}$
Output: $\mathcal{T}_j$

$t_0 \leftarrow 1$; $t_1 \leftarrow 2$;

User: `tag query`
$Q_\tau \leftarrow \texttt{List}()$, for $\tau = 0, 1$;
foreach $j_l \in S_j$ **do**
  Generating $\mathbf{z}_l \in \mathbb{F}_4^\gamma$ uniformly;
  $Q_\tau.\texttt{append}(\phi(j_l) + t_\tau \mathbf{z}_l)$, for $\tau = 0, 1$;
  $\texttt{Send}(Q_\tau, \text{Auditor}_\tau)$, for $\tau = 0, 1$;
end

Auditor τ: `tag response`
$R_\tau \leftarrow \texttt{List}()$;
foreach $\mathbf{q}_l \in Q_\tau$ **do**
  $r_l \leftarrow \texttt{List}()$;
  foreach $\pi = 1$ *to* K **do**
    $v_{l,\pi} \leftarrow \texttt{eval}(M_\pi, \mathbf{q}_l)$;
    $\mathbf{g}_{l,\pi} \leftarrow (\texttt{eval}(M_\pi^1, \mathbf{q}_l) \ \cdots \ \texttt{eval}(M_\pi^\gamma, \mathbf{q}_l))$;
    $r_l.\texttt{append}(\texttt{tuple}(v_{l,\pi}, \mathbf{g}_{l,\pi}))$;
  end
  $R_\tau.\texttt{append}(r_l)$;
end
$\texttt{Send}(R_\tau, \text{User})$;

User: `decoding`
$\mathcal{T}_j \leftarrow \texttt{List}()$;
foreach $j_l \in S_j$ **do**
  $T_{j_l} \leftarrow 0^K$;
  foreach $\pi = 1$ *to* K **do**
    $v_{l,\pi}^\tau, \mathbf{g}_{l,\pi}^\tau \leftarrow R_\tau[l][\pi]$, for $\tau = 0, 1$;
    $\mathrm{u}_{l,\pi} \leftarrow (v_{l,\pi}^0 \ \langle \mathbf{g}_{l,\pi}^0, \mathbf{z}_l \rangle \ v_{l,\pi}^1 \ \langle \mathbf{g}_{l,\pi}^1, \mathbf{z}_l \rangle)^\mathsf{T}$;
    $T_{j_l}[\pi] \leftarrow (M^{-1}\mathrm{u}_{l,\pi})[0]$
  end
end

Matrix representation of the polynomials

When generating the responses, the TPAs calculate the values and gradients of the stored polynomials based on the received queries. We find that the TPAs needs execute $(1 + \gamma) \times K \times |S_j|$ times of function evaluation operation and $\gamma \times K \times |S_j|$ times of partial differentiation operation, and these operations are considered to be expensive. In our design, we use matrices to represent the polynomials and their partial derivatives. Specifically, polynomial

$$F_\pi(x_0, x_1, \ldots, x_{\gamma-1}) = \sum_{i=1}^{n} t_{i\pi} \prod_{j:\phi(i)[j]=1} x_j$$

is represented by matrix M_π, which satisfies

$$M_\pi[\sum_{v}^{i-1} t_{v\pi}] = \phi(i), \text{ if } t_{i\pi} = 1.$$

For a partial derivative of F_π on x_j, we adopt a matrix M_π^j to represent it. We initially set $M_\pi^j = M_\pi$ and the matrix is generated based on the following rule:

$$\text{For each } M_\pi[i]: \begin{cases} \text{Remove the row, if } M_\pi[i, \pi] = 0; \\ \text{Set } M_\pi^j[i, \pi] = 0, \text{ otherwise,} \end{cases}$$

where $M_\pi[\cdot]$ means a row of it with corresponding index, and $M_\pi[\cdot, \cdot]$ an element in the matrix. We note that the construction of the matrix representation can be done in the pre-processing once the tags are generated, and the space overheads of adopting the matrix representation is $O(nK)$.

As shown in Alg. 1, the TPA computes the values and

gradients by the eval operation, which is defined by

$$\texttt{eval}(M, \mathrm{q}) = \sum_{i}^{\texttt{dim}(M).r} \prod_{j, M[i,j]=1}^{\gamma} M[i,j]\mathrm{q}[j],$$

where $\texttt{dim}(M).r$ is the number of rows in the matrix M.

IV. ICE: Analysis

A. Correctness

Lemma 1: If the tags are correct, then the outputs of the protocol are correct.

Proof: Our goal is to show that $P = \tilde{P}$ if the edge holds the correct data blocks. Tags are generated by $\mathrm{T}_i = (g^{b_i}) \mod N$, for $b_i \in F$. Then, we have

$$\begin{aligned} R &= \prod_{k=1}^{n_i} (\tilde{\mathrm{T}}_{i,k}^{a_k} \mod N) \mod N \\ &= \prod_{k=1}^{n_i} (\mathrm{T}_{i,k}^{a_k \tilde{s}} \mod N) \mod N \\ &= \prod_{k=1}^{n_i} ((g^{m_{i,k}})^{a_k \tilde{s}} \mod N) \mod N \end{aligned}$$

Then,

$$\begin{aligned} \tilde{P} &= R^s \mod N \\ &= \prod_{k=1}^{n_i} ((g^{m_{i,k}})^{a_k \tilde{s} s} \mod N) \mod N \\ &= g^{s\tilde{s}\sum_{k=1}^{n_j} a_k m_{j,k}} \mod N \\ &= (g_s)^{\tilde{s}\sum_{k=1}^{n_j} a_k m_{j,k}} \mod N \end{aligned}$$

Recall that the edge generates the proof as

$$P = (g_s)^{\tilde{s}\sum_{k=1}^{n_j} a_k m_{j,k}} \mod N$$

We have that $P = \tilde{P}$, and this completes the proof. ∎

Lemma 2: The tag retrieval process is correct.

Proof: The detailed construction can be found in [9], [10], and here we provide a proof sketch for the sake of completeness. To prove the tag retrieval process is correct, we only need to prove that for each bit in the tags, the retrieval is correct. Let $g_\pi(t) = F_\pi(\phi(j_l) + t\mathbf{z}_l)$, then we have $g_\pi(0) = F_\pi(\phi(j_l)) = t_{j_l\pi}$. Because ϕ is a map from indexes to a Hamming space with weight 3, F_π is a polynomial with degree 3, and therefore $g_\pi(t)$ can be represented as $g_\pi(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^4$. Thus, we have $g'_\pi(t) = c_1 + 2c_2 t + c_3 t^2$.

On the other hand, we have that $g_\pi(t_0) = v^0_{l,\pi}$ and $g'_\pi(t_0) = \langle \mathbf{g}^0_{l,\pi}, \mathbf{z}_l \rangle$, and similarly for $g_\pi(t_1)$ and $g'_\pi(t_1)$. We set $t_0 = 1$ and $t_1 = 2$, and then we have

$$\mathrm{u}_{l,\pi} = M(c_0 \ c_1 \ c_2 \ c_3)^{\mathsf{T}},$$

where $\mathrm{u}_{l,\pi} = (v^0_{l,\pi} \ \langle \mathbf{g}^0_{l,\pi}, \mathbf{z}_l \rangle \ v^1_{l,\pi} \ \langle \mathbf{g}^1_{l,\pi}, \mathbf{z}_l \rangle)^{\mathsf{T}}$ and

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 1 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

Then, we have $t_{j_l\pi} = g_\pi(0) = c_0 = M^{-1}\mathrm{u}_{l,\pi}[0]$, and this completes the proof. ∎

Theorem 3: If the edge is honest, then it can pass the checking successfully.

Proof: By Lem. 1 and Lem. 2, we immediately prove the theorem. ∎

B. Security and Privacy

Before proving the security of the proposed protocol, we first review the definition of KEA1-r assumption [3], [16].

Definition 4 (KEA1-r (Knowledge of Exponent Assumption)): For any adversary $\mathcal{A}$ taking input (N, g, g^s) and returning (C, Y) with $Y = C^s$, there exists an extractor which given the same inputs as $\mathcal{A}$ returns c such that $C = g^c$.

Lemma 5 ([17]): Denote the prime factorization of n by $p_1^{v_1} \cdots p_t^{v_t}$. Let g be any function such that 1) $lcm(p_1 - 1, \cdots, p_t - 1) | g(n)$; 2) $|g(n)| = O(|n|^k)$ for some constant k. Then there exists a polynomial reduction from prime factorization to g. The prime factorization of n is solved with the cost of $O(|n|^{k+4} M(|n|))$, where $M(|n|)$ denotes the cost of multiplying two integers of binary length $|n|$.

Theorem 6: The proposed protocol is secure against the untrusted edge nodes under the KEA1-r and the large integer factorization assumptions.

Proof: Due to the limit of space, we omit the details of the proof here and leave them to the journal version. ∎

Theorem 7: Under the semi-honest model, a third party verifier cannot compute any information about the user's data from our protocol's execution.

Proof: From our protocol, we can see that the verifier has the input $(N, g, r, s, \{T_i\}_{i\in[1,n]})$ and receives $\{\widetilde{T}_i\}_{i\in S}$ from the user and $\widetilde{P}$ from the edge. So we construct a simulator $\mathcal{S}$ to imitate the verifier's view.

- $\mathcal{S}$ randomly choose a subset S' of $[1, n]$ such that $|S'| = |S|$, and $\tilde{s}' \in \mathbb{Z}_N \backslash \{0\}$. Then $\mathcal{S}$ computes $\widetilde{T}'_i = T_i^{\tilde{s}'}$, $i \in S'$.
- $\mathcal{S}$ choose (r_1, s_1) as in our protocol, sends (r_1, g^{s_1}) to the edge and gets $\widetilde{P}'$.

We get $\mathcal{S}$'s view $(N, g, r_1, s_1, \{T_i\}_{i\in[1,n]}, \{\widetilde{T}'_i\}_{i\in S'}, \widetilde{P}')$ and the verifier's view $(N, g, r, s, \{T_i\}_{i\in[1,n]}, \{\widetilde{T}_i\}_{i\in S}, \widetilde{P})$. We can easily find these two distributions are identical. ∎

Then, we show that the tag retrieval process is private in terms of the query pattern.

Theorem 8: The auditor cannot infer which tag the user queries by using Alg. 1.

Proof: For each $j_l \in S_j$, $\mathbf{z}_l$ is uniformly randomly generated from $\mathbb{F}_4^\gamma$. Thus, we have $\phi(j_l) + t_\tau \mathbf{z}_l$ is uniformly

distributed in $\mathbb{F}_4^\gamma$ for any $t_\tau \neq 0$. Therefore, for any tag query $q = \phi(\cdot) + t_\tau \mathbf{z}_l$, for any indexes j, j', we have

$$\Pr[q|j] = \Pr[q|j'].$$

Thus, the auditor cannot infer user's queried index of tag based on the user's query. ■

C. Complexity

We analyze the computation and communication complexity of the proposed protocol. The computation cost on the user side is $n_j T_{\exp} + T_{\text{prng}} + O(n^{1/3} + n_j K)$, where $T_{\exp}$ is the time of modular exponentiation over large numbers, T_{prng} is the time of generating a pseudo-random number. For the edge, the computation cost is $n_j(T_{\text{prng}} + T_{\text{add}}) + T_{\exp}$, where T_{add} is the time of addition over large numbers. The computation cost is $(n_j + 1)(T_{\text{prng}} + T_{\exp}) + n_j T_{\text{mul}} + O(n^{5/3})$ on the TPA, where T_{mul} is the time of multiplication over large numbers. The communication cost is shown in Tab. I, in which $|N|$ is the bit length of module N.

Table I
COMMUNICATION COST (BITS)

from	to		
	User	Edge	TPA
User	-	$O(1)$	$n_j\|N\| + O(n^{1/3})$
Edge	-	-	$O(1)$
TPA	$O(n_j K n^{1/3})$	$O(1)$	-

V. EXTENDED PROTOCOL FOR BATCH VERIFICATION

In the edge computing, an end device may get connected with multiple edges at the same time, and there is a need to verify data integrity on these edges at the same time. A straightforward solution is to verify each edge node separately by applying ICE-basic protocol. However, there are two major issues for apply ICE-basic protocol in this scenario: 1) the individual verification of these edges can be tedious and inefficient for the TPA; 2) more importantly, if the user uses the same secret in these individual verification, the overlapping information of subsets that have been pre-downloaded to the edges may be inferred by the TPA.

We extend the integrity checking for batch verification and propose ICE-batch protocol. Our construction is also based on HVT [3] and PIR [9], [10]. The protocol consists of 8 components, which is similar as the basic protocol. The first 5 components KeyGen, TagGen, UserSetup, TPASetup, and IndexQuery are the same as the basic protocol, except that in IndexQuery, the user query the data block indexes of a set of edges, and get a set $\mathcal{S} = \{S_1, S_2, \ldots, S_J\}$. Then, we describe other components of ICE-batch as follows:

$\mathbf{ChallengeEdge}(sk, pk) \rightarrow \mathcal{C}$. This operation generates a set of challenges $\mathcal{C} = \{\text{chal}_1, \text{chal}_2, \ldots, \text{chal}_J\}$ by the TPA and the user for requesting integrity proofs of $F_1, F_2, \ldots, F_J$ from the edges. The verifier generates a random element $s \in \mathbb{Z}_N^*$, and the user generates a set of random keys $\{e_1, e_2, \ldots, e_J\}$, where $e_j \in [1, 2^K - 1]$, for $j = 1, 2, \ldots, J$. They send each challenge $\text{chal}_j = (e_j, g_s)$ to edge_j, where $g_s = g^s \mod N$.

$\mathbf{ProofEdge}(F_j, chal_j) \rightarrow P$. Each edge also needs to compute a response to prove the integrity based on the challenge. It first generates a sequence of coefficients for the subset of blocks that it possesses based on the received random key e_j, denoted by $a_1^{(j)}, a_2^{(j)}, \ldots, a_{n_j}^{(j)}$, then the edge computes

$$P_j = (g_s)^{\sum_{k=1}^{n_j} a_k^{(j)} m_{j,k}} \mod N.$$

$\mathbf{VerifyEdge}(pk, \mathcal{T}, \mathcal{C}, \mathcal{P}, \mathcal{S}) \rightarrow \{0, 1\}$. This process consists of the following steps:

- The user computes the union of sets in $\mathcal{S}$, which is denoted by $U = \bigcup_{j=1}^{J} S_j$. Based on the set of indexes U, the user gets the corresponding set of tags $\mathcal{T}_U$ from the TPA based on the private tag retrieval that we have proposed in Sec. III-B.
- Based on each random key e_j, the user generates the coefficients $a_1^{(j)}, a_2^{(j)}, \ldots, a_{n_i}^{(j)}$. For $k \in U$, denote by $\Lambda_k = \{\text{edge}_{k_1}, \text{edge}_{k_2}, \ldots, \text{edge}_{k_{|\Lambda_k|}}\}$ the set of edges that satisfy $k \in S_j$, and its the corresponding index in S_j is $\tau(j)$. For tag $\text{T}_{U,k} \in \mathcal{T}_U$, the user calculates

$$\tilde{\text{T}}_{U,k} = \text{T}_{U,k}^{\sum_{\lambda=1}^{|\Lambda_k|} a_{\tau(j)}^{(k_\lambda)}}.$$

  And next, the user sends the regenerated set of tags $\tilde{\mathcal{T}}_U = \{\tilde{T}_{U,1}, \tilde{\text{T}}_{U,2}, \ldots, \tilde{\text{T}}_{U,|U|}\}$ to the TPA.
- The TPA computes

$$R = \prod_{k=1}^{|U|} (\tilde{\text{T}}_{U,k}) \mod N, \ \tilde{P} = R^s \mod N,$$

  and $P = \prod_{j=1}^{J} P_j \mod N$. Then it checks whether $P = \tilde{P}$. If true, return 1; otherwise, return 0.

Compared with the basic protocol, the major improvement of the extended protocol is in the VerifyEdge phase. The rationale behind the design is twofold. First of all, we have observed that the same data block may be pre-downloaded by multiple edges for the QoS awareness. However, when performing the integrity verification, the overlaps between data blocks on different edges brings unnecessary burdens for the TPA. The extended protocol only considers the union of indexes of these edges, and the TPA need not have a knowledge of which blocks a certain edge has pre-downloaded. The other idea of ICE-batch is that the communication between edges and TPAs and communication between users and TPAs are reduced at the cost of increasing the communication between users and edges. Such a tradeoff benefits from the architecture of edge computing, in which the links between user and edges are faster, and those connected to TPAs (which are usually deployed at remote servers) are slower.

Below we present the correctness of ICE-batch. The security and privacy analysis of the extended protocol is similar to the basic protocol, and we omit it here.

Theorem 9: If the edges are honest, then they can pass the checking successfully.

Proof: The proof is similar to Thm. 3, and our goal is to show that $P = \tilde{P}$ if the edges hold the correct data blocks. Due to the limit of space, we omit the details of the proof here. ■

Table II
CONFIGURATIONS OF THE EXPERIMENT ENVIRONMENTS

Entity	Experiment environments			
	Device	CPU	RAM	#thread
CSP	Server	Intel Xeon E5-2650 @ 2.00GHz × 32	64GB	Single
TPA	Server	Intel Xeon E5-2650 @ 2.00GHz × 32	64GB	Multiple
Edge	Laptop	Intel Core i7-7500U @ 2.70GHz × 4	8GB	Single
User	RasPi	ARM A53 @ 1.2GHz × 2	1GB	Single
	Laptop	Intel Core i5-5200U @ 2.20GHz × 2	8GB	Single

VI. EXPERIMENTS

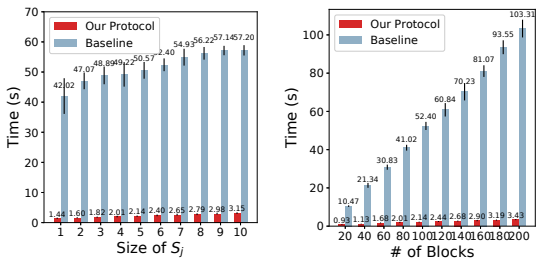
A. Setup

There are four types of entities involved in the proposed protocol: CSP, TPA, edges, and users. We have implemented a prototype to evaluate the proposed protocols, and different entities are emulated by three types of machines with different capabilities and configurations in our experiments. Specifically, the CSP and TPA are deployed on Dell PowerEdge E720 servers, the edge node is deployed on a ThinkPad T470 laptop, and the users are deployed on two types of end devices: Thinkpad T450s laptop and Raspberry Pi 3 Model B development kit. More details of the the experiment environments of these four types of entities are shown in Tab. II.

In the experiments, all the codes are written in the Python and the cryptographical operations are implemented by using the PyCrypto library [18] and gmpy2 library [19]. We choose the length of RSA module N to be 1,024 bits; the key size of the pseudo-random function f to be 1,024 bits. We set the size of data blocks ranging from 256KB (2^{21} bits) to 1024KB (2^{23} bits) in our experiments. All the results of evaluation are average results over 100 runs.

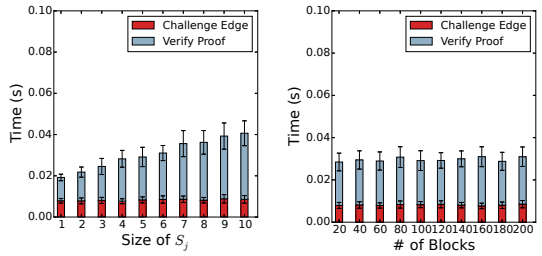
B. Computation Cost on the TPA

We first present the experiment results of the computation cost on the TPA. In the proposed protocol, the major computation cost is contributed by the tag response and integrity checking parts. In addition, we find that the execution time of the protocol mainly depends on the size of the subset of the data blocks that need to be checked and the total number of data blocks in a file. In this set of experiments, we evaluate the tag response time and integrity checking time by varying the size of S_j and the number of data blocks, respectively.



(a) varying the size of S_j (b) varying n

Figure 2. Computation cost on the TPA: Tag Response



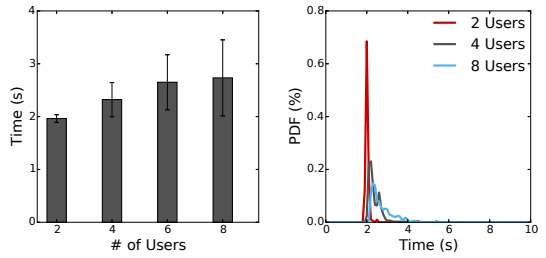
(a) varying the size of S_j (b) varying n

Figure 3. Computation cost on the TPA: Integrity Checking

Fig. 2 shows the computation cost of the tag response on the TPA. We compare our protocol with a micro benchmark, which is a tag response process without the matrix representation of the polynomials. We can find that the proposed matrix representation can significantly reduce the computation cost in the tag response process. Fig. 2a shows that the computation time increases with the size of S_j increasing, and due to the adoption of multi-thread optimization on the TPA, the time does not increase linearly. In our protocol, the time is about 1 to 3 seconds for the cases where the size of S_j ranging from 1 to 10. We can also find that the computation time increases with the number of data blocks in Fig. 2b.

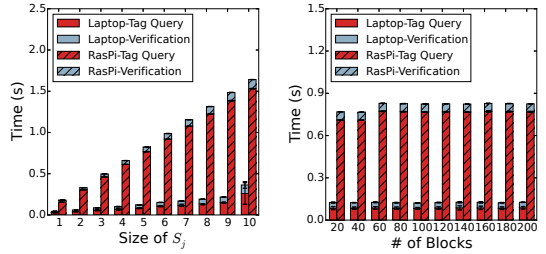
Fig. 3 shows the integrity checking time on the TPA, which mainly consists of two parts: 1) the TPA generates a challenge for the edge; 2) the TPA verifies the proof generated by the edge. We can find that the time of challenging the edge varies very little, and the time of verifying the proof increases with the size of S_j. The protocol is very efficient for the integrity checking, and the time is no more than 0.05 seconds (50ms) in our experiments.

We also evaluate the effect of the number of users on the TPA's performance. From Fig. 4a, we can observe that, the computation time increases slowly with the number of users increasing. An interesting finding is that the fluctuation of the time gets larger with there are more users, and there is



(a) total time (b) probability density function

Figure 4. Computation cost on the TPA: multi-user scenario



(a) varying the size of S_j (b) varying n

Figure 5. Computation cost on the end-user devices

a long tail of the time distribution when the number of user is large (as shown in Fig. 4b). The phenomenon may be caused by the parallel optimization, and we suppose it can be mitigated by a multi-user-aware design, which might be an interesting direction of the future works.

C. Computation Cost on the End Devices

The experiments are conducted on two types of devices: laptop and Raspberry Pi. Fig. 5a and Fig. 5b show the results with varying the size of S_j and the number of blocks, respectively.

We can find that the time increases with the size of S_j and varies very little with the number of data blocks. For the laptop, the protocol is very efficient. The tag query process takes at most 0.26 seconds, and the verification process takes about 0.10 seconds when the size S_j is less than 10. Therefore, the total computation cost on the user side is less than 0.40 seconds on the laptops in our experiments.

For the Raspberry Pi, which has much lower computation capability, the tag query process takes from 0.17 to 1.53 seconds when the size of S_j varies from 1 to 10 for encoding the tag queries and decoding the tag responses. It takes from 0.01 to 0.11 seconds with varying the size of S_j for verification. In total, the computation cost is at most 1.63 seconds on the Raspberry Pi devices when the size of S_j is less than 10.

D. Computation Cost on Edges

The major computation cost on the edge node is contributed by the process of generating proofs that show it holds the correct data. By the analysis, we suppose that the execution time mainly depends on the size of S_j and the size of each data block. In the experiments, we vary the size of S_j from 1 to 10, and evaluate the computation cost on data blocks with 256KB, 512KB, and 1024KB, respectively. In Fig. 6, we can find that the computation time varies very little with the change of S_j's size. We think the reason is that the costs of the modular addition and modular multiplication are negligible to the cost of modular exponentiation operations, and when generating a proof, only one modular exponentiation operation is performed, regardless the size of S_j.

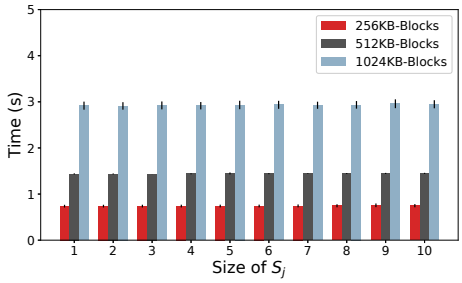


Figure 6. Computational cost on the edges

Another observation is that the computation cost increases with the increase of data block size. For 256KB blocks, the computation time varies from 0.74 to 0.76 seconds; for 512KB blocks, the edge takes about 2x time, varying from 1.43 to 1.45 seconds; and for 1024KB blocks, the computation time varies from 2.91 to 2.96 seconds (about 4x comparing with the 256KB blocks). Roughly, the computation time increases linearly with the size of data blocks.

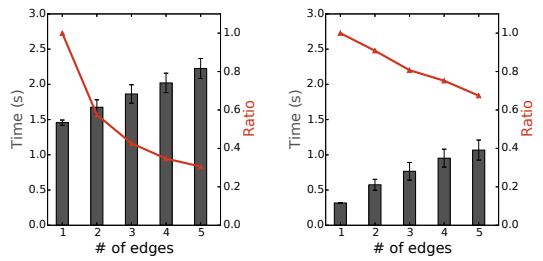
E. Evaluation on the Extended Protocol

We want to find out to what degree the extended protocol reduces the computation cost on both TPAs and end devices. In this set of experiments, we set $n = 100$ and assume that each edge pre-downloads 3 data blocks from a set of 10 data blocks. In Fig. 7, we can find that the computation time on the TPA and the Raspberry Pi increases moderately with the number of edges increasing.

We also adopt the ratio $\frac{\texttt{time}(\text{ICE-batch})}{\texttt{time}(\text{ICE-basic}) \times |\mathcal{S}|}$ to evaluate the protocol, where $|\mathcal{S}|$ is the number of edges, and obviously $\texttt{time}(\text{ICE-basic}) \times |\mathcal{S}|$ is the time that ICE-basic performs verification on $\mathcal{S}$. We can observe that the ratio (red line in Fig. 7) decreases with the number of edges increasing, because the overlapping data blocks get more when there more edges pre-download from a certain set of data blocks. Fig. 8 also shows the communication cost of the extended protocol, in which we can see a similar decrease of ratio.

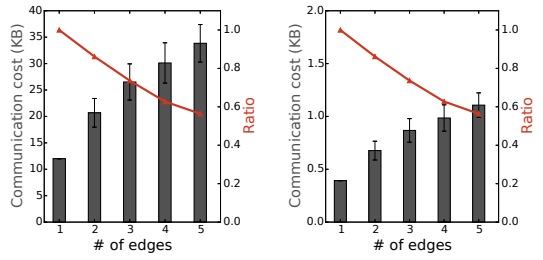
F. Preprocess Time

Tab. III shows the time for setting up the protocol. The preprocess on the end device side consists of three parts: key



(a) TPA (b) End Device (RasPi)

Figure 7. Computation cost of the extended protocol.



(a) TPA ⇒ RasPi (b) RasPi ⇒ TPA

Figure 8. Communication cost of the extended protocol.

generation, tag generation, and setup of the tag query. We can find that the key generation time is about 3.10 and 0.03 seconds for the Raspberry Pi and laptop, respectively. The time of generating tags ranges from 0.38 to 3.82 seconds for the Raspberry Pi in our experiments, and the laptop is about 15x faster than it. In our experiments, we find that the time of setting up the tag query is less than 10^{-3} seconds for these two types of end devices, which is negligible to others. For the TPAs, the major preprocess is for the tag response process, which costs less than 3 seconds when $n \leq 200$.

VII. Related Works

A. Mobile Edge Computing

Mobile edge computing and similar concepts,*e.g.*, fog computing [20], cloudlet [20], have drawn many attentions in the recent years. A bunch of surveys or position papers, *e.g.*, [21], [22] , have pointed out that edge computing is a promising technology that can dramatically facilitate the internet of things (IoT) and mobile cloud computing (MCC). Previous works on the mobile edge computing mainly focus on the architecture design of the mobile edge computing (*e.g.*, [23], [24]), the applications on it (*e.g.*, [25], [26]), and the resource configuration for it (*e.g.*, [27]–[29]).

Recently, the security problems in the mobile edge computing have also drawn some attention, *e.g.*, [30], [31]. In [31], Bhardwaj et al. present a scheme for protecting the integrity of programs on the edges by adopting trusted execution environments.

Table III
Preprocess time (s)

n		40	80	120	160	200
User RasPi	KeyGen	3.10				
	TagGen	0.76	1.53	2.30	3.06	3.82
User Laptop	KeyGen	0.03				
	TagGen	0.05	0.10	0.16	0.20	0.26
TPA	TPASetup	1.25	1.59	1.86	2.08	2.32

B. Integrity Verification

Integrity is one of the most important properties for data security. The most related works of our paper is the integrity verification of remote stored data, which has drawn much attention in the last decade with the rapid development and popularization of cloud technology.

The problem of checking integrity of remote data is first introduced by in [32], which proposes a RSA-based method for this problem. Proof of retrievability (POR) [2] and provable data possession (PDP) [3] are two most famous definitions of remote data integrity checking. Based on these two definitions, lots of works have been conducted for remote data integrity checking with various features, *e.g.*, data dynamics [7], [33], public verifiability [6], [7], and privacy preservation [6], [7], [34]. Previous works on remote data integrity checking may not apply well for mobile edge computing, because the architecture has significantly changed and a new type of entities, *i.e.*, edge, is introduced to the system. In this work, we address the data integrity checking problem in the mobile edge computing, and propose practical solutions for it based on the concept of PDP [3], [6].

VIII. Concluding Remarks

In this paper, we study the data integrity verification problem in mobile edge computing. We have proposed two protocols, which are suitable for the cases where the users want to verify the data integrity on a single edge or multiple edges, respectively. The proposed protocols are rigorously proved to be correct, secure against the untrusted edges, and privacy-preserving against the their-party auditors. A proof-of-the-concept system is implemented for evaluating the protocols, and experimental results show that the proposed protocols are efficient.

Acknowledgment

The authors would like to thank all the reviewers for their helpful comments. This work was supported in part by NSFC-61425024, NSFC-61872176, NSFC-61402223, the Jiangsu Province Double Innovation Talent Program, and NSFC-61321491. F. Xu was supported in part by NSFC-61872180, Microsoft Research Asia, CCF-NSFOCUS "Kunpeng" Research Fund, and Alipay Research Fund. Q. Li was supported in part by NSF grant CNS-1816399.

REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] A. Juels and B. S. K. Jr., "Pors: proofs of retrievability for large files," in *Proc. ACM CCS'07*, 2007, pp. 584–597.

[3] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. X. Song, "Provable data possession at untrusted stores," in *Proc. ACM CCS'07*, 2007, pp. 598–609.

[4] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring data storage security in cloud computing," in *IEEE IWQoS'09*, 2009.

[5] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *ESORICS'09*, ser. Lecture Notes in Computer Science, vol. 5789. Springer, 2009, pp. 355–370.

[6] Z. Hao, S. Zhong, and N. Yu, "A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 9, pp. 1432–1437, 2011.

[7] J. Li, L. Zhang, J. K. Liu, H. Qian, and Z. Dong, "Privacy-preserving public auditing protocol for low-performance end devices in cloud," *IEEE Trans. Information Forensics and Security*, vol. 11, no. 11, pp. 2572–2583, 2016.

[8] F. Sebé, J. Domingo-Ferrer, A. Martínez-Ballesté, Y. Deswarte, and J. Quisquater, "Efficient remote data possession checking in critical information infrastructures," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 8, pp. 1034–1038, 2008.

[9] D. P. Woodruff and S. Yekhanin, "A geometric approach to information-theoretic private information retrieval," *SIAM J. Comput.*, vol. 37, no. 4, pp. 1046–1056, 2007.

[10] Z. Dvir and S. Gopi, "2-serverr pir with subpolynomial communication," *J. ACM*, vol. 63, no. 4, pp. 39:1–39:15, Sep. 2016.

[11] Z. Hao and Q. Li, "Poster abstract: Edgestore: Integrating edge computing into cloud-based storage systems," in *IEEE/ACM Symposium on Edge Computing, SEC'16*, 2016, pp. 115–116.

[12] U. Drolia, K. Guo, R. Gandhi, and P. Narasimhan, "Poster abstract: Edge-caches for vision applications," in *IEEE/ACM Symposium on Edge Computing, SEC'16*, 2016, pp. 91–92.

[13] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *Proc. IEEE ICDCS'17*, K. Lee and L. Liu, Eds., 2017, pp. 276–286.

[14] E. Kushilevitz and R. Ostrovsky, "Replication is NOT needed: SINGLE database, computationally-private information retrieval," in *Proc. FOCS '97*, 1997, pp. 364–373.

[15] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. E. S. III, "Public key encryption that allows PIR queries," in *Proc. CRYPTO'07*, ser. Lecture Notes in Computer Science, vol. 4622. Springer, 2007, pp. 50–67.

[16] M. Bellare and A. Palacio, "The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols," in *Proc. CRYPTO'04*, ser. Lecture Notes in Computer Science, vol. 3152. Springer, 2004, pp. 273–289.

[17] G. L. Miller, "Riemann's hypothesis and tests for primality," *Journal of computer and system sciences*, vol. 13, no. 3, pp. 300–317, 1976.

[18] D. C. Litzenberger, "PyCrypto-the python cryptography toolkit," *URL: https://www. dlitz. net/software/pycrypto*, 2016.

[19] C. V. Horsen, "Gmpy2: Mupltiple-precision arithmetic for python," 2016.

[20] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[21] P. G. López, A. Montresor, D. H. J. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. P. Barcellos, P. Felber, and E. Rivière, "Edge-centric computing: Vision and challenges," *Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[22] S. Yi, Z. Qin, and Q. Li, "Security and privacy issues of fog computing: A survey," in *WASA'15*, vol. 9204. Springer, 2015, pp. 685–695.

[23] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust, "Mobile-edge computing architecture: The role of MEC in the internet of things," *IEEE Consumer Electronics Magazine*, vol. 5, no. 4, pp. 84–91, 2016.

[24] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. IEEE INFOCOM'16*, 2016.

[25] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE ICDCS'17*, 2017, pp. 328–339.

[26] P. Hao, Y. Bai, X. Zhang, and Y. Zhang, "Edgecourier: an edge-hosted personal service for low-bandwidth document synchronization in mobile cloud storage services," in *SEC*. ACM, 2017, pp. 7:1–7:14.

[27] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *Proc. IEEE INFOCOM'17*, 2017.

[28] Y. Huang, X. Song, F. Ye, Y. Yang, and X. Li, "Fair caching algorithms for peer data sharing in pervasive edge computing environments," in *Proc. IEEE ICDCS'17*, 2017, pp. 605–614.

[29] T. Bahreini and D. Grosu, "Efficient placement of multi-component applications in edge computing systems," in *SEC*. ACM, 2017, pp. 5:1–5:11.

[30] S. Echeverría, D. Klinedinst, K. Williams, and G. A. Lewis, "Establishing trusted identities in disconnected edge environments," in *IEEE/ACM Symposium on Edge Computing, SEC'16*, 2016, pp. 51–63.

[31] K. Bhardwaj, M. Shih, P. Agarwal, A. Gavrilovska, T. Kim, and K. Schwan, "Fast, scalable and secure onloading of edge functions using airbox," in *IEEE/ACM Symposium on Edge Computing, SEC'16*, 2016, pp. 14–27.

[32] Y. Deswarte, J. Quisquater, and A. Saïdane, "Remote integrity checking - how to trust files stored on untrusted servers," in *Integrity and Internal Control in Information Systems VI - IFIP TC11/WG11.5 Sixth Working Conference on Integrity and Internal Control in Information Systems (IICIS)*, ser. IFIP, vol. 140. Springer, 2003, pp. 1–11.

[33] E. Shi, E. Stefanov, and C. Papamanthou, "Practical dynamic proofs of retrievability," in *Proc. ACM CCS'13*, 2013, pp. 325–336.

[34] Y. Yu, M. H. Au, G. Ateniese, X. Huang, W. Susilo, Y. Dai, and G. Min, "Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage," *IEEE Trans. Information Forensics and Security*, vol. 12, no. 4, pp. 767–778, 2017.