Journal of Machine Learning and Information Security
https://www.sciltp.com/journals/jmlis

Article

Secure Edge Data Auditing with Multiple Vendors and Servers

Zifeng Qin, Haojun Miao and Fei Chen *

College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060 , China
* Correspondence: fchen@szu.edu.cn

How To Cite: Qin, Z.; Miao, H.; Chen, F. Secure Edge Data Auditing with Multiple Vendors and Servers. *Journal of Machine Learning and Information Security* **2025**, *1*(1), 3.

Received: 4 August 2025
Revised: 28 September 2025
Accepted: 10 October 2025
Published: 22 October 2025

Abstract: Mobile Edge Computing (MEC) has emerged as a pivotal solution to reduce latency for latency-sensitive applications by deploying data closer to end-users. Ensuring data integrity in mobile edge computing environments has become increasingly critical. While existing research primarily focuses on Single-Vendor Multi-Server EDI (SVMS-EDI), practical MEC deployments often involve multiple application vendors (AVs) sharing edge servers. Unlike SVMS-EDI, multiple application vendors bring significant computation cost and communication cost. Moreover, in addition to the security model of SVMS-EDI problem, application vendors are not fully trusted under the Multi-Vendor Multi-Server (MVMS) scenario. Therefore, robust solutions tailored for MVMS-EDI are required. This paper introduces MVMS-HMAC, a novel scheme that combines HMAC-based verification with distributed ledger technology to address the MVMS-EDI problem efficiently and securely. The proposed solution effectively counters key threats such as cheating attacks from malicious AVs and forge, replace, and replay attacks from compromised edge servers. By incorporating selective auditing, MVMS-HMAC minimizes computation and communication overhead while maintaining strong security guarantees. Theoretical analysis and experimental evaluations demonstrate the scheme's correctness, security, and efficiency, outperforming existing approaches. The proposed scheme advances the field by providing a comprehensive problem model and security framework for MVMS-EDI, along with a practical scheme that enhances resistance to adversarial behaviors. The open-source implementation further facilitates community engagement and adoption.

Keywords: mobile edge computing; data integrity auditing; multi-vendor multi-server; smart contract

1. Introduction

The rapid advancement of mobile Internet technologies, especially 5G, has driven widespread adoption of latency-sensitive applications like autonomous driving, online game and cloud gaming[1–3]. Traditional cloud architectures fail to meet their stringent latency requirements, making Mobile Edge Computing (MEC) an effective alternative by deploying services closer to end-users [4]. For instance, EdgeOne platform of Tencent Cloud significantly reduces latency of online game by using edge servers [5]. However, MEC introduces critical security challenges, particularly regarding data integrity on edge servers [6]. Data corruption from failures or attacks can severely impact services, from game disruptions to safety-critical failures in autonomous systems [6]. Ensuring Edge Data Integrity (EDI) is therefore essential for MEC industry [7].

While existing research has proposed various solutions for Edge Data Integrity (EDI) problem [8–13], most focus on Single-Vendor Multi-Server scenarios. Practical deployments, however, typically involve Multi-Vendor Multi-Server environments where edge servers host data from multiple vendors concurrently [8]. The architectural differences between SVMS-EDI and MVMS-EDI prevent direct solution migration. SVMS-EDI's single-vendor-per-server model enables straightforward auditing, while MVMS-EDI's multi-tenancy requires servers to simultaneously process verification requests from multiple vendors, creating significant computation overhead. Furthermore, the all-to-all communication pattern between vendors and their leased servers leads to prohibitive communication cost [8]. Moreover, multiple application vendors bring more complex security situation. Application vendors may not be full-trusted. Malicious vendors may try to extort by denying correct auditing result. These fundamental differences demand novel approaches specifically designed for MVMS-EDI scenarios.

Recent research has begun to address the MVMS-EDI problem, with the smart contract based scheme MVMS-SC emerging as a promising solution [8]. By selectively verifying a subset of edge servers, this approach significantly reduces both computational overhead and communication cost while effectively addressing MVMS-EDI requirements. However, the MVMS-SC scheme exhibits critical security vulnerabilities that limit its practical application. Malicious application vendors may launch cheating attack to fraudulently deny verification results and potentially extort the edge infrastructure platform, while compromised edge servers could employ forge attack, replay attack or replace attack to deliberately conceal data corruption and evade responsibility. Unfortunately, the current MVMS-SC framework demonstrates insufficient resistance against these threats, particularly in defending against cheating attack from application vendors and forge attack from edge servers.

Therefore, we propose MVMS-HMAC, a novel solution building upon MVMS-SC's selective auditing approach while maintaining efficiency. In response to the security situation of MVMS-EDI problem, MVMS-HMAC incorporates robust mechanisms to effectively counter dishonest behaviors from both application vendors and edge servers. To deal with the malicious application vendor, MVMS-HMAC uses smart contracts to run the main auditing process and use one smart contract to verify the original data. To defend the attack from malicious edge servers, MVMS-HMAC leverages the HMAC message authentication method. The proposed solution not only preserves high efficiency but also significantly enhances security guarantees. Furthermore, we open-source the MVMS-HMAC implementation [14]. This work represents a step forward in developing practical, secure, and efficient solutions for the MVMS-EDI problem in real-world edge computing environments.

Our main contributions can be summarized as follows:

- We proposed a new MVMS-HMAC scheme, which can effectively and securely address the edge data integrity issue in scenarios involving multiple application vendors and multiple edge servers.
- We conducted a systematic analysis of MVMS-HMAC, as well as demonstrating its ability to effectively counter cheating attacks from dishonest application vendors and forge attacks, replace attacks, and replay attacks from dishonest edge servers.

The structure of this paper is organized as follows. Section 2 provides an overview of the related work. Section 3 defines and models MVMS-EDI problem, and details the security model as well as the objectives of the scheme design. Section 4 introduces MVMS-HMAC and brings its design details. Section 5 presents a theoretical analysis of MVMS-HMAC. Section 6 offers an experimental analysis of MVMS-HMAC. Finally, Section 7 concludes the paper.

2. Related Work

Before the EDI problem received research attention, the issue of Cloud Data Integrity (CDI) in cloud computing had already been widely studied, and several mature schemes were proposed. Among them, two are the most representative: the first is Provable Data Possession (PDP) [15], and the second is Proofs of Retrievability (PoR) [16]. PDP processes data by dividing it into blocks and attaching authentication tags to each data block. During auditing, the user requests the cloud to return a portion of the data and the corresponding authentication tags. The user then recomputes the evidence from the retrieved data and compares it with the proof computed by the cloud to verify data integrity. On the other hand, PoR also divides data into blocks during storage. However, unlike PDP, it randomly inserts sentinel blocks into the data blocks and shuffles all blocks. During auditing, PoR instructs the cloud to randomly return data blocks, including sentinel blocks. By verifying the sentinels, the user can, with a certain probability, trust the integrity of the data.

Early research on CDI primarily focused on single-cloud scenarios [17–19]. Subsequently, it gradually expanded to multi-cloud environments [20,21]. However, the CDI problem still differs from the Edge Data Integrity (EDI) problem, making CDI schemes difficult to apply to solving the EDI issue. Therefore, researchers began to seek alternative approaches. Tong et al. [22] were among the first to study data integrity in Mobile Edge Computing (MEC) environments. The proposed scheme utilized PDP to perform data integrity auditing on edge servers, which resulted in a significant computational burden. Later, Li et al. [13] proposed EDI-V, a method based on Merkle hash tree. Subsequently, they introduced EDI-S [12], which employs Schnorr aggregated signatures. Cui et al. [23] proposed ICL-EDI, which uses homomorphic authentication tags for EDI auditing. Ding et al. [24] introduced EDI-DA, which utilizes an index-single linked table and dynamic arrays to support dynamic data. Furthermore, Li et al. [11] proposed CooperEDI, which allows edge servers to collaboratively perform audits, as well as EdgeWatch [25], a joint audit scheme incorporating blockchain technology. Additionally, Tong et al. [26] proposed an audit method that introduces a third party auditor (TPA). More recently, Wang et al. [27] proposed an auditing scheme that features easy key management and damaged data identification. Zhou et al. [28] extends storage auditing to support multi-copy cloud storage scenario. For multi-cloud storage auditing, Yang et al. [29] proposed to leveraging mutual auditing between different clouds, thus eliminating the need of a third-party auditor.

Existing schemes for constructing the EDI problem all assume a single application vendor, addressing the single-vendor and multi-server EDI scenarios. Research on the multi-vendor multi-server (MVMS) EDI problem is still limited. When the number of application vendors increases from one to multiple, the trustworthiness of each vendor becomes a crucial consideration. Zhao et al. [8] proposed MVMS-SC. This scheme scores edge servers based on their un-audited duration and quality of service (QoS). During auditing, the smart inspection algorithm selects the lowest-scoring edge servers for auditing, addressing the issue of increased computational burdens on some edge servers due to multiple computations. While this method assumes that application vendors cannot be fully trusted, it does not effectively address this assumption and performs inadequately in handling malicious edge servers. Subsequently, Islam et al. [30] introduced MEDI-V, which uses a Merkle hash tree (MHT) for auditing. Application vendors first construct the MHT based on the original data and transmit the minimal necessary information required for construction to the edge servers. Edge servers then construct the MHT using the data replicas and the provided information, returning the root value to the application vendors for verification. This method demonstrates good efficiency but is unable to handle malicious application vendors effectively.

Compared to existing schemes, our work is dedicated to providing a more detailed and comprehensive security model for MVMS-EDI. We propose a scheme that is correct, efficient, and secure, better addressing the challenges posed by malicious application vendors and malicious edge servers.

3. Problem Formulation

3.1. Overview

In the context of Mobile Edge Computing (MEC), application vendors cache replicas of their original data on edge servers to reduce latency [31]. Edge servers, deployed by various infrastructure providers at regional base stations or access points [32], are located in close proximity to users within those regions. Users in these areas can access edge servers to obtain significantly faster responses compared to traditional cloud server models. The Edge Data Integrity (EDI) problem focuses on detecting when data replicas on edge servers become corrupted and ensuring data integrity.

Figure 1 illustrates a MEC environment in a multi-vendor, multi-server scenario. In this setting, each application vendor can deploy replicas of its data across multiple edge servers, and each edge server may be shared by different application vendors. For example, AV_1 caches d'_1 on ES_1, ES_3, and ES_n; AV_2 caches d'_2 on ES_2 and ES_3; and AV_m caches d'_m on ES_n. Consequently, ES_1 and ES_2 each store a single data replica, whereas ES_3 and ES_n store two replicas each.

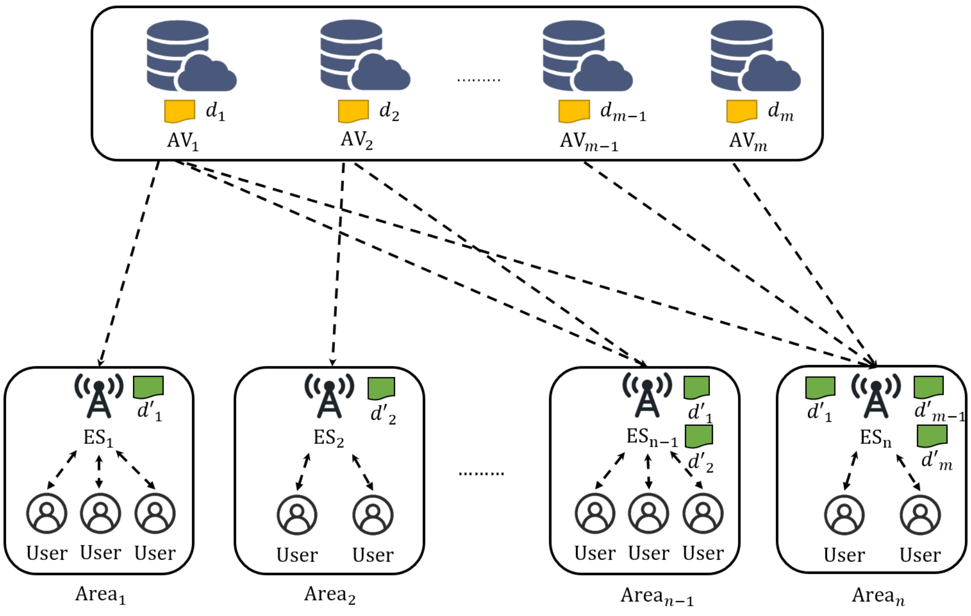


Figure 1. MEC Environment with Multi-Vendor and Multi-Server.

Existing solutions designed for the Single-Vendor Multi-Server EDI (SVMS-EDI) problem encounter significant challenges when extended to the Multi-Vendor Multi-Server EDI (MVMS-EDI) scenario [8]. This is primarily because each application vendor must send requests to every edge server it utilizes, requiring each server to audit all corresponding data replicas and return the results. Thus, edge servers storing replicas from multiple vendors must perform multiple audits. For instance, ES_3 would need to conduct two separate audits, which substantially increases both computational and communication cost. For clarity, Table 1 summarizes the notations used throughout this work.

Table 1. Notation.

Symbol	Description
AV	Set of application vendors
ES	Set of edge servers
AV_i	The i-th application vendor
S_i	Set of edge servers storing data replicas for AV_i
s_{ij}	The j-th edge server in S_i
d_i	Original data belonging to AV_i
λ	Security level
K_s	Key used for computing HMAC during setup phase
D	Collection of all original data
b_i	Number of edge servers that AV_i needs to verify
C_i	Challenge from AV_i
S'_i	Set of edge servers being audited by AV_i
S'	Set of all edge servers that need to be audited
K	Key used for HMAC computation
s'_j	The j-th edge server in S'
d'_i	Data replica of d_i, belonging to AV_i
D'_j	Set of data replicas in s'_j
r_{ji}	Proof information of d'_i in s'_j
p_j	Set containing all proof information generated by s'_j and the associated application vendor ID
P	Set of all p_j
r'_i	Proof information computed from d_i
p'_i	Computed by AV_i, containing its ID and r'_i
P'	Set of all p'_i
Ψ	Set of information about corrupted data replicas

3.2. System Model

We present a formal system model to address the Multi-Vendor Multi-Server Edge Data Integrity (MVMS-EDI) problem, as shown in Figure 2. This model involves two primary entities: multiple application vendors, denoted as $\mathrm{AV} = \{\mathrm{AV}_1, \mathrm{AV}_2, \ldots, \mathrm{AV}_m\}$, where m is the total number of vendors, and multiple edge servers, denoted as $\mathrm{ES} = \{\mathrm{ES}_1, \mathrm{ES}_2, \ldots, \mathrm{ES}_n\}$, where n is the total number of edge servers.

Each application vendor maintains information about the edge servers it utilizes, specifically the set S_i for AV_i. Communication between edge servers is facilitated through high-speed channels. Application vendors are considered semi-trusted: while they are expected to execute the protocol honestly, they may attempt cheating attacks by denying audit results and holding edge infrastructure providers (EIPs) accountable. Conversely, edge servers may engage in forge, replay, or replace attacks to conceal data corruption.

A MVMS-EDI audit scheme consists of six steps, outlined as follows:

- $\mathsf{Setup}(\cdot)$: In the initial phase, application vendors deploy smart contracts onto the distributed ledger.
- $\mathsf{Outsource}(D) \rightarrow D'$: Application vendors cache replicas of their original data on edge servers. Vendors also store associated HMAC on edge servers. The vendors and edge servers employ the smart contract SetupSC for achieving consensus on the outsourced data.
- $\mathsf{Audit}(\cdot) \rightarrow C$: An application vendor acts as an auditor by packaging its ID, the set of utilized edge servers, and the number of servers to be audited into an audit request C_i, which is sent to the relevant edge servers.
- $\mathsf{ProveES}(C) \rightarrow (K, P)$: Upon receiving audit requests, edge servers trigger ProveSC, which generates a random key K. This key is broadcast to all nodes in the distributed ledger network. Based on the auditing

strategy, a subset of the least reliable edge servers is selected. These servers compute proofs using K and their stored data replicas. All generated proofs are aggregated into the set P, which is sent to VerifySC.

- $\mathsf{ProveAV}(D, K) \to P'$: After receiving K, application vendors compute proofs using K and their original data. The set P' from application vendors side is then sent to VerifySC.
- $\mathsf{Verify}(P, P') \to \delta/\langle\delta, \Psi\rangle$: Upon receiving P and P', VerifySC verifies P against P' and outputs the verification result $\delta \in \{0, 1\}$. If $\delta = 0$, data corruption is indicated. In such a case, a locating algorithm identifies the corrupted data, and the resulting information Ψ is sent to the relevant edge servers.

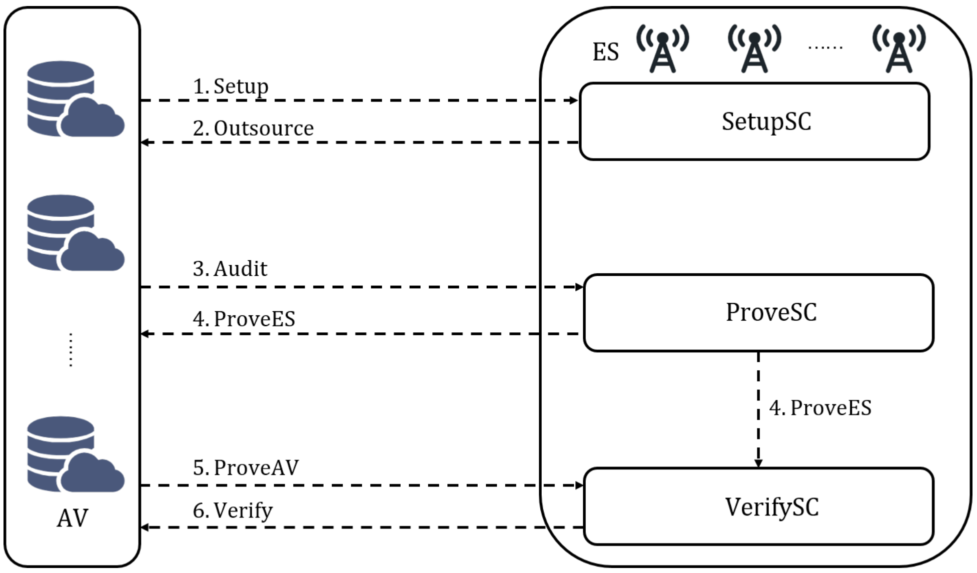


Figure 2. System Model.

3.3. Security Model

In this section, we detail the security model for the proposed scheme. Application vendors are considered semi-trusted entities: they are expected to follow the protocol but may attempt to deny audit results and hold EIPs responsible. Meanwhile, edge servers may attempt to compromise the audit process through forge, replace, or replay attacks.

- Adversarial model: Malicious application vendors are denoted as $\mathcal{A}_{\text{AV}}$, and malicious edge servers as $\mathcal{A}_{\text{ES}}$.
- Operational mechanism: We denote the results of various phases as σ_i, C_i, P_i, P'_i, and δ_i, where $i = 1, 2, \ldots$.
- Adversarial actions: Attackers may forge or modify information pertaining to the audit process, denoted with a superscript "*", e.g., D^* or δ^*.
- Security parameter and polynomial relations: λ is the security parameter; poly(λ) denotes any polynomial function of λ.
- Negligibility: A function $f(\lambda)$ is considered negligible, denoted negl(λ), if for any fixed positive polynomial poly(λ), $f(\lambda) <$ poly(λ) as $\lambda \to \infty$.

First, we analyze the cheating attack performed by malicious application vendor. We denote the probability that $\mathcal{A}_{\text{AV}}$ successfully launching a cheating attack as Adv$[\mathcal{A}_{\text{AV}}]$, as shown in Equation (1).

$$\text{Adv}[\mathcal{A}_{\text{AV}}] = \Pr\left[\begin{array}{ccc} \mathsf{Setup}(\cdot), & & \\ \mathsf{Outsource}(D) \to D', & & \mathcal{A}_{\text{AV}}(D, P_i, \delta_i, \Psi_i) \\ \mathsf{Audit}(\cdot) \to C_i, & \cdot & \text{outputs a forgery} \\ \mathsf{ProveES}(C_i) \to (K_i, P_i) & \cdot & (D^*, P'^*) \\ \mathsf{ProveAV}(D, K) \to P'_i, & & \text{such that } \delta^* = 0 \\ \mathsf{Verify}(P, P') \to \delta_i / < \delta_i, \Psi_i >, & & \end{array}\right] \quad (1)$$

The capabilities of the malicious application vendor $\mathcal{A}_{\text{AV}}$ are illustrated on the left side of ":". The application vendor correctly executes the auditing process. The right side depicts its malicious behavior. By modifying the original data, the application vendor can attempt to pass the $\mathsf{Outsource}$ phase and cause $P'^*_i \neq P_i$, thereby resulting in $\delta^* = 0$.

Next, we analyze the forge attack, replace attack, and replay attack of the malicious edge server $\mathcal{A}_{\text{ES}}$. We unify the aforementioned attack behaviors of the malicious edge server as cheating behaviors. The probability that $\mathcal{A}_{\text{ES}}$

successfully implements a cheat behavior is denoted as Adv$[\mathcal{A}_{\text{ES}}]$, as shown in Equation (2).

$$\text{Adv}[\mathcal{A}_{\text{ES}}] = \Pr\left[\begin{array}{ccc} \mathsf{Setup}(\cdot), & & \\ \mathsf{Outsource}(D) \to D', & & \mathcal{A}_{\text{ES}}(D', P_i, \delta_i, \Psi_i) \\ \mathsf{Audit}(\cdot) \to C_i, & \cdot & \text{outputs a forgery} \\ \mathsf{ProveES}(C) \to P_i, & \cdot & (D'^{*}, P^{*}) \\ \mathsf{ProveAV}(D, K) \to P_i', & & \text{such that } \delta^{*} = 1 \\ \mathsf{Verify}(P, P') \to \delta_i / < \delta_i, \Psi_i >, & & \end{array}\right] \tag{2}$$

The capabilities of a malicious edge server are depicted on the left, while malicious behaviors are shown on the right. The adversary, denoted as $\mathcal{A}_{\text{ES}}$, aims to set $\delta^{*} = 1$ through auditing in scenarios where data replicas have been compromised. To achieve this, $\mathcal{A}_{\text{ES}}$ can employ methods such as constructing D'^{*} or P^{*}. The construction of D'^{*} can be carried out through direct guessing (forge Attack) or by replacing corrupted segments with intact portions (replace attack). Similarly, P^{*} can be fabricated either by direct guessing or by replaying previous auditing processes. In the proposed security model, collusion among edge servers is not considered, because application vendors typically do not rely on a single edge server.

Definition 1. *Assume edge servers do not collude. If Adv$[\mathcal{A}_{AV}] \leq$ negl(λ) and Adv$[\mathcal{A}_{ES}] \leq$ negl(λ), then the MVMS-EDI scheme is secure.*

3.4. Design Requirement

To enhance the practicality of the proposed scheme, we outline the design requirements and objectives as follows:

- Correct: The proposed scheme must accurately perform EDI Auditing. When data replicas are corrupted, it should correctly detect the corruption and notify the application vendor.
- Secure: The proposed scheme must be secure, effectively resisting cheating attacks from dishonest application vendors as well as forge, replace, and replay attacks from dishonest edge servers, thereby safeguarding the interests of both the application vendor and the EIP. In MVMS scenario, how to defend the cheating attack from malicious application vendors and how to make malicious edge servers forge-proof in polynomial time are challenges to be solved.
- Efficient: The proposed scheme must be efficient, with low and acceptable computation cost and communication cost. In MVMS scenario, how to minimize the computation cost and communication cost when multiple application vendors launch integrity auditing is also a challenge.

4. The Proposed Scheme

4.1. Main Idea

To address the MVMS-EDI problem, we design our proposed scheme based on recent advancements, specifically building upon the MVMS-SC scheme introduced in [8]. While MVMS-SC can perform MVMS-EDI auditing correctly and efficiently, it falls short in addressing cheating attacks by malicious application vendors and forgery attacks by malicious edge servers.

The MVMS-SC scheme presents two primary security concerns. First, a malicious edge server can pass an audit without actually storing the original data replicas. Specifically, the edge server can substitute the data replica with only its hash digest, thereby significantly reducing its storage requirements. As shown in Equation (12) of MVMS-SC [8], the malicious edge server can still pass the audit because only the digest is verified. Second, a malicious application vendor can exploit the system by denying the proofs calculated by edge servers. In particular, the vendor can slightly modify the original data, causing the digest to change and resulting in verification failure, as the digests of the original data and its replicas would no longer match.

Motivated by these limitations, we propose the MVMS-HMAC scheme to achieve correct, efficient, and secure MVMS-EDI auditing.To resolve the first security concern, we change the proof computation method from a simple hash to a keyed hash value using HMAC. Computing the HMAC of a data replica requires a secret key, which is randomly generated in each auditing round, resulting in different HMAC values for each round. Therefore, edge servers must retain the actual data replica to generate valid proofs and ensure data integrity. To address the second concern, MVMS-HMAC enables the distributed ledger (DL) network to reach consensus during the $\mathsf{Outsource}$ phase when application vendors cache data replicas to edge servers. A smart contract, SetupSC, is employed to facilitate this process. When an application vendor caches a new data replica, it must compute the corresponding digest and send it to the edge server as a data commitment. SetupSC is then triggered, prompting the network to reach consensus by verifying the digest of the data replica. Once consensus is achieved, the original data remains unchanged until it is explicitly updated. Consequently, a malicious application vendor cannot exploit the platform by subsequently altering the original data.

The proposed scheme operates as follows: First, application vendors execute the Setup phase to deploy the smart contract on the DL network. Next, application vendors cache the replica of the original data on the edge server, along with the associated cryptographic hash digest, and triggers SetupSC to initiate consensus. Subsequently, when an application vendor wishes to initiate an integrity audit, it executes the Audit phase, generates a challenge, and sends it to the edge server. Upon receiving the challenge, the ProveSC contract is triggered to execute the ProveES phase. In this phase, ProveSC first generates a cryptographic key and broadcasts it to all nodes in the DL network. Upon receiving the key, edge servers compute the HMAC values of their stored data replicas and combine these values, along with the IDs of the application vendors and their own IDs, to construct proofs. All proofs are aggregated into a set, and the DL network reaches consensus on this set. Meanwhile, in the ProveAV phase, after receiving the key, application vendors calculate the HMAC values of their original data and combine these with their own IDs to generate their proofs. The sets of proofs from ProveES and ProveAV are both sent to the VerifySC contract. Upon receiving both proof sets, VerifySC verifies the edge servers' proofs against those from the application vendors. If verification succeeds, the Verify phase concludes; otherwise, the protocol proceeds to locate the corrupted data.

From a security perspective, the keyed digest using HMAC mitigates forgery, replacement, and replay attacks by malicious edge servers. Furthermore, during the Setup phase, the scheme validates the application vendor's data to defend against cheating attacks from malicious vendors. To exclude malicious application vendors from the verification process, we adopt the smart contract approach on a distributed ledger as proposed in [8]. The Setup, Prove, and Verify phases are all executed by their respective smart contracts.

Regarding efficiency, MVMS-HMAC utilizes the selection algorithm from MVMS-SC [8] to selectively audit the edge servers. To further reduce computational overhead, the scheme uses HMAC-based proof computation during the ProveES phase. Additionally, in the Verify phase, aggregate verification is performed first, and localization is conducted only if aggregate verification fails. To minimize communication cost, both the edge server and the application vendor batch proofs before transmission, thereby reducing the volume of data exchanged during the auditing process.

4.2. Scheme Details

Setup. The Setup phase initializes the scheme. During this phase, application vendors deploy the smart contracts on the distributed ledger (DL) network. SetupSC contract is used to achieve consensus on the original data, ProveSC contract is responsible for calculating proofs on the edge servers, and VerifySC contract is used to verify the proofs from edge servers against those from application vendors, as well as to locate any corrupted data.

Outsource. The Outsource phase is crucial for preventing cheating attacks by malicious application vendors. It enables edge servers to verify the integrity of the original data. The detailed procedure for Outsource is provided in Algorithm 1. In this phase, SetupSC first generates a random key K_s. The algorithm then iterates over each AV$_i$ in AV. For each AV$_i$, a replica of the original data d_i is created, denoted as d'_i, and the HMAC of d_i is computed using K_s. The set of all data replicas is denoted as D', and all resulting HMAC values are aggregated into a set H. Application vendors cache d'_i and transmit the corresponding HMACs to the smart contract. Subsequently, the DL network reaches consensus on H using a consensus algorithm. Edge servers independently compute the HMACs of their received d'_i and compare them with the provided HMACs to verify the authenticity of the original data. If the HMAC values are not consistent, abort the protocol. We note that the H serves as an agreement of the outsourced data for both the application vendors and edge servers.

Audit and ProveES. Algorithm 2 presents the details. To check the integrity of outsourced data in edge servers, application vendors execute the Audit phase. Each AV_i determine the set of edge servers S'_i to be audited based on its data replica set S_i and the detailed audited data blocks b_i. The detailed selection function is the same as that in [8]). All S'_i are merged into S'. Each AV$_i$ constructs its own C_i by packaging its ID$_{\text{AV}_i}$, S_i, as well as the fraction of challenged servers for AV$_i$, i.e., $b_i = S'_i/S_i$. The collection $C = \{C_1, C_2, \ldots, C_m\}$ is sent to activate ProveSC contract.

On receiving an audit challenge, the edge servers run the ProveES algorithm. In ProveES, a fresh cryptographic key K is then randomly generated according to the security parameter λ and broadcast to every s'_j in S'. K

is distributed to all nodes in the DL network. Each s'_j in S' computes r_{ji} for each data replica d'_i as follows:

$$r_{ji} = \mathsf{HMAC}(d'_i, K) \tag{3}$$

Each p_j is constructed from the set of r_{ji}, the $\mathrm{ID}_{\mathrm{AV}_i}$ associated with d'_i, and $\mathrm{ID}_{s'_j}$. The set p_j is broadcast to the DL network, and consensus among edge servers is achieved via the consensus algorithm.

$\mathsf{ProveAV}$. In this phase, after receiving the key K broadcast by ProveSC contract, each AV_i computes r'_i for d_i locally as follows:

$$r'_i = \mathsf{HMAC}(d_i, K) \tag{4}$$

Each AV_i constructs p'_i by combining r'_i with its $\mathrm{ID}_{\mathrm{AV}_i}$. All p'_i are assembled into P', which is then sent to VerifySC contract. Algorithm 3 shows the steps.

Algorithm 1 Outsource

```
Require: $D$
Ensure: $D'$
1: $K_s \leftarrow \mathsf{Random}(1^\lambda)$
2: for $\mathrm{AV}_i \in \mathrm{AV}$ do
3:     $d'_i \leftarrow d_i$
4:     $D' \leftarrow D' \cup d'_i$
5:     $H \leftarrow H \cup \mathsf{HMAC}(d_i, K_s)$
6: end for
7: Send $D'$ and $H$ to edge servers.
8: The DL network reaches consensus on $H$.
9: return $D'$
```

Algorithm 2 Audit and ProveES

```
Require: $C$
Ensure: $P$
1: for $\mathrm{AV}_i \in \mathrm{AV}$ do
2:     $S'_i$ is selected via the algorithm in [8]
3:     $S' \leftarrow S' \cup S'_i$
4: end for
5: $K \leftarrow \mathsf{Random}(1^\lambda)$
6: $S'$ receives $K$
7: for $s'_j \in S'$ do
8:     for $d'_i \in D'_j$ do
9:         $r_{ji} \leftarrow \mathsf{HMAC}(d'_i, K)$
10:    end for
11:    $p_j = \{\mathrm{ID}_{s'_j}, \{r_{ji}, \mathrm{ID}_{\mathrm{AV}_i}\} \mid d'_i \in D'_j\}$
12:    Broadcast $p_j$ to the DL network
13: end for
14: The DL network reaches consensus on $P = \{p_j\}$.
15: return $P$
```

Verify. Algorithm 4 outlines the specific details of the Verify process. Initially, VerifySC selects a large prime γ based on the security parameter λ. VerifySC then computes δ by comparing two aggregated sums modulo γ as in Equation (5).

$$\delta \leftarrow \sum_{i=1}^{|P'|} |S'_i| \times r'_i \bmod \gamma \overset{?}{=} \sum_{j=1}^{|P|} \sum_{r_{ji} \in p_j} r_{ji} \bmod \gamma \tag{5}$$

The left-hand side aggregates values from P' by summing, for each AV_i, the product of $|S'_i|$ and the corresponding r'_i. The right-hand side aggregates all r_{ji} from P. Here, γ is a large prime number determined by λ. This modular operation ensures the result remains within a fixed size. If $\delta = 1$, verification passes; if $\delta = 0$, data corruption is detected.

Algorithm 3 ProveAV

```
Require: $K$
Ensure: $P'$
 1: for $\text{AV}_i \in \text{AV}$ do
 2:     $r'_i \leftarrow \mathsf{HMAC}(d_i, K)$
 3:     $p'_i = \{r'_i, \text{ID}_{\text{AV}_i}\}$
 4:     $P' \leftarrow P' \cup p'_i$
 5: end for
 6: return $P'$
```

In the case of corruption, the VerifySC further compares P and P' to identify the corrupted data set Ψ. An empty set Ψ is initialized. For each p_j in P, VerifySC inspects every pair $\{r_{ji}, \text{ID}_{\text{AV}_i}\}$, comparing r_{ji} with the corresponding r'_i. If $r_{ji} \neq r'_i$, the data replica d'_{ji} cached by AV_i in s'_j is identified as corrupted. The IDs of AV_i and s'_j are added to Ψ. Finally, both δ and Ψ are returned to the application vendors. We note that if some application vendor and edge server do not agree with the auditing result, the application vendor could prove its correctness by showing the original data.

Algorithm 4 Verify

```
Require: $P$, $P'$
Ensure: $\delta$, $\Psi$
 1: Select a large prime $\gamma$
 2: $\delta \leftarrow \sum_{i=1}^{|P'|} |S'_i| \times r'_i \bmod \gamma \overset{?}{=} \sum_{j=1}^{|P|} \sum_{r_{ji} \in p_j} r_{ji} \bmod \gamma$
 3: if $\delta = 1$ then
 4:     return $\delta$
 5: else
 6:     $\Psi \leftarrow \emptyset$
 7:     for $p_j \in P$ do
 8:         for each $\{r_{ji}, \text{ID}_{\text{AV}_i}\}$ in $p_j$ do
 9:             if $r_{ji} \neq r'_i$ then
10:                 $\Psi \leftarrow \Psi \cup \{\text{ID}_{\text{AV}_i}, p_j.\text{ID}_{s'_j}\}$
11:             end if
12:         end for
13:     end for
14:     if dispute exists then
15:         AV and ES resolves the dispute by employing the initial data commitment $H$
16:     end if
17:     return $\delta$, $\Psi$
18: end if
```

Finally, we also note that the proposed scheme also supports data dynamics straight-forwardly. The main reason is that the outsourcing step does not involve any authentication tag computation that was required by traditional approaches. For a data update operation, the application vendor just needs to spread the update to the edge clouds and agrees on a commitment of the newest data on the smart contract. We assumed that the edge clouds are not colluded. For possible collusion in the extreme case, the application vendor may use a Merkle authentication tree to compute the commitment. With such setting, the application vendor could audit the edge cloud by randomly challenging the data blocks on the leaf nodes of the Merkle authentication tree. This approach will detect data damaged even the edge clouds collude with each other. Compared with the proposed scheme, this adaption incurs additional performance overhead; it may serve for more conservative users that need to handle collusion.

5. Analysis

5.1. Correctness

The correctness of the proposed audit scheme is straight-forward. Suppose all parties are honest. For intact outsourced data, both sides of Equation (5) are the same. The verification is correct. For some damaged data, however, the two sides of Equation (5) are different before modulo the random prime γ. After taking the modular operation, Equation (5) holds with probability $\frac{1}{\gamma}$, which is negligible. Therefore, the proposed scheme is correct.

5.2. Security

Based on the security model defined in Section 3.3, we further provide formal cryptographic reductions demonstrating that the security of MVMS-HMAC relies on the fundamental security properties of HMAC. We consider two types of adversaries: malicious application vendors, denoted as $\mathcal{A}_{\text{AV}}$, who attempt cheating attacks, and malicious edge servers, denoted as $\mathcal{A}_{\text{ES}}$, who may launch forge, replace, and replay attacks.

We first recall the security properties of HMAC that underpin our scheme. HMAC is a secure pseudorandom function (PRF) if, for all probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$, the advantage in distinguishing HMAC from a truly random function is negligible, i.e.,

$$\text{Adv}^{\text{PRF}}_{\text{HMAC}}(\mathcal{A}) = \left|\Pr[\mathcal{A}^{\text{HMAC}_K(\cdot)} = 1] - \Pr[\mathcal{A}^{\text{Random}(\cdot)} = 1]\right| = \text{negl}(\lambda) \tag{6}$$

where $K \leftarrow \{0,1\}^\lambda$ is uniformly sampled and R denotes a truly random function. HMAC is collision-resistant if, for any PPT adversary $\mathcal{A}$,

$$\Pr[(m_1, m_2) \leftarrow \mathcal{A}(1^\lambda) : m_1 \neq m_2 \wedge \mathsf{HMAC}(K, m_1) = \mathsf{HMAC}(K, m_2)] = \text{negl}(\lambda) \tag{7}$$

Security Against Malicious Application Vendors. Before detailing the security argument, we first discuss security intuition. Suppose the edge servers are honest and well stores the outsourced data and some application vendor cheats on data d_i. Then the returned proof p_i is correct. If some application vendor cheats, the submitted proof $p_i^{'}$ must be different from p_i. Note that the data behind p_i is authenticated when outsourcing the data. In order to cheat on the data behind $p_i^{'}$, the application vendor must find a cryptographic hash collision. Therefore, the two compromise a collision for the HMAC function.

Theorem 1. *Suppose HMAC is collision resistant and edge servers are honest. Then MVMS-HMAC is secure against malicious application vendors, as formalized in Definition 1.*

Proof. We construct a reduction algorithm $\mathcal{B}$ that transforms any successful adversary $\mathcal{A}_{\text{AV}}$ conducting a cheating attack into an adversary against the collision security of HMAC.

Given the target HMAC algorithm for adversary $\mathcal{B}$, let Adv$[\mathcal{B}]$ be the probability of successfully finding a collision. We plug the attacked HMAC into the proposed scheme and runs $\mathcal{A}_{\text{AV}}$ to find a forgery from some application vendor. Specifically, let $d_i, d_i^{'}, p_i, p_i^{'}$ be the original data, proof from the edge server, and proof from the application vendor, returned by the adversary $\mathcal{A}_{\text{AV}}$ as a forgery. In the outsourcing phase, the edge servers have confirmed that the two data copies are consistent using the HMAC digest. Because the adversary $\mathcal{A}_{\text{AV}}$ succeeds in cheating by holding the edge server as accountable for data loss, we have $p_i \neq p_i^{'}$. Thus it holds that $d_i \neq d_i^{'}$. However, they have the same HMAC digest; the adversary $\mathcal{B}$ then returns $\{d_i, d_i^{'}\}$ as the found collision.

As a result, we have Adv$[\mathcal{B}] \geq$ Adv$[\mathcal{A}_{\text{AV}}]$. If Adv$[\mathcal{A}_{\text{AV}}]$ is not negligible, we have Adv$[\mathcal{B}]$ is not negligible neither, contradicting with the collision resistance assumption of HMAC. □

Security Against Malicious Edge Servers. For malicious edge servers, we build security upon the pseudorandomness of HMAC. The security intuition is as follows. If HMAC is truly random, we argue that the collision probability is negligible. It reduces to a simpler problem, i.e., analyzing the collision probability of a given value with a random value.

Theorem 2. *If HMAC is a secure PRF, then MVMS-HMAC is secure against malicious edge servers, as formalized in Definition 1.*

Proof. We use a sequence of game argument.

Game 0. This is the original security game as in Definition 1. Let Adv$[\mathcal{A}_E S]$ be the probability of successfully cheating the application vendors on some corrupted data.

Game 1. This game is almost the same as Game 0 except that HMAC is replaced with a truly random function. Let Adv$_1[\mathcal{A}_E S]$ be the cheating probability. Based on the pseudorandomness of HMAC, we have

$$|\text{Adv}[\mathcal{A}_{\text{ES}}] - \text{Adv}_1[\mathcal{A}_{\text{ES}}]| \leq \text{Adv}^{\text{PRF}}_{\text{HMAC}}. \tag{8}$$

Now we analyze the probability of Adv$_1[\mathcal{A}_{\text{ES}}]$. Suppose the challenged data block d_i on edge server j is damaged. Then the j-th edge sever needs to find a value r_{ji} to pass the verification equation, i.e. Equation 5. No matter what strategies the edge server employ to generate r_{ji}, since the left side of Equation 5 is random, the probability that Equation 5 holds is $\frac{1}{\gamma}$. Thus, we have

$$\mathrm{Adv}_1[\mathcal{A}_{\mathrm{ES}} = \frac{1}{\gamma}]. \tag{9}$$

Combing Equations (8) and (9), we have $\mathrm{Adv}[\mathcal{A}_{\mathrm{ES}}] \leq \mathrm{Adv}^{\mathrm{PRF}}_{\mathrm{HMAC}} + \frac{1}{\gamma}$, which is negligible. Therefore, the proposed scheme is secure against malicious edge servers. □

5.3. Efficiency

We analyze the efficiency of each phase with respect to both computational and communication cost. Computation cost is expressed in terms of time complexity. Communication cost is measured by the size of transmitted data.

- Computation cost: The Outsource, Audit, and ProveES phases primarily involve data transmission and thus are not computationally intensive. In the Outsource phase, after SetupSC sends K_s to the application vendor, each vendor computes the HMAC of its original data. Because the data size is fixed and vendors compute in parallel, the time complexity for each vendor is $O(1)$. Verification by all edge servers incurs a complexity of $O(n)$. In the ProveES phase, selecting edge servers and computing proofs together require $O(m)$ time. In the ProveAV phase, each application vendor computes r' and constructs p', both in $O(1)$ time. The Verify phase consists of aggregate verification and potentially localization, each with complexity $O(m)$. Notably, as edge servers communicate over high-speed channels, aggregation can be performed efficiently in advance, further improving the practical efficiency of Verify.
- Communication cost: In the Outsource phase, SetupSC distributes a key K_s of length λ bits to each vendor, who then uploads an HMAC value of approximately 2λ bits. Thus, the total communication cost is about $3m\lambda$ bits. In the Audit phase, each challenge includes $\mathrm{ID}_{\mathrm{AV}_i}$, S_i, and b_i. The cost is in $O(mn)$ bits. In the ProveES phase, the key K has length λ bits. Each p_j consists of $\mathrm{ID}_{s'_j}$, r_{ji}, and $\mathrm{ID}_{\mathrm{AV}_i}$, with the total size for P being $O(nm\lambda)$ bits. Owing to high-speed channels between edge servers, this is acceptable in practice. In the ProveAV phase, each p'_i includes $\mathrm{ID}_{\mathrm{AV}_i}$ and r'_i, the cost is $O(m\lambda)$ for all vendors. In the Verify phase, if no corruption is detected, only δ (1 bit) is returned. If corruption is detected, additional information Ψ must be sent. If there are ψ corrupted replicas, this adds $O(m\psi)$ bits to the communication cost.

6. Experimental Evaluation

6.1. Setup

We implemented a prototype of MVMS-HMAC in Python 3.10 to evaluate its practical performance. The source code is publicly available at [14]. To achieve a balance between security and efficiency, we set the security parameter $\lambda = 128$ in our implementation. Additionally, we employed SHA-256 as the underlying hash function for HMAC computation to meet the 128-bit security level requirement.

For our experimental evaluation of MVMS-HMAC, we varied four key parameters: the number of application vendors (m), the number of edge servers (n), the data size (DataSize), and the auditing rate (b). Here, the auditing rate refers to the proportion of edge servers each application vendor audits in each round. For clarity, we set the auditing rate to a uniform value across all experiments. To comprehensively assess the robustness of our scheme, we designed four experimental scenarios (as in Table 2):

- Setting I: m varies from 5 to 100, while n, DataSize, and b are fixed at 64, 64 MB, and 40%, respectively.
- Setting II: n varies from 16 to 1024, with m, DataSize, and b fixed at 25, 64 MB, and 40%, respectively.
- Setting III: DataSize varies from 16 MB to 512 MB, while m, n, and b are fixed at 25, 64, and 40%, respectively.
- Setting IV: b varies from 20% to 80%, with m, n, and DataSize fixed at 25, 64, and 64 MB, respectively.

In our simulated environment, we ignored network effects and focused mainly on computational processing. Since both application vendors and edge servers perform tasks in parallel, we used the average processing time to represent the per-entity computational overhead. The datasets used in the experiments were randomly generated. All experiments were conducted on a PC equipped with an Intel i7-12700 CPU and 16GB RAM. Each experiment was repeated 50 times, and the average value was reported to ensure reliability.

Table 2. Dataset and Outsource Phase Computation Cost.

	m	n	**DataSize**	b	**Outsource (ms)**
Setting I	5	64	64 MB	40%	235
	10				472.1
	25				1180.95
	50				2315.3
	100				4610.5
Setting II	25		64 MB	40%	1172.25
		32			1162.38
		64			1180.95
	25	128			1158.26
		256			1181.58
		512			1154.58
		1024			1171.99
Setting III	25	64	16 MB	40%	294.9
			32 MB		586.08
			64MB		1180.95
			128 MB		2336.91
			256 MB		4670.45
			512 MB		9216.3
Setting IV	25	64	64MB	20%	1172.41
				40%	1180.95
				60%	1167.82
				80%	1165.36

6.2. Results

6.2.1. Computation Cost

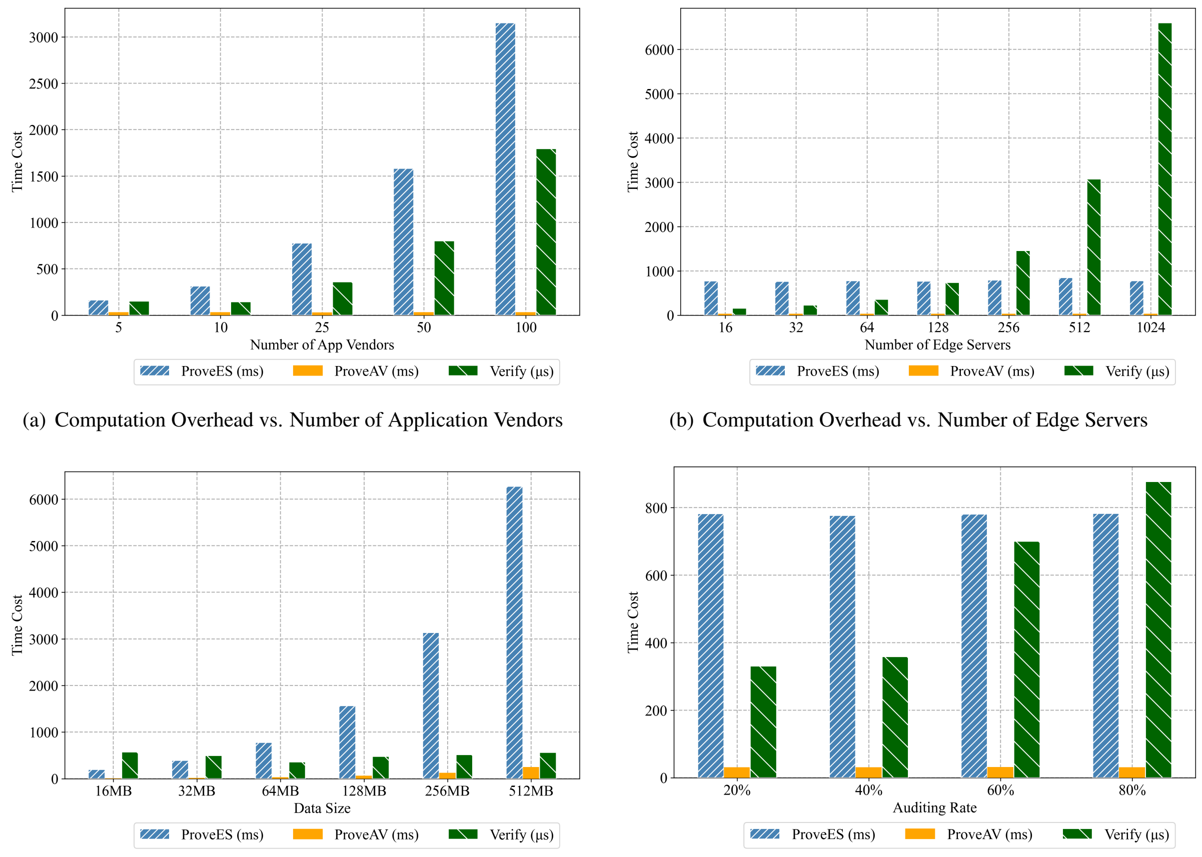
Outsource. This phase is the initial stage of the scheme, where application vendors cache data replicas and compute the HMAC values of each original data item, broadcasting these to edge servers for verification. In our experiments, we simulated the consensus algorithm and measured only the computational time. Table 2 presents the time consumed.

Since both HMAC computation by vendors and verification by edge servers are performed in parallel, increasing the number of application vendors directly leads to an increase in overall data volume. Consequently, the computational cost of this process is positively correlated with the number of application vendors (m), but is independent of the number of edge servers (n). This phase is not part of routine auditing and does not involve the auditing rate (b). Additionally, the HMAC computation time increases with the size of the data (DataSize). In Setting I, the time increases from 235 ms to 4610.5 ms as m grows, while in Setting III, the time rises to 9216.3 ms as the data size reaches 512 MB. In Settings II and IV, the time consumption remains stable, averaging around 1169.87 ms. Since this phase is only executed periodically or upon data updates, its computational overhead is acceptable.

ProveES. The primary computational task in this phase is for each edge server to compute the proof for its stored data replicas. The computation depends on both the number of data replicas per server and the data size. Thus, as m and DataSize increase, the computation time rises, as shown in Figure 3a,c. For $m = 100$, the ProveES phase requires 3153.59 ms. Similarly, increasing DataSize from 16 MB to 512 MB results in a time increase from 196.58 ms to 6276.46 ms. Since proof computation is parallelized, the time is largely independent of n and b. Figure 3b,d show that as n and b increase, the time remains stable at approximately 783.85 ms. Overall, this computational cost is reasonable.

ProveAV. In this phase, each application vendor computes the proof r' for its original data. The computation time for this phase is linearly related to the size of the original data, as shown in Figure 3c. When DataSize increases from 16 MB to 512 MB, the time required rises by only 243.23 ms. In other scenarios, the time remains stable, averaging 31.52 ms. Thus, the computational overhead of this phase is quite small.

Verify. The computational cost of the Verify phase is illustrated in Figure 3. This phase aggregates and verifies the proofs P' from vendors and P from edge servers. The computation is mainly influenced by the number of application vendors (m), the number of edge servers (n), and the auditing rate (b). Figure 3a shows that in Setting I, the verification time increases with m, reaching only 1796.6 µs when $m = 100$. In Setting II, as n increases from 16 to 1024, the time rises from 156.42 µs to 6600.41 µs (Figure 3b). Setting III demonstrates that verification time remains stable at approximately 498.46 µs as DataSize increases (Figure 3c). In Setting IV, as b increases, the verification time rises modestly by 546.64 µs (Figure 3d). In all cases, the overhead is reasonably small and acceptable.



(a) Computation Overhead vs. Number of Application Vendors

(b) Computation Overhead vs. Number of Edge Servers

(c) Computation Overhead vs. Data Size

(d) Computation Overhead vs. Auditing Rate

Figure 3. Auditing Cost.

6.2.2. Communication Cost

As discussed in Section 5, we have theoretically analyzed the communication cost, which we now further validate using our prototype. With a security level of 128 bits, the lengths of K_s and K are 16 bytes, and each HMAC output is 32 bytes. The largest communication overhead for each application vendor and the smart contract occurs during the Audit phase, when sending the challenge to ProveSC. For $n = 1024$, this cost is 2052 bytes, which is well within acceptable limits for modern networks. In the ProveES phase, each edge server also broadcasts its proof p. For $n = 100$, the size of p is 3400 bytes. As edge server communication occurs over high-speed channels, this cost is also acceptable.

6.3. Comparison with Existing Research

We compare MVMS-HMAC to the recent MVMS-SC scheme [8], which addresses the MVMS-EDI problem and effectively counters replay attacks from malicious edge servers. However, MVMS-SC is less effective against cheating attacks from malicious vendors and against replace and forge attacks from malicious edge servers. As the MVMS-SC prototype is not open-sourced, we use reported data from [8] for comparison. Under a scenario with 100 application vendors, 1024 edge servers, 512 MB data size, and 80% auditing rate, the auditing times for MVMS-SC [8] are approximately 3 s, 0.8 s, 8 s, and 0.8 s, respectively (see Figure 5 in [8]). In terms of communication cost, the primary factor is the prime number γ, which is 128 bytes for a 1024-bit prime. In contrast, MVMS-HMAC's total execution times are 3.4 s, 0.8 s, 6.5 s, and 0.8 s for the same scenario, with a communication cost of 2052 bytes at $n = 1024$ with the same setting. The performance is comparable. However, regarding security, MVMS-HMAC effectively resists cheating attacks from malicious vendors and forge, replay, and replace attacks from malicious edge servers. Overall, MVMS-HMAC demonstrates improved security with comparable performance.

7. Conclusions

In this paper, we investigated the edge data integrity checking problem in a multi-vendor, multi-server context. We provided a comprehensive problem analysis and modeling, presented a detailed security model, and proposed a novel edge data integrity auditing scheme, MVMS-HMAC, capable of addressing these challenges. We introduced MVMS-HMAC, an auditing scheme tailored to the MVMS-EDI scenario, which effectively counters the identified attack vectors and achieves accurate, secure, and efficient auditing. While the proposed scheme works theoretically, one main limitation of the proposed scheme remains, i.e., large scale of real world evaluation could be further conducted and analyzed. Given the limited research on the MVMS-EDI problem to date, we believe that the proposed scheme offers valuable insights and advances the state-of-the-art in edge data integrity management.

Author Contributions

Z.Q.: conceptualization, software, writing; H.M.: data duration, investigation, validation; and F.C.: conceptualization, methodology, investigation, writing. All authors have read and agreed to the published version of the manuscript.

Funding

This research was supported in part by the National Natural Science Foundation of China under Grant 62272318 and in part by Shenzhen Science and Technology Program under Grant JCYJ20220531102802005.

Institutional Review Board Statement

Not applicable.

Informed Consent Statement

Not applicable.

Data Availability Statement

Not applicable.

Conflicts of Interest

The authors declare no conflict of interest.

Use of AI and AI-assisted Technologies

No AI tools were utilized for this paper.

References

1. Li, H.; Duan, L. Theory of mixture-of-experts for mobile edge computing. In Proceedings of the IEEE INFOCOM 2025—IEEE Conference on Computer Communications, London, UK, 19–22 May 2025; pp. 1–10.
2. Satyanarayanan, M. The emergence of edge computing. *Computer* **2017**, *50*, 30–39.
3. Mao, Y.; You, C.; Zhang, J.; et al. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Commun. Surv. Tutorials* **2017**, *19*, 2322–2358.
4. Liu, F.; Tang, G.; Li, Y.; et al. A Survey on Edge Computing Systems and Tools. *Proc. IEEE* **2019**, *107*, 1537–1562.
5. EdgeOne. Game Release. Tencent. Available online: https://edgeone.ai/solutions/gaming?source=666 (accessed on 27 July 2025).
6. Xiao, Y.; Jia, Y.; Liu, C.; et al. Edge computing security: State of the art and challenges. *Proc. IEEE* **2019**, *107*, 1608–1631.
7. Zhao, Y.; Qu, Y.; Xiang, Y.; et al. A Comprehensive Survey on Edge Data Integrity Verification: Fundamentals and Future Trends. *Acm Comput. Surv.* **2024**, *57*, 1–34.
8. Zhao, Y.; Qu, Y.; Chen, F.; et al. Data Integrity Verification in Mobile Edge Computing with Multi-Vendor and Multi-Server. *IEEE Trans. Mob. Comput.* **2024**, *23*, 5418–5432.
9. Yao, Y.; Chang, J.; Zhang, A. Efficient data sharing scheme with fine-grained access control and integrity auditing in terminal–edge–cloud network. *IEEE Internet Things J.* **2024**, *11*, 26944–26954.
10. Li, Y.; Shen, J.; Ji, S.; et al. Blockchain-based data integrity verification scheme in AIoT cloud–edge computing environment. *IEEE Trans. Eng. Manag.* **2023**, *71*, 12556–12565.

11. Li, B.; He, Q.; Chen, F.; et al. Cooperative Assurance of Cache Data Integrity for Mobile Edge Computing. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 4648–4662.
12. Li, B.; He, Q.; Chen, F.; et al. Inspecting Edge Data Integrity with Aggregate Signature in Distributed Edge Computing Environment. *IEEE Trans. Cloud Comput.* **2021**, *10*, 2691–2703.
13. Li, B.; He, Q.; Chen, F.; et al. Auditing Cache Data Integrity in The Edge Computing Environment. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 1210–1223.
14. Qin, Z. Souce Code. GitHub, 2025. Available online: https://github.com/szu-security-group/MVMS-HMAC (accessed on 27 July 2025).
15. Ateniese, G.; Burns, R.; Curtmola, R.; et al. Provable Data Possession at Untrusted Stores. In Proceedings of the 14th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 29 October–2 November 2007; pp. 598–609.
16. Juels, A.; Kaliski Jr., B.S. PORs: Proofs of Retrievability for Large Files. In Proceedings of the 14th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 29 October–2 November 2007; pp. 584–597.
17. Chen, F.; Cai, J.; Xiang, T.; et al. Practical Cloud Storage Auditing Using Serverless Computing. *Sci. China Inf. Sci.* **2024**, *67*, 132102.
18. Chen, F.; Meng, F.; Li, Z.; et al. Public Cloud Object Storage Auditing: Design, Implementation, and Analysis. *J. Parallel Distrib. Comput.* **2024**, *189*, 104870.
19. Wang, Q.; Wang, C.; Li, J.; et al. Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing. In *European Symposium on Research in Computer Security*; Springer: Berlin/Heidelberg, Germany, 2009, pp. 355–370.
20. Li, T.; Chu, J.; Hu, L. CIA: A Collaborative Integrity Auditing Scheme for Cloud Data with Multi-Replica on Multi-Cloud Storage Providers. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *34*, 154–162.
21. Li, J.; Yan, H.; Zhang, Y. Efficient Identity-Based Provable Multi-Copy Data Possession in Multi-Cloud Storage. *IEEE Trans. Cloud Comput.* **2019**, *10*, 356–365.
22. Tong, W.; Jiang, B.; Xu, F.; et al. Privacy-Preserving Data Integrity Verification in Mobile Edge Computing. In Proceedings of the IEEE 39th International Conference on Distributed Computing Systems. IEEE, Dallas, TX, USA, 7–10 July 2019; pp. 1007–1018.
23. Cui, G.; He, Q.; Li, B.; et al. Efficient Verification of Edge Data Integrity in Edge Computing Environment. *IEEE Trans. Serv. Comput.* **2021**, *15*, 3233–3244.
24. Ding, Y.; Li, Y.; Yang, W.; et al. Edge Data Integrity Verification Scheme Supporting Data Dynamics and Batch Auditing. *J. Syst. Archit.* **2022**, *128*, 102560.
25. Li, B.; He, Q.; Yuan, L.; et al. EdgeWatch: Collaborative Investigation of Data Integrity at The Edge Based on Blockchain. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, 14–18 August 2022; pp. 3208–3218.
26. Tong, W.; Chen, W.; Jiang, B.; et al. Privacy-Preserving Data Integrity Verification for Secure Mobile Edge Storage. *IEEE Trans. Mob. Comput.* **2022**, *22*, 5463–5478.
27. Wang, L.; Hu, M.; Jia, Z.; et al. SStore: an efficient and secure provable data auditing platform for cloud. *IEEE Trans. Inf. Forensics Secur.* **2024**, *19*, 4572–4584.
28. Zhou, H.; Shen, W.; Liu, J. Certificate-based multi-copy cloud storage auditing supporting data dynamics. *Comput. Secur.* **2025**, *148*, 104096.
29. Yang, Y.; Chen, Y.; Xiong, P.; Chen, F.; Chen, J. Decentralized Self-Auditing Multiple Cloud Storage in Compressed Provable Data Possession. *IEEE Trans. Dependable Secur. Comput.* **2025**, *22*, 3901–3915.
30. Islam, M.R.; Xiang, Y.; Uddin, M.P.; Zhao, Y.; Kua, J.; Gao, L. Multiple Edge Data Integrity Verification With Multi-Vendors and Multi-Servers in Mobile Edge Computing. *IEEE Trans. Mob. Comput.* **2025**, *24*, 4668–4683.
31. Xia, X.; Chen, F.; He, Q.; Grundy, J.; Abdelrazek, M.; Jin, H. Online Collaborative Data Caching in Edge Computing. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 281–294.
32. Xia, X.; Chen, F.; He, Q.; Grundy, J.C.; Abdelrazek, M.; Jin, H. Cost-Effective App Data Distribution in Edge Computing. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 31–44.