Efficient Verification of Edge Data Integrity in Edge Computing Environment

Guangming Cui, Qiang He, *Senior Member, IEEE*, Bo Li, Xiaoyu Xia, Feifei Chen, *Member, IEEE*, Hai Jin, *Fellow, IEEE*, Yang Xiang, *Fellow, IEEE*, and Yun Yang, *Senior Member, IEEE*

Abstract—The new edge computing paradigm extends cloud computing by allowing service vendors to deploy their service instances and data on distributed edge servers to serve their service users in close geographic proximity to those edge servers. Caching edge data on edge servers profoundly reduces the retrieval latency perceived by users. However, these edge data are subject to corruption due to intentional and/or accidental exceptions. This is a major challenge for service vendors but has been overlooked. Thus, verifying the integrity of edge data accurately and efficiently is a critical security problem in the edge computing environment. A unique characteristic of the edge computing environment is that edge servers suffer from constrained computing capacities. Thus, verifying data integrity on massive edge servers individually is computationally expensive and impractical. In this paper, we tackle this Edge Data Integrity (EDI) problem with an inspection and corruption localization scheme for EDI named ICL-EDI. This scheme allows service vendors to inspect data integrity and localize corrupted edge data cached on multiple edge servers accurately and efficiently. To evaluate its performance, we implement ICL-EDI and conduct extensive experiments to demonstrate its effectiveness and efficiency.

Index Terms—Edge data integrity, edge computing, service vendor, privacy-protection

1 Introduction

Cloud computing has achieved tremendous success in the past decade by providing various users with flexible and reliable computing and storage capacities. As mobile computing becomes increasingly popular, a variety of latency-sensitive apps are proliferating, such as face recognition, natural language processing and interactive gaming [1]. However, these latency-sensitive apps often suffer from high and unpredictable latency between end-users and cloud servers. In recent years, edge computing has been widely acknowledged as the key to addressing this latency that is fundamentally challenging the cloud computing paradigm [2]. It is a key 5G enabler technology that facilitates the 5G mobile network by deploying edge servers with computing and storage resources at base stations [3]. Service vendors like YouTube and Uber can cache data within end-users' geographic proximity to offer low-latency services to service users [4], [5], [6].

- *Guangming Cui, Qiang He, Bo Li, Yang Xiang, and Yun Yang are with the School of Science, Computing and Engineering Technologies, Swinburne University of Technology, Hawthorn, VIC 3122, Australia. E-mail: {gcui, qhe, boli, yxiang, yyang}@swin.edu.au.*
- *Xiaoyu Xia and Feifei Chen are with the School of Information Technology, Deakin University, Geelong, VIC 3220, Australia. E-mail: {xiaoyu.xia, feifei.chen}@deakin.edu.au.*
- *Hai Jin is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: hjin@hust.edu.cn.*

Manuscript received 2 April 2020; revised 11 April 2021; accepted 3 June 2021. Date of publication 17 June 2021; date of current version 9 December 2022. This work was supported by Australian Research Council Discovery Projects under Grants DP180100212 and DP200102491. (Corresponding author: Qiang He.) Digital Object Identifier no. 10.1109/TSC.2021.3090173

However, in the edge computing environment, it is difficult for service vendors to fully control their data cached on distributed edge servers as they no longer possess these data locally [7]. Similar to the reasons given in [8], edge servers must not be assumed reliable or trustworthy in the highly-distributed edge computing environment that are subject to accidental and/or intentional software and/or hardware exceptions. In the edge computing environment, this problem is further and significantly complicated. *From the service vendors' perspective, the integrity of their edge data is challenged by the unique characteristics of edge computing, which render existing approaches for cloud data integrity obsolete.* First, edge servers often suffer from limited computing capacities, while most approaches for ensuring cloud data integrity have assumed and leveraged the virtually unlimited computing capacities of cloud servers [8], [9]. Second, a service vendor usually caches its data - especially popular ones - on edge servers in a particular area for serving nearby service users. It is impractical for the service vendor to inspect these edge data one by one to localize the corrupted ones due to excessive computation and bandwidth consumption. Existing approaches for ensuring data security in the cloud, e.g., provable data possession (PDP) [8] and many of its variants [9], [10], are designed for thin-client users to raise data integrity challenge requests to powerful cloud storage providers like AWS or Google. In the edge computing scenario, it is the service vendors that challenge the thin-client (compared to cloud servers) edge servers for the integrity of their cached edge data. *This fundamentally differentiates the edge data integrity (EDI) problem from the PDP problem [8].*

To attack the EDI problem, this paper proposes a new scheme named ICL-EDI to help service vendors localize corrupted edge data replicas through data inspection. ICL-EDI

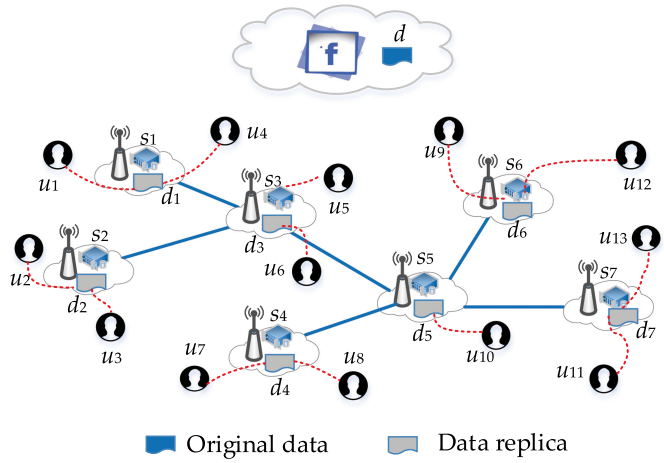


Fig. 1. Edge data integrity for facebook horizon.

leverages homomorphic tags to allow service vendors to verify the integrity of the multiple edge data replicas simultaneously. In addition, it employs the sampling technique with a binary search to significantly lower the computational overheads for both service vendors and edge servers.

The key contributions of this paper are as follows:

- It studies the novel EDI problem, aiming to help service vendors ensure edge data integrity by localizing corrupted edge data replicas accurately and efficiently.
- We propose ICL-EDI, an inspection and corruption localization scheme for EDI, and theoretically prove its correctness and probabilistic integrity guarantees.
- We implement ICL-EDI and experimentally evaluate the ability of ICL-EDI to localize corrupted edge data replicas among a vast number of edge data replicas efficiently.

The paper is structured as follows. We motivate the EDI problem in Section 2 and present ICL-EDI in Section 3. Then, we theoretically and experimentally evaluate ICL-EDI in Section 4. The related work is reviewed in Section 5. Finally, We conclude this paper in Section 6.

2 Motivating Example

Facebook Horizon[1], a virtual reality (VR) service, is a typical latency-sensitive application that may benefit significantly from caching data on edge servers. Fig. 1 shows a geographic area involving seven edge servers, $s_1, \ldots, s_7$. Facebook caches one replica of a viral VR video d on each of these edge servers. These replicas are denoted as $d_1, \ldots, d_7$. The 13 Facebook Horizon users in the area, i.e., $u_1, \ldots, u_{13}$ can access these video replicas from edge servers with low latency rather than from the remote cloud. To ensure that these edge video replicas are not corrupted, Facebook must verify their integrity remotely without having to download them from the edge servers. Based on the PDP scheme [8], Facebook can challenge each edge server individually on the integrity of $d_1, \ldots, d_7$. When a challenge request is received, edge server s_i $(1 \leq i \leq n)$ must calculate an integrity proof poi_i and send it to Facebook. Facebook can then inspect the integrity proof poi_i to confirm the integrity of d_i.

1. https://www.oculus.com/facebook-horizon/

This process must be performed seven times, one for each of the seven edge video replicas. This also applies to other geographic areas where replicas of d are cached on edge servers. Verifying edge data integrity in this way is computationally expensive and will easily become impractical when the number of edge video replicas scales up. In addition, the verification also incurs computational overheads to edge servers by asking them to calculate massive integrity proofs. Due to the limited physical sizes of edge servers, the computing resources on edge servers are highly limited [11]. Serving Facebook as well as many other service vendors, e.g., Twitter, Uber, Youtube, etc., edge servers cannot afford the heavy computation burden caused by producing massive integrity proofs.

Conventional cloud data integrity schemes, e.g., PDP [8], as well as many of its variants proposed in the last decade, place heavy computation burdens on both service vendors and edge servers. A new and lightweight scheme must be designed specifically to enable the integrity verification for a potentially huge number of edge data replicas.

3 ICL-EDI

In this section, we first overview ICL-EDI, and the present the algorithms facilitating ICL-EDI.

3.1 ICL-EDI Overview

In the past decade, PDP [8] has been intensively studied, which allows users to remotely verify their data stored on cloud servers [8], [9]. Unfortunately, the PDP scheme and its variants are not suitable for EDI.

As discussed in Section 2, a service vendor often caches data replicas on edge servers geographically distributed in an area to serve its users in the area. The number of edge data replicas may be very large, e.g., when the data is a VR video that goes viral on Facebook Horizon. Based on conventional PDP schemes, a service vendor must challenge such a vast number of edge servers on the integrity of the edge data replicas and verify the returned proofs of integrity individually. In this way, the service vendor can localize and update corrupted edge data replicas one by one. However, heavy computational overheads are incurred by the verification on both the edge servers and the service vendor. In particular, under conventional PDP schemes, thin-client users raise challenging requests for cloud storage providers to answer. Cloud storage providers like Amazon and Google have virtually unlimited computing resources and can afford the high computational overheads incurred by answering challenge requests. In the context of EDI, it is not the case. In the edge computing environment, edge servers suffer from limited computing resources due to their limited physical sizes [12], [13], [14]. Thus, *an efficient EDI scheme must not pose high computational overheads on edge servers. This is the first design objective of ICL-EDI.*

Conventional PDP schemes are designed to allow the data owner to challenge only a very small number of cloud storage providers - one in most cases - on the integrity of its data. Under conventional PDP schemes, a service vendor - the data owner in the context of EDI - must inspect the individual proofs of integrity returned by the edge servers one after another. Inspecting a vast number of proofs of integrity

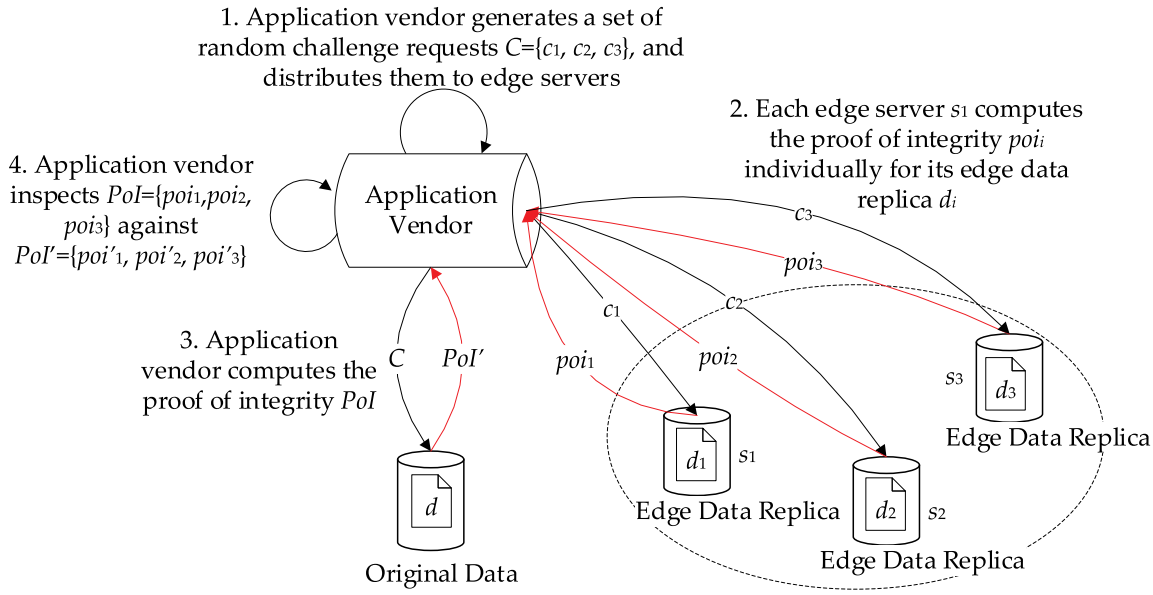


Fig. 2. Overview of ICL-EDI.

incurs excessive computational overheads. The service vendor must pay for the computing resources hired in the cloud to accommodate the computational overheads. Thus, in the context of EDI, conventional PDP schemes will incur massive computational overheads and are therefore impractical. Thus, *an efficient EDI scheme must allow a service vendor to verify the integrity of a potentially large number of edge data replicas with low computational overheads. This is the second design objective of ICL-EDI.*

To achieve the above two design objectives, ICL-EDI employs a request-response (or challenge-answer) model. To ensure the integrity of d, the service vendor sends a set of challenge requests to the edge servers, regularly or on-demand. The edge servers must respond to the received challenge requests properly to prove that their edge data replicas are not corrupted. Based on edge servers' responses, the service vendor can inspect whether its edge data replicas are corrupted - *inspection* - and if yes, on which edge server(s) - *corruption localization*.

Fig. 2 illustrates ICL-EDI in general. As demonstrated, the service vendor goes through 4 steps to challenge the edge servers $s_1, \ldots s_n$ on the integrity of the corresponding edge data replicas $d_1, \ldots, d_n$. In Step 1, as the challenger, it initiates the challenge by generating a set of challenge requests $C = \{c_1, \ldots, c_n\}$. Then, it sends these challenge requests to the edge servers, one to each edge server. In Step 2, each edge server $s_i (1 \leq i \leq n)$, as a defender, answers challenge request c_i by computing the proof of integrity for its edge data replica d_i, denoted as poi_i. Edge servers must answer the challenge requests and they do so in parallel in order to minimize the time span of the entire verification. If an edge server fails or does not respond to the challenge, it is considered that the edge data replica cached on this edge server is corrupted. The proofs of integrity produced by the edge servers, denoted by $PoI = \{poi_1, \ldots, poi_n\}$, are returned to the service vendor. In Step 3, the service vendor generates the correct proofs of integrity, denoted by PoI' based on the original data d in its own cache. Similar to [8], we assume in this study that the original data in the data owner's - the service vendor's in the context of EDI - own storage is safe and uncorrupted. In Step 4, based on the original data d in its local storage, the service vendor inspects whether any edge data replicas are corrupted and, if yes, localizes corrupted edge data replicas, i.e., finds out the edge server(s) whose edge data replicas are corrupted. After the verification, the service vendors can take corresponding actions, e.g., update the corrupted edge data replicas.

3.2 ICL-EDI Algorithms

ICL-EDI employs homomorphic tags as verification metadata based on the concept of homomorphic signatures [15]. Thus, we first introduce the concept of *homomorphic tag*:

Definition 1 (Homomorphic Tag). *Given a block b_i^j of edge data replica d_i, T_i^j the tag of b_i^j. Given two tags T_i^j and $T_i^{j'}$, if they can be combined into a value $T_i^{j \oplus j'}$ corresponding to the sum of $b_i^j \oplus b_i^{j'}$, these tags are called homomorphic tags.*

ICL-EDI employs five algorithms during its 4 steps illustrated in Fig. 2 to achieve the two design objectives discussed in Section 3.1.

3.2.1 KeyGen

In *Step 1*, the service vendor runs *KeyGen* to generate a public key pk based on a secret key sk to facilitate the inspection of edge data replica d_i. Using pk, the service vendor and each edge server $s_i \in S$ run *TagGen* to generate a set of homomorphic tags $T_i = \{T_i^1, \ldots, T_i^m\}$ for edge data replica d_i. Algorithms *KeyGen* and *TagGen* are implemented as follows:

KeyGen. $1^k \rightarrow (pk, sk)$. Given a parameter k and two large primes p and q such that $p = 2p' + 1$, $q = 2q' + 1$, where p' and q' are also primes. Then, let $N = pq$ be an RSA module, and g be a generator which satisfies $g = \delta^2$, where $\gcd(\delta \pm 1, N) = 1$. Next, public key $pk = (N, g)$ and secret key $sk = (p, q)$ are calculated. The pseudocode of *KeyGen* is presented

in Algorithm 1. It can be easily seen that the time complexity of *KeyGen* is $O(1)$.

Algorithm 1. KeyGen

Require: parameter k
Ensure: pair of public and secret keys (pk, sk)
1: Find two large primes p and q s.t. $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are primes as well.
2: $N \leftarrow pq$
3: $g \leftarrow \delta^2$ subject to $\gcd(\delta \pm 1, N) = 1$
4: Public key: $pk \leftarrow (N, g)$
5: Secret key: $sk \leftarrow (p, q)$
6: **return** (pk, sk)

3.2.2 TagGen

TagGen. $(pk, d_i) \rightarrow T_i$. Given a public key pk, generator g and an edge data replica d_i, *TagGen* generates a homomorphic tag T_i^j for each block b_i^j of d_i: $T_i^j = (g^{b_i^j}) \bmod N$. The output of *TagGen* is a set of homomorphic tags, denoted by $T_i = \{T_i^1, \ldots, T_i^m\}$, one for each block of data d_i. The pseudocode of *TagGen* is presented in Algorithm 2. The time complexity of *TagGen* is $O(m)$ for each edge data replica d_i, where m is the number of blocks.

Algorithm 2. TagGen

Require: public key pk and an edge data replica d_i
Ensure: set of homomorphic tags T_i for edge data replica d_i
1: $T_i^j \leftarrow (g^{b_i^j}) \bmod N$
2: $T_i \leftarrow \{T_i^1, \ldots, T_i^m\}$
3: **return** T_i for edge data replica d_i

For example, given an edge data replica d_i that contains three blocks ($m = 3$), i.e., $d_i = \{b_i^1, b_i^2, b_i^3\} = \{1, 2, 4\}$ and $pk = (N, g) = (+\infty, 10)$, *TagGen* will generate a set of homomorphic tags $T_i = \{10^1, 10^2, 10^4\}$ from $\{b_i^1, b_i^2, b_i^3\}$.

3.2.3 ChalEdge

In *Step 2*, to challenge edge servers $s_i, i = 1, \ldots, n$, on the integrity of edge data replica $d_i, i = 1, \ldots, n$, the service vendor first generates a set of challenge requests $C = \{c_1, .., c_n\}$, one for each edge data replica by running *ChalEdge* with public key pk and random key rk_i as inputs. Then, it sends $c_1, \ldots, c_n$ to the corresponding edge server $s_1, \ldots, s_n$. To answer challenge request c_i, edge server s_i produces the proof of integrity poi_i for d_i by running *ProGen*, and sends poi_i back to the service vendor. Algorithms *ChalEdge* and *ProGen* are implemented as follows:

Algorithm 3. ChalEdge

Require: public key pk
Ensure: set of challenges $C = \{c_1, \ldots, c_n\}$ for all edge data
1: **for all** $i \leq n$ **do**
2: Compute a random key $rk_i \in [1, 2^k - 1]$ with $\mathcal{F}$
3: $c_i \leftarrow (rk_i)$
4: $C \leftarrow C \bigcup \{c_i\}$
5: **end for**
6: **return** C

ChalEdge. $(pk) \rightarrow c_i$. *ChalEdge* runs a pseudorandom function $\mathcal{F}$ to generate a random key $rk_i \in [1, 2^k - 1]$, similar to [8], [9], [16]. It then sends a challenge request $c_i = (rk_i)$ to edge server s_i to verify the integrity of edge data replica d_i. The pseudocode of *ChalEdge* is presented in Algorithm 3. Its time complexity is $O(n)$, where n is the number of edge data replicas.

3.2.4 ProGen

ProGen. $(pk, c_i, d_i) \rightarrow poi_i$. To answer challenge request c_i, edge server s_i first samples l blocks from its edge data replica d_i according to a pseudorandom permutation generated based on the received random key rk_i, denoted by $\{b_i^{a_i^1}, \ldots, b_i^{a_i^l}\}$. Let $A_i = \{a_i^1, \ldots, a_i^l\}$ denote the set of indexes of the l sampled blocks. Then, s_i generates l homomorphic tags from the sampled data blocks based on the public key pk, denoted by $T_i^{a_i^1}, \ldots, T_i^{a_i^l}$. Next, it calculates the proof of edge data replica d_i:

$$poi_i = g^{\sum_{a_i^j \in A_i} b_i^{a_i^j}} \bmod N \tag{1}$$

The pseudocode of *ProGen* is presented in Algorithm 4. Each edge server runs Algorithm 4 to compute the proof of integrity for its edge data replica based on a set of sample blocks. The time complexity of computing proof of integrity is $O(l)$, where l is the number of sampled data blocks. Usually, a larger l allows a higher verification accuracy. However, ICL-EDI does not require a large l to achieve a high verification accuracy. As reported in Section 4, ICL-EDI achieves a 99.3% accuracy by sampling only 450 out of the 450,00 blocks of data d, a sampling rate of only 1.00%. *ProGen* is the main algorithm used by the edge servers. Its low time complexity ensures that the ICL-EDI would not pose high computational overheads on edge servers. *This allows ICL-EDI to achieve the first design objective discussed in Section 3.1* by minimizing the computational overheads incurred to edge servers during the verification process.

Algorithm 4. ProGen

Require: a challenge request c_i and edge data replica d_i
Ensure: a proof of edge data replica d_i
1: Compute l samples based on rk_i: $\{b_i^{a_i^1}, \ldots, b_i^{a_i^l}\}$
2: Compute l tags based on pk: $T_i = \{T_i^{a_i^1}, \ldots, T_i^{a_i^l}\}$
3: $poi_i \leftarrow g^{\sum_{a_i^j \in A_i} b_i^{a_i^j}} \bmod N$
4: **return** poi_i

For example, given three blocks $b_i^1 = 1$, $b_i^2 = 2$ and $b_i^3 = 4$ sampled from edge data replica d_i, *ProGen* will generate a proof of integrity for d_i: $poi_i = 10^{1+2+4} = 10^7$. Next, this proof will be sent to the service vendor to be inspected.

In *Step 3*, the service vendor first runs the same *ProGen* algorithm as the edge servers on data d in its local storage to produce the correct proofs of verification, denoted as $PoI' = \{poi_1', \ldots, poi_n'\}$. Then, in *Step 4*, it compares the proofs of integrity $PoI = \{poi_1, \ldots, poi_n\}$ received from edge servers $s_1, \ldots, s_n$ against correct proofs of integrity PoI' to localize the corrupted edge data replica(s). The service goes through both Step 3 and Step 4 locally. Thus, they are both included in *ProGen*, which is implemented as follows:

3.2.5 CheckPro

CheckPro. $(pk, C, PoI, d) \rightarrow \{\phi_1, \phi_2, \ldots, \phi_n\}$, where $\phi_i = 1, 1 \leq i \leq n$ indicates that edge data replica d_i is corrupted, and 0 otherwise. *CheckPro* employs binary search in the process below to rapidly localize the corrupted edge data replicas:

1) Given two sets of proofs of integrity, i.e., PoI received from the edge servers and PoI' calculated based on the service vendor's original data d, the service vendor calculates the inspection results, i.e., R for PoI and R' for PoI', where R is calculated as follows:

$$R = \left(\prod_{i=1}^{n} (poi_i \bmod N) \right) \bmod N \tag{2}$$

2) The service vendor then checks R and R', where R' is calculated as follows:

$$R' = \left(g^{\sum_{i=1}^{n} \sum_{a_i^j \in A_i} b_i^{a_i^j}} \bmod N) \right) \bmod N \tag{3}$$

If $R = R'$, the integrity of edge data is proved by PoI. Otherwise, PoI is divided in half, denoted as $PoI = PoI_- \bigcup PoI_+$.

3) If PoI indicates corrupted edge data replicas, PoI_- and PoI_+ are checked individually to localize the corrupted edge data replicas by repeating steps (1)-(3) until all the corrupted edge data replicas are localized.

Algorithm 5. CheckPro

Require: public key pk, all proofs of integrity PoI and an original data d
Ensure: inspection results
1: Initialize the inspection and corruption localization result $\Phi = \{\phi_1, \ldots, \phi_n\}, \forall \phi_i = 0$
2: Compute the tags from original data d based on pk
3: Compute the proofs of integrity PoI' based on original data d with **ProGen**
4: **repeat**
5: $R \leftarrow (\prod_{poi_i \in PoI} (poi_i \bmod N)) \bmod N$
6: $R' \leftarrow (\prod_{poi_i' \in PoI'} (poi_i' \bmod N)) \bmod N$
7: **if** $R = R'$ **then**
8: **for all** $poi_i \in \Phi(PoI)$ **do**
9: $\phi_i \leftarrow 1$
10: **end for**
11: **return** $\Phi(PoI)$
12: **else**
13: Divide PoI into two same parts PoI_- and PoI_+
14: **CheckPro**: (pk, T, C, PoI_-, d)
15: **CheckPro**: (pk, T, C, PoI_+, d)
16: **end if**
17: **until** $|PoI| = 1$
18: **return** Φ

The pseudocode of *CheckPro* is presented in Algorithm 5. *CheckPro* is the main algorithm used by the service vendor to verify the proofs of integrity returned by the challenged edge servers. Its average time complexity is $O(\log_2 n)$. This shows that the computational overheads incurred by *CheckPro* to the service vendor increase slowly with the number of edge data replicas to be verified. *It ensures that ICL-EDI achieves the second design objective discussed in Section 3.1* by allowing the service vendor to verify the integrity of a very large number of edge data replicas efficiently.

The result Φ returned by Algorithm 5 contains the verification results for all the n edge data replicas. On Line 7 in Algorithm 5, $\Phi(PoI)$ is a subset of Φ calculated based on PoI. For example, let us suppose ten edge data replicas in total, $\Phi = \{\phi_1, \ldots, \phi_{10}\}$. If PoI contains only three proofs of integrity, say $PoI = \{poi_1, poi_5, poi_9\}$, $\Phi(PoI)$ contains three corresponding proofs of integrity, i.e., $\Phi(PoI) = \{\phi_1, \phi_5, \phi_9\}$, calculated and returned by edge servers s_1, s_5 and s_9.

4 Performance Evaluation

As discussed in Section 3.2, the service vendor challenges the n edge servers on the integrity of n edge data replicas at the same time by using a pair of public and secret keys (pk, sk). To answer the challenge requests, each edge server $s_i \in S (1 \leq i \leq n)$ calculates the integrity proof for its edge data replica d_i individually. The probabilistic sampling plays an important role in the calculation of integrity proof. It significantly impacts the computational overheads incurred on the edge servers. In general, a decrease in the sampling rate will reduce the computational overheads incurred on the edge servers. However, it will also decrease the verification accuracy. To evaluate the performance of ICL-EDI, in this section, we first theoretically analyze its correctness and verification accuracy. Then, we experimentally evaluate its verification accuracy and efficiency.

4.1 Theoretical Analysis

Given a set of n edge servers $S = \{s_1, \ldots, s_n\}$ each caching one of the n edge data replicas, i.e., $D = \{d_1, \ldots, d_n\}$. Each edge data replica d_i contains m blocks $B = \{b_i^1, \ldots, b_i^m\}$. Let (pk, sk) denote the pair of the public key and secret key, where $pk = (N, g)$ and $sk = (p, q)$ as defined in Section 3.2. Based on the algorithms discussed in Section 3.2, the correctness of ICL-EDI is defined as follows.

Theorem 1. *If the transmission of data, e.g., homomorphic tags, public keys, proofs of integrity, etc., between the service vendor and the edge servers is reliable, the outputs of ICL-EDI are correct.*

Proof. Let $R = R'$ indicate that the integrity of all the edge data replicas is verified, where R' is the verification results obtained based on the original data in the service vendor's own storage and R is the verification results obtained based on the edge data replicas cached on the edge servers. A set of homomorphic tags $T_i = \{T_i^1, \ldots, T_i^m\}$ is generated by $T_i^j = (g^{b_i^j}) \bmod N$ for $b_i^j \in B$. Then, given the verification results returned by the edge data replicas $POI = poi_1, \ldots, poi_n$, the verification results R, are calculated as follows:

$$R = \left(\prod_{poi_i \in POI} (poi_i \bmod N) \right) \bmod N \tag{4}$$

Since

$$poi_i = g^{\sum_{a_i^j \in A_i} b_i^{a_i^j}} \tag{5}$$

for all the n edge data replicas, the integrity proofs are

$$POI = \left(\prod_{poi_i \in POI} (g^{\sum_{a_i^j \in A_i} b_i^{a_i^j}} \bmod N) \right) \bmod N$$
$$= \left(g^{\sum_{d_i \in D} \sum_{a_i^j \in A_i} b_i^{a_i^j}} \bmod N) \right) \bmod N \quad (6)$$

According to Eq. (3), there is $R = R'$. Thus, Theorem 1 holds. □

As defined in Section 3.2, *ProGen* randomly samples l blocks from edge data replica d_i. Assume that there are t corrupted blocks in edge data replica $d_i = \{b_i^1, \ldots, b_i^m\}$ and l is the number of blocks challenged by the service vendor. Theorem 2 ensures the probabilistic integrity guarantee of ICL-EDI.

Theorem 2. *Let X be the number of corrupted blocks in d_i that can be detected by challenging k sample blocks. We use $P(X \geq 1)$ here to represent the probability that at least one corrupted block is detected - the detection of one corrupted block suffices to confirm a corrupted edge data replica. $P(X \geq 1)$ fulfills:*

$$1 - \left(\frac{m-t}{m}\right)^l \leq P(X \geq 1) \leq 1 - \left(\frac{m-l+1-t}{m-l+1}\right)^l \quad (7)$$

Proof. $P(X \geq 1)$ can be converted as follows:

$$P(X \geq 1) = 1 - P(X = 0) \quad (8)$$

where $P(X = 0)$ means that none of the corrupted blocks is detected and is calculated as follows:

$$P(X = 0) = \frac{m-t}{m} \cdot \frac{m-t-1}{m-1} \cdot \ldots \cdot \frac{m-l+1-t}{m-l+1} \quad (9)$$

It implies that

$$P(X \geq 1) = 1 - \frac{m-t}{m} \cdot \frac{m-t-1}{m-1} \cdot \ldots \cdot \frac{m-l+1-t}{m-l+1} \quad (10)$$

As

$$\frac{m-i-t}{m-i} \geq \frac{m-i-t-1}{m-i-1}$$

it follows:

$$1 - \left(\frac{m-t}{m}\right)^l \leq P(X \geq 1) \leq 1 - \left(\frac{m-l+1-t}{m-l+1}\right)^l \quad (11)$$

Therefore, Theorem 2 holds. □

$P(X)$ indicates the probability for a service vendor to detect corrupted edge data replicas with ICL-EDI. It will be experimentally evaluated in Section 4.2.

4.2 Experimental Evaluation

To evaluate ICL-EDI, we implemented a prototype and conducted extensive experiments on a machine equipped with an Intel i5-7400T CPU and 8GB memory).

4.2.1 Baseline Approach

Since PDP was proposed in 2007, many variants have been proposed to offer new features, e.g., public verifiability and privacy preservation. Only a small body of work attempted to improve the efficiency of the PDP scheme [16], [17]. However, these improvements are made under one common assumption - the challenger (data owner) suffers from limited computing resources. Thus, techniques employed in these work are not suitable in EDI scenarios where it is the defenders, i.e., the edge servers that suffer from limited computing resources.

To our best knowledge, the MR-PDP [9] is remotely similar to ICL-EDI in that it also allows the verification of multiple replicas of the data owner's data. However, it is not chosen as the baseline approach in our experiments. There are two main reasons. First, MP-PDP focuses on verifying the uniqueness of the data replicas to ensure that there are indeed a number of unique replicas as promised by cloud storage providers. This is not critical in EDI and completely different from the design objectives for ICL-EDI discussed in Section 3.1. Second, to achieve its design objective, MP-PDP introduces extremely high computational overheads to pre-processing the original data and challenging the data replicas. According to the experimental results reported in [9], MR-PDP takes approximately 9 seconds to pre-process just one 1MB data replica and almost 3 seconds to challenge just 16 data replicas. Such high computational overheads immediately violate the requirement of EDI for low computational overheads, as discussed in Section 3.1.

As discussed in Section 3.1, EDI is fundamentally different from PDP. None of the existing PDP schemes is suitable for solving the EDI problem. Therefore, we have adapted the original PDP scheme [8] to leverage the service vendors' access to more computing resources compared to the edge servers. The new version of PDP, referred to as PDP-E in this section, is implemented as the baseline approach to be compared with ICL-IDE in the experiments. To shift the majority of the computational overheads from the edge servers to the service vendor, PDP-E employs the same *ProGen* algorithm as ICL-EDI for the edge servers to compute integrity proofs. As the challenger, the service vendor verifies the proofs of integrity received from edge servers individually. This also avoids the impact of the randomness in the pseudorandom function and the random keys employed by *ProGen* on the performance comparison between ICL-EDI and PDP-E. For the same reason, PDP-E also uses the same $KeyGen$ algorithm as ICL-EDI to avoid the impact of different methods for generating homomorphic tags, and inspects the edge data replica one after another to locate corrupted ones. In this way, a fair comparison can be performed between ICL-EDI and PDP-E.

4.2.2 Performance Metrics

For effective evaluation, we measure the accuracy, which is the ratio of the experiment instances where the corrupted edge data replicas are localized. A high accuracy indicates high effectiveness. For efficiency evaluation, we measure the total computation times taken by ICL-EDI and PDP-E to complete one challenge on average. Low computation time indicates high efficiency.

4.2.3 Experimental Settings

To comprehensively evaluate ICL-EDI, we simulate various EDI scenarios by changing the values of five experiment parameters:

- Verification Scale (n), measured by the number of edge data replicas. By changing the value of this parameter, we can evaluate ICL-EDI's performance in solving EDI problems at different scales.
- Data Size (m), measured by the number of blocks in the edge data as well as each edge data replica. By changing the value of this parameter, we can observe ICL-EDI's performance in handling data of different sizes.
- Corruption Severity (t), measured by the number of corrupted blocks in corrupted edge data replicas d_i. By changing the value of this parameter, we evaluate ICL-EDI's performance in EDI scenarios where edge date replicas might be slightly, moderately and severely corrupted.
- Corruption Rate (h), measured by the ratio of corrupted edge data replicas. By changing the value of this parameter, we evaluate ICL-EDI's performance in handling scenarios where different proportions of edge data replicas might be corrupted.
- Sampling Rate (l), measured by the ratio of sampled blocks in d. By changing the value of this parameter, we evaluate how the sampling rate impacts the performance of ICL-EDI.

Table 2 summarizes the parameter settings in the experiments. While the value of one of the parameters changes, the values of the other parameters are fixed. Both ICL-EDI and PDP-E are implemented in Java version 8. The parameters used in *KeyGen* and *TagGen* are specified based on widely-used Java encryption libraries Jasypt[2] and Chilkat[3].

4.2.4 Effectiveness Evaluation

Table 3 summarizes the accuracies achieved by ICL-EDI and PDP-E in the experiments. Figs. 3a, 3b, 3c, 3d, and 3e compares the verification accuracies achieved by ICL-EDI and PDP-E and illustrate the impacts of the five parameters. Overall, ICL-EDI achieves the same accuracy as PDP-E, an average of 82.26% across all the experiments. The same accuracy achieved by ICL-EDI and PDP-E is expected. As discussed in Section 4.2.1, we let PDP-E use the same *ProGen* and *KeyGen* algorithms as ICL-EDI to facilitate a fair comparison between ICL-EDI and PDP-E. Thus, ICL-EDI does not take advantage of its *ProGen* and *KeyGen* algorithms. Thus, the overall results shown in Figs. 3a, 3b, 3c, 3d, and 3e indicate that ICL-EDI does not sacrifice verification accuracy for efficiency. For a concise presentation, the remaining discussion in this subsection will focus on ICL-EDI.

As shown in Fig. 3a, the average verification accuracy achieved by ICL-EDI is 77.86%. Fig. 3a shows that in experiment Set #1, there is no specific correlation between ICL-EDI's verification accuracy and the number of edge data replicas. ICL-EDI's verification accuracy remains at a high level as the verification scale increases. In this experiment set, the corruption severity (t) is fixed at 1%. Given a fixed sampling rate (l), for each edge data replica, the possibility of finding the corrupted edge data replicas is similar but not exactly the same because the samples are generated randomly. This is illustrated by the slight fluctuations in Fig. 3a. ICL-EDI's average verification accuracy in experiment Set #2 is 78.83%, as shown in Fig. *3b*. In addition, Fig. 3b shows something similar to Fig. 3a - no significant correlation between ICL-EDI's verification accuracy and data size m (t). Similar to experiment Set #1, the possibility of ICL-EDI finding the corrupted edge data replicas in experiment Set #2 is slightly impacted by the randomness in the sampling process performed by the edge servers, as shown in Fig. 3b. Fig. 3c demonstrates the results of experiment Set #3. ICL-EDI achieves an average verification accuracy of 90.96% in this experiment set. As the corruption severity increases from 0.5% to 2.0%, ICL-EDI's verification accuracy increases rapidly. This indicates that ICL-EDI can leverage the extra information about corrupted edge data replicas to improve its verification accuracy. As the corruption severity continues to increase, ICL-EDI's verification accuracy approaches 1.0 and stabilizes. This shows that ICL-EDI is capable of handling highly volatile environments where edge data replicas may be severely corrupted. In environments with low corruption severity, ICL-EDI's verification accuracy can be improved by increasing its sampling rate (l), as will be demonstrated in Fig. 3e and discussed later. The corresponding increase in the computational overheads will be discussed later in Section 4.2.5.

TABLE 1
Key Notations

Notation	Meaning
a_i^j	jth sample block of edge data replica d_i
A_i	set of indexes of blocks sampled from edge data replica d_i
b_i^j	jth block of d_i
c_i	challenge request on d_i
C	set of challenge requests
d	original data
d_i	edge data replica of d on s_i
$\mathcal{F}$	pesudo-random function
g	generator that satisfies $g = \delta^2$, where $\gcd(\delta \pm 1, N) = 1$
l	number of samples or sampling rate
m	number of blocks in d and d_i
n	number of edge servers/edge data replicas
$N = pq$	public RSA modulus
poi_i	proof of integrity for d_i
PoI	set of n proofs of integrity for d_i
p, q	large prime
pk	public key
rk_i	pseudo random key for d_i
R	inspection result of PoI
sk	secret key
s_i	ith edge server
S	set of edge servers
t	corruption severity
T_i^j	homomorphic tag of the jth block of d_i
T_i	set of homomorphic tags for d_i
T	homomorphic tags of n edge data replicas

2. http://www.jasypt.org/
3. http://www.chilkatsoft.com/java-encryption.asp

TABLE 2
Experiment Settings

	Verification Scale (n)	Data Size (m)	Corruption Severity (t)	Corruption Rate (h)	Sampling Rate (l)
Set #1	$2^8, 2^9, \ldots, 2^{17}$	2^{16}	1%	3‰	3‰
Set #2	2^{12}	$2^{10}, 2^{11}, \ldots, 2^{19}$	1%	3‰	3‰
Set #3	2^{12}	2^{16}	0.5%,1%,..., 5%	3‰	3‰
Set #4	2^{12}	2^{16}	1%	1‰, 2‰,...,10‰	3‰
Set #5	2^{12}	2^{16}	1%	3‰	1‰, 2‰,...,10‰

Fig. 3d shows the results of experiment Set #4. ICL-EDI's average verification accuracy is 79.04. In this experiment set, the ratio of corrupted edge data replicas increases from 0.1% to 1.0%. Given n edge data replicas, ICL-EDI will not stop when it finds a corrupted edge data replica. Instead, it always inspects all the n verification proofs returned by the edge servers. Thus, the corruption rate does not impact ICL-EDI's verification accuracy. In experiment Set #5, ICL-EDI's average verification accuracy is 84.59%, as shown in Fig. 3e. As the sampling rate increases from 1‰to 10‰, ICL-EDI's verification accuracy increases rapidly and stabilizes. This is because a higher sampling rate allows ICL-EDI to inspect the edge data replicas more thoroughly.

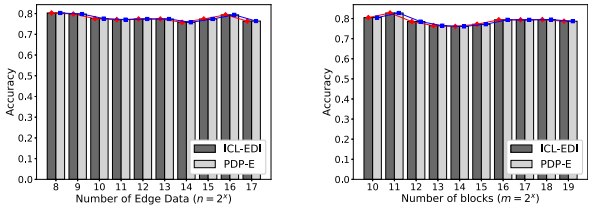
4.2.5 Efficiency Evaluation

Table 4 summarizes the computational overheads of ICL-EDI and PDP-E in different experiment sets. Figs. 4a, 4b, 4c, 4d, and 4e illustrate the average total times lower than ICL-EDI and PDP-E to complete the verification of all the edge data replicas in experiment Set #1 - Set #5. *In general, ICL-EDI takes order-of-magnitude less time than PDP-E to complete the verification.* Across all four sets of experiments, ICL-EDI's average time consumption is 24.64 milliseconds, 97.50% lower than PDP-E. This is made possible by ICL-EDI's $O(\log_2 n)$ computation complexity as analyzed in Section 3.2. The overall results illustrate that ICL-EDI is capable of verifying EDI efficiently in a variety of EDI scenarios.

As shown in Fig. 4a, in experiment Set #1, ICL-EDI's average time consumption is only 64 milliseconds, 97.90% lower than PDP-E. As the number of edge data replicas to be inspected increases from 2^8 to 2^{17}, PDP-E's overall time consumption experiences an exponential increase from 30 milliseconds to 15,232.9 milliseconds. In the meantime, the time consumption of ICL-EDI increases only slightly, from 1 millisecond to about 322 milliseconds. Fig. 4b shows the results in experiment Set #2, 9.7 milliseconds for ICL-EDI versus 483.1 milliseconds for PDP-E. ICL-IDE takes only as much as 97.99% of the time taken by PDP-E to complete the verification. Fig. 4b also shows that the efficiency of ICL-IDE is not impacted by the increase in the data size (m). The reason is that the increase in data size does not incur extra computational overheads to either ICL-IDE or PDP-E. For a similar reason, in Fig. 4c, we can see that the increase in the corruption severity does not impact the efficiency of either ICL-EDI or PDP-E in experiment Set #3. However, ICL-EDI's average time consumption in this experiment set is 97.98% lower than PDP-E's, i.e., 9.8 milliseconds versus 485.1 milliseconds. Fig. 4d shows that, in experiment Set #4, the increase in the corruption rate does not impact ICL-EDI's time consumption. As discussed in Section 4.2.4, ICL-EDI always inspects all the n verification proofs returned by the edge servers regardless. Thus, it takes approximately a similar time to complete the verification as the corruption rate increases. Fig. 4e shows the overall time consumption of ICL-EDI and PDP-E as the sampling rate increases from 1‰to 10‰. On average, ICL-EDI takes 18.9 milliseconds to complete the verification, 97.89% lower than PDP-E in experiment Set #5. Given the same data size, an increase in the sampling rate results in an increase in the time for the computation and verification of the samples. As the sampling rate l increases from 1‰to 10‰, ICL-EDI's time consumption only experiences a slight increase from 4 milliseconds to 34 milliseconds while PDP-E's time consumption increases rapidly from 159 milliseconds to 1,634 milliseconds.

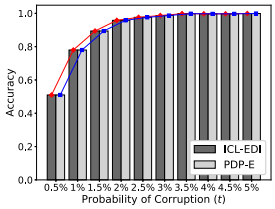
TABLE 3
Accuracy versus Parameters

Set #1	**Verification Scale (n)**	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	**AVG**
	ICL-EDI	0.803	0.798	0.776	0.771	0.774	0.774	0.758	0.774	0.794	0.764	0.7786
	PDP-E	0.803	0.798	0.776	0.771	0.774	0.774	0.758	0.774	0.794	0.764	0.7786
Set #2	**Data Size (m)**	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	**AVG**
	ICL-EDI	0.805	0.828	0.785	0.765	0.761	0.772	0.794	0.794	0.795	0.787	0.7886
	PDP-E	0.805	0.828	0.785	0.765	0.761	0.772	0.794	0.794	0.795	0.787	0.7886
Set #3	**Corruption Severity (t)**	0.5%	1%	1.5%	2%	2.5%	3%	3.5%	4%	4.5%	5%	**AVG**
	ICL-EDI	0.51	0.78	0.894	0.958	0.976	0.987	0.998	0.997	0.997	0.999	0.9096
	PDP-E	0.51	0.78	0.894	0.958	0.976	0.987	0.998	0.997	0.997	0.999	0.9096
Set #4	**Corruption Rate (h)**	1‰	2‰	3‰	4‰	5‰	6‰	7‰	8‰	9‰	10‰	**AVG**
	ICL-EDI	0.771	0.788	0.786	0.791	0.814	0.803	0.785	0.774	0.821	0.771	0.7904
	PDP-E	0.771	0.788	0.786	0.791	0.814	0.803	0.785	0.774	0.821	0.771	0.7904
Set #5	**Sampling Rate (l)**	1‰	2‰	3‰	4‰	5‰	6‰	7‰	9‰	9‰	10‰	**AVG**
	ICL-EDI	0.396	0.621	0.769	0.844	0.915	0.967	0.973	0.987	0.993	0.994	0.8459
	PDP-E	0.396	0.621	0.769	0.844	0.915	0.967	0.973	0.987	0.993	0.994	0.8459

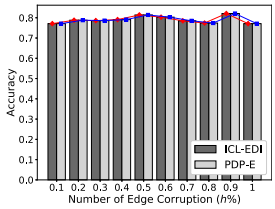


(a) Set #1: vs. n

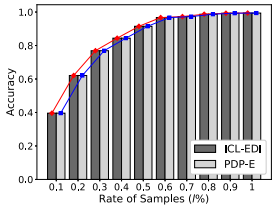
(b) Set #2: vs. m



(c) Set #3: vs. t



(d) Set #4: vs. h



(e) Set #5: vs. l

Fig. 3. Accuracy versus parameters.

The χ-square test is conducted to evaluate the statistical significance of the experimental results. As demonstrated and discussed in Section 4.2.4, ICL-EDI and PDP-E achieve the same accuracy in the experiments. Thus, the corresponding statistical test results are omitted here. In terms of efficiency, most of the p-values obtained from the χ-square test fall into the range of (0.003,0.048), all below 0.05. This indicates that the efficiency of ICL-EDI is statistically higher than PDP-E.

5 Related Work

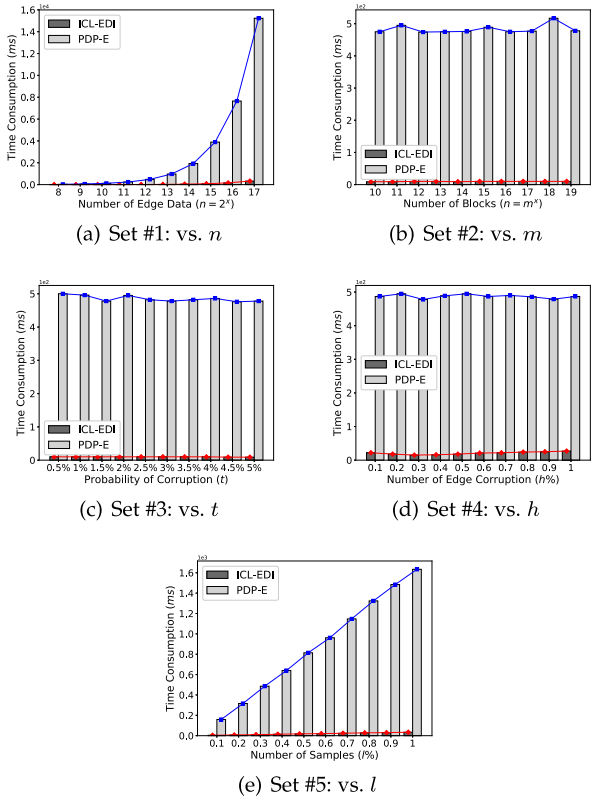
Edge computing allows service vendors like Facebook and Uber to deploy their services on edge servers at the network edge to ensure low service latency for their users [18]. It tackles many disadvantages of cloud computing and in the meantime, poses many new challenges, including computation offloading [19], [20], user allocation [21], [22], [23], [24], [25], application deployment [18], [26], collaborative edge computing [27].

In addition to these research problems, edge data caching that studies when, how, and where to cache data on edge servers to serve users has attracted tremendous attention [12], [28], [29], [30], [31], [32], [33]. It allows service vendors to provision low-latency data retrieval services to their users [14], [34]. Vu *et al.* [12] studied multi-layer edge data caching where edge servers and users are both capable of caching data, taking into account wireless signal transmission at the physical layer. Similarly, Zhang *et al.* employed both in-network devices and edge servers to cache data with the aim to guarantee multimedia transmissions quality [35]. Chen *et al.* [31] tried to combine data caching and computation offloading at the network edge, also considering the communication aspect as an additional dimension. Zhang *et al.* [33] proposed an approximation method for caching data on edge servers to serve clustered users with optimized bandwidth allocation. Gabry *et al.* [32] studies edge data caching with the aim to maximize the overall energy efficiency, considering the backhaul rate for distributing data. Drolia *et al.* [29] included both edge servers and remote cloud in their edge data caching solution. They proposed an approach named Cachier to balance the caching workloads between cloud and edge servers, while minimizing users' data retrieval latency. Zhang *et al.* [28] proposed a cooperative architecture for enhancing edge caching by including smart vehicles as edge servers. To find a new way to improve edge data caching, Xia *et al.* proposed to utilize edge servers' storage resources collectively through collaborative edge data storage [34] and collaborative edge data distribution [6].

Caching data on edge servers presents new security concerns as edge servers usually cannot be maintained in the same way as cloud servers hosted in centralized data centers [11]. Thus, edge data integrity (EDI) has become a significant obstacle that challenges various service vendors in the edge computing environment. Thus, approaches that

TABLE 4
Computational Overheads versus Parameters (ms)

Set #1	Verification Scale (n)	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	AVG
	ICL-EDI	1	1	3	5	10	19	40	79	160	322	64
	PDP-E	30	59	119	239	479	970	1928	3892	7651	15232	3059.9
Set #2	Data Size (m)	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	AVG
	ICL-EDI	9	9	10	10	9	10	10	10	10	10	9.7
	PDP-E	475	495	474	475	476	489	475	477	517	478	483.1
Set #3	Corruption Severity (t)	0.5%	1%	1.5%	2%	2.5%	3%	3.5%	4%	4.5%	5%	AVG
	ICL-EDI	10	10	10	10	10	10	10	10	9	9	9.8
	PDP-E	500	496	478	495	482	478	482	486	476	478	485.1
Set #4	Corruption Rate (h)	1‰	2‰	3‰	4‰	5‰	6‰	7‰	8‰	9‰	10‰	AVG
	ICL-EDI	22	18	15	16	18	21	22	24	25	27	20.8
	PDP-E	487	495	478	489	495	487	490	486	479	487	487.3
Set #5	Sampling Rate (l)	1‰	2‰	3‰	4‰	5‰	6‰	7‰	9‰	9‰	10‰	AVG
	ICL-EDI	4	7	10	14	18	20	24	28	30	34	18.9
	PDP-E	159	316	486	640	814	962	1147	1324	1484	1634	896.6



(a) Set #1: vs. n (b) Set #2: vs. m

(c) Set #3: vs. t (d) Set #4: vs. h

(e) Set #5: vs. l

Fig. 4. Computational overheads versus parameters.

can help service vendors ensure edge data integrity are urgently needed [7]. There is a large body of schemes for protecting data integrity in the cloud computing environment. The most representative ones are PDP (provable data possession) [8] and its variants. The PDP scheme is designed to inspect whether a piece of users' data stored on remote cloud is correct. However, these schemes designed to ensure cloud data integrity are not suitable for ensuring edge data integrity because of the inevitable high computational overheads incurred to service vendors and/or edge servers.

To attack the EDI problem, Tong *et al.* [10] employed the conventional PDP scheme to inspect the integrity of users' data saved on edge servers, with a focus on preserving user's privacy. Their approach is implemented as PDP-E in our experiments to be compared with ICL-EDI. Unfortunately, PDP-E inherits the limitation of low efficiency from PDP schemes and poses a high computation burden on edge servers whose computing resources are highly constrained compared with cloud servers. The other major limitation of PDP-E is its inability to verify the proofs of integrity for multiple edge data replicas simultaneously. This cripples PDP-E in the edge computing environment where massive data replicas must be cached on many edge servers to serve the users in different areas [14]. As a straightforward application of the PDP scheme, PDP-E achieves the same accuracy as the original PDP scheme and ICL-EDI in detecting corrupted edge data replicas. However, it suffers from low efficiency. Thus, it is not a practical solution to the EDI problem. Yue *et al.* [36] proposed a blockchain-based scheme to ensure edge data integrity for both data owners and data consumers. However, they failed to consider the fact that multiple data replicas need to be verified in the edge computing environment. In addition to their respective limitations, a common and critical limitation of the above EDI approaches is their inability to localize corrupted edge data replicas.

Very recently, Li *et al.* proposed a PDP-liked scheme to verify the data integrity in the edge computing environment [7]. Specifically, the service vendor generates a Merkle Hash Tree (MHT) based on the original file as ground truth. Then, each edge server generates a new MHT based on data blocks sampled from its own edge data replica. If the replica is not corrupted, this MHT should be a subtree of the ground truth. By sequential comparison, the service vendor can verify the integrity of an edge data replica. However, the application of MHT limits the sampling flexibility - the verification has to use 2^k (k is the tree height) data blocks, which either undermines verification accuracy (with a small k) or reduces verification efficiency (with a large k). Later, they proposed another EDI approach named EDI-S based on elliptic curve cryptography [37]. Different from ICL-EDI, EDI-S is a deterministic approach that has to generate integrity proofs based on the entire edge data replica. As a result, it incurs high computational overheads on edge servers, suffering from the same limitation as PDP schemes.

To attack the EDI problem efficiently, we proposed ICL-EDI to help service vendors inspect their edge data and localize corrupted edge data replicas among multiple edge servers. ICL-EDI overcomes the limitations of state-of-the-art approaches by 1) inspecting all the edge data replicas at the same time; and 2) efficiently localizing corrupted edge data replicas.

6 Conclusion and Future Work

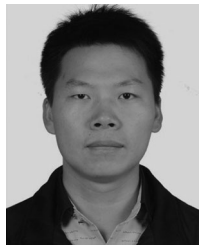
In this paper, we proposed ICL-EDI, an inspection and corruption localization scheme for edge data integrity with three unique features. First, ICL-EDI is a lightweight scheme that does not incur a high computation burden on edge servers. Second, it allows a service vendor to validate multiple edge data replicas simultaneously and efficiently. Third, it allows a service vendor to localize corrupted edge data replicas efficiently. We theoretically proved the correctness and probability of data corruption localization. We also implemented ICL-EDI and conducted extensive experiments to evaluate its performance. The results showed that ICL-EDI outperforms the state-of-the-art approaches significantly in efficiency without accuracy loss.

ICL-EDI is designed to inspect multiple replicas of the same edge data. In the future, we will extend ICL-EDI to allow service vendors to verify different edge data simultaneously. In addition, ICL-EDI may be vulnerable to certain attacks. For example, malicious edge servers may falsify data replicas and forge integrity proofs to cheat the verification. We will also investigate potential attacks against ICL-EDI and design corresponding defense mechanisms.

References

[1] P. Lai *et al.*, "Cost-effective user allocation in 5G NOMA-based mobile edge computing systems," *IEEE Trans. Mobile Comput.*, early access, May 04, 2021, doi: 10.1109/TMC.2021.3077470.

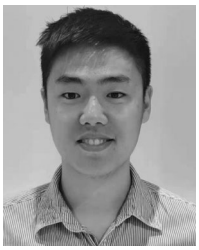
[2] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, Oct.–Dec. 2017.
[3] P. Ranaweera, A. D. Jurcut, and M. Liyanage, "Survey on multi-access edge computing security and privacy," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 1078–1124, Apr.–Jun. 2021.
[4] H. Wu, F. Lyu, C. Zhou, J. Chen, L. Wang, and X. Shen, "Optimal UAV caching and trajectory in aerial-assisted vehicular networks: A learning-based approach," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 12, pp. 2783–2797, Dec. 2020.
[5] F. Lyu *et al.*, "LeaD: Large-scale edge cache deployment based on spatio-temporal WiFi traffic statistics," *IEEE Trans. Mobile Comput.*, vol. 20, no. 8, pp. 2607–2623, Aug. 2020.
[6] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Cost-effective app data distribution in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 31–44, Jan. 2021.
[7] B. Li, Q. He, F. Chen, H. Jin, Y. Xiang, and Y. Yang, "Auditing cache data integrity in the edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1210–1223, May 2021.
[8] G. Ateniese *et al.*, "Provable data possession at untrusted stores," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2007, pp. 598–609.
[9] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-replica provable data possession," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2008, pp. 411–420.
[10] W. Tong, B. Jiang, F. Xu, Q. Li, and S. Zhong, "Privacy-preserving data integrity verfication in mobile edge computing," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 1007–1018.
[11] Q. He *et al.*, "A game-theoretical approach for mitigating edge DDoS attack," *IEEE Trans. Dependable Secure Comput.*, early access, Jan. 29, 2021, doi: 10.1109/TDSC.2021.3055559.
[12] T. X. Vu, S. Chatzinotas, and B. Ottersten, "Edge-caching wireless networks: Performance analysis and optimization," *IEEE Trans. Wireless Commun.*, vol. 17, no. 4, pp. 2827–2839, Apr. 2018.
[13] G. Cui, Q. He, F. Chen, H. Jin, and Y. Yang, "Trading off between user coverage and network robustness for edge server placement," *IEEE Trans. Cloud Comput.*, early access, Jul. 10, 2020, doi: 10.1109/TCC.2020.3008440.
[14] X. Xia, F. Chen, J. Grundy, M. Abdelrazek, H. Jin, and Q. He, "Constrained app data caching over edge server graphs in edge computing environment," *IEEE Trans. Serv. Comput.*, early access, Feb. 24, 2021, doi: 10.1109/TSC.2021.3062017.
[15] R. Johnson, D. Molnar, D. Song, and D. Wagner, "Homomorphic signature schemes," in *Proc. Cryptographers' Track RSA Conf.*, 2002, pp. 244–262.
[16] G. Ateniese, R. Di Pietro , L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proc. Int. Conf. Secur. Privacy Commun. Netow.*, 2008, Art. no. 9.
[17] Y. Wang, Q. Wu, B. Qin, S. Tang, and W. Susilo, "Online/offline provable data possession," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 5, pp. 1182–1194, May 2017.
[18] B. Li *et al.*, "READ: Robustness-oriented edge application deployment in edge computing environment," *IEEE Trans. Serv. Comput.*, early access, Aug. 10, 2020, doi: 10.1109/TSC.2020.3015316.
[19] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, Jul./Sep. 2017.
[20] H. Zhao, S. Deng, C. Zhang, W. Du, Q. He, and J. Yin, "A mobility-aware cross-edge computation offloading framework for partitionable applications," in *Proc. IEEE Int. Conf. Web Serv.*, 2019, pp. 193–200.
[21] P. Lai *et al.*, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Proc. Int. Conf. Serv.-Oriented Comput.*, 2018, pp. 230–245.
[22] P. Lai *et al.*, "Edge user allocation with dynamic quality of service," in *Proc. Int. Conf. Serv.-Oriented Comput.*, 2019, pp. 86–101.
[23] Q. He *et al.*, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 515–529, Mar. 2020.
[24] P. Lai *et al.*, "Quality of experience-aware user allocation in edge computing systems: A potential game," in *Proc. 40th IEEE Int. Conf. Distrib. Comput. Syst.*, 2020, pp. 223–233.
[25] G. Cui, Q. He, F. Chen, Y. Zhang, H. Jin, and Y. Yang, "Interference-aware game-theoretic device allocation for mobile edge computing," *IEEE Trans. Mobile Comput.*, early access, Mar. 5, 2021, doi: 10.1109/TMC.2021.3064063.
[26] F. Chen, J. Zhou, X. Xia, H. Jin, and Q. He, "Optimal application deployment in mobile edge computing environment," in *Proc. 13th IEEE Int. Conf. Cloud Comput.*, 2020, pp. 184–192.
[27] L. Yuan *et al.*, "CoopEdge: A decentralized blockchain-based platform for cooperative edge computing," in *Proc. 30th Web Conf.*, 2021, pp. 2245–2257.
[28] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Cooperative content caching in 5G networks with mobile edge computing," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 80–87, Jun. 2018.
[29] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 276–286.
[30] J. Yao, T. Han, and N. Ansari, "On mobile edge caching," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2525–2553, Jul./Sep. 2019.
[31] M. Chen, Y. Hao, L. Hu, M. S. Hossain, and A. Ghoneim, "Edge-CoCaCo: Toward joint optimization of computation, caching, and communication on edge cloud," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 21–27, Jun. 2018.
[32] F. Gabry, V. Bioglio, and I. Land, "On energy-efficient edge caching in heterogeneous networks," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3288–3298, Dec. 2016.
[33] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 8, pp. 1791–1805, Aug. 2018.
[34] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 281–294, Feb. 2021.
[35] X. Zhang and Q. Zhu, "Hierarchical caching for statistical QoS guaranteed multimedia transmissions over 5G edge computing mobile wireless networks," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 12–20, Jun. 2018.
[36] D. Yue, Y. Li, Ruixuan amd Zhang, W. Tian, and Y. Huang, "Blockchain-based verification framework for data integrity in edge-cloud storage," *J. Parallel Distrib. Comput.*, vol. 146, pp. 1–14, 2020.
[37] B. Li, Q. He, F. Chen, H. Jin, Y. Xiang, and Y. Yang, "Inspecting edge data integrity with aggregated signature in distributed edge computing environment," *IEEE Trans. Cloud Comput.*, early access, Feb. 16, 2021, doi: 10.1109/TCC.2021.3059448.



Guangming Cui received the master's degree from Anhui University, Hefei, China, in 2018. He is currently working toward the PhD degree with Swinburne University of Technology, Melbourne, Australia. His research interests include software engineering, edge computing and service computing.

Qiang He (Senior Member, IEEE) received the first PhD degree from the Swinburne University of Technology, Melbourne, Australia, in 2009 and the second PhD degree from the Huazhong University of Science and Technology, Wuhan, China, in 2010. He is currently an associate professor with Swinburne. His research interests include mobile edge computing, cloud computing, software engineering and service computing. For more information, please visit https://sites.google.com/site/heqiang/.

Bo Li received the MS degree from Shandong Normal University, Jinan, China, in 2010. He is currently working toward the PhD degree with the Swinburne University of Technology, Melbourne, Australia. His research interests include software engineering, edge computing, and network security issues.



Xiaoyu Xia received the master's degree from The University of Melbourne, Melbourne, Australia, in 2015. He is currently working toward the PhD degree with Deakin University. His research interests include edge computing, service computing, and software engineering.

Feifei Chen (Member, IEEE) received the PhD degree from the Swinburne University of Technology, Melbourne, Australia, in 2015. She is currently a lecturer at Deakin University. Her research interests include software engineering, edge computing, cloud computing, and green computing.

Hai Jin (Fellow, IEEE) received the PhD degree in computer engineering from the Huazhong University of Science and Technology, Wuhan, China, in 1994. He is currently a full professor with Huazhong University of Science and Technology. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

Yang Xiang (Fellow, IEEE) received the PhD degree from Deakin University, Geelong, Australia. He is currently a full professor with Swinburne University of Technology, Australia. His research interests include cyber security, data analytics, distributed systems, and networking.

Yun Yang (Senior Member, IEEE) received the PhD degree in computer science from the University of Queensland, Brisbane, Australia, in 1992. He is currently a full professor with Swinburne University of Technology, Melbourne, Australia. His research interests include software technologies, cloud and edge computing, workflow systems, and service computing.