# PoTR: Accurate and Efficient Proof of Timely-Retrievability for Storage Systems

Cláudio Correia Rita Prates Miguel Correia Luís Rodrigues
*INESC-ID, Instituto Superior Técnico, Universidade de Lisboa*
{claudio.correia, rita.prates, miguel.p.correia, ler}@tecnico.ulisboa.pt

***Abstract*—The use of remote storage has become prevalent both by organizations and individuals. By relying on third-party storage, such as cloud or peer-to-peer storage services, availability, fault tolerance, and low access latency can be attained in a cost-efficient manner. Unfortunately, storage providers may misbehave and violate Service-Level Agreements (SLAs). In this paper, we propose, implement and evaluate a new *Proof of Timely-Retrievability* (PoTR) that aims at assessing whether a provider is able to retrieve data objects with a latency lower than some SLA-specific threshold $\delta$. We leverage Trusted Execution Environments (e.g., Intel SGX) to ensure that the proof is produced by the node being audited and to reduce the communication between the auditor and the audited node. We have experimentally evaluated our design considering a challenging edge computing setting, where storage services are provided by resource-constrained fog nodes, and the distance between the auditor and the audited node can be large. Despite the variance in edge network delays, we show that the auditor is able to effectively detect SLA violations.**

***Index Terms*—edge storage, SLA violations, auditing, proofs of storage**

| Systems | Data Locality | Efficient/Cheap Execution | Challenge Delegation Protection | Timer Delay Protection |
|---|---|---|---|---|
| *Single Remote Auditor* | | | | |
| ***PoTR*** | ✓ | ✓ | ✓ | ✓ |
| PDP (2007) [14] | ✗ | ✓ | ✗ | ✓ |
| PoRet (2016) [15] | ✗ | ✓ | ✗ | ✓ |
| Benet et al. (2017) [12] | ✗ | ✓ | ✗ | ✓ |
| Li et al. (2020) [13] | ✗ | ✓ | ✗ | ✓ |
| Filecoin [1] | ✗ | ✓ | ✗ | ✓ |
| *Multiple Auditors* | | | | |
| Benson et al. (2011) [9] | ✓ | ✗ | ✗ | ✓ |
| Gondree et al. (2013) [10] | ✓ | ✗ | ✗ | ✓ |
| *Local Auditor Relying on TEE Clocks* | | | | |
| Dang et al. (2017) [11]* | ✓ | ✓ | ✓ | ✗ |
| EnclavePoSt (2022) [16] | ✓ | ✓ | ✓ | ✗ |
| *Multiple Auditors and Reliance on TEE Clocks* | | | | |
| ReliableBox (2021) [17]* | ✗ | ✗ | ✗ | ✗ |

* deprecated since Intel excluded trusted time from SGX Linux PSW [18].

TABLE I: PoTR properties compared with the related work

## I. Introduction

Today, there are many scenarios in which end users, or organizations, store data in machines run by third parties, either to ensure durability and availability, or to ensure that customers can access data with low latency. Relevant examples include cloud storage (e.g., Dropbox, iCloud, and Google Drive), peer-to-peer storage (e.g., Filecoin [1], IPFS [2], and Swarm [3]), content distribution networks (e.g., Akamai [4] and Cloudflare [5]), and, more recently, edge storage [4], [6]. In this emerging edge computing environment, the latency for data access will become a crucial constrain and challenge for any data placement algorithm, due to the highly distributed and zero-trust environment [7].

Despite significant advances in storage systems, trust in providers has remained unchanged [8]. Customers require mechanisms to verify that QoS (Quality of Service) is being respected. Relevant QoS aspects include the guarantee that the third party will not discard or corrupt the stored data, that the data is stored on multiple distinct machines, in specific geographic locations, and that users are served with some bounded delay. Unfortunately, a misbehaving provider may opt to avoid complying with the agreement if it can gain some benefits and pass unnoticed. For example, the provider may keep the data in fewer locations than agreed with the customer, assuming that it may be impossible for the customer to audit how many replicas are used or where these replicas are placed. This threat has motivated the development of auditing techniques that are capable of extracting storage proofs, that is, evidence that the third party is complying with (or violating) the defined quality of service [9]–[13].

In this paper, we present a mechanism that aims at assessing whether a given server node is able to retrieve data objects with a latency lower than some SLA-specific threshold $\delta$. By estimating an upper bound on the data access latency, we can also determine if the data is placed where expected. Namely, when the SLA threshold $\delta$ is small, it is possible to verify if data is being stored at the audited node or elsewhere. Our new proof mechanism, named *Proof of Timely-Retrievability* (PoTR), takes advantage of the existence of Trusted Execution Environments (TEEs) [19]–[21] (concretely, Intel SGX enclaves [22]) to ensure that the challenge is executed by the node being audited [20], not by some other remote node. By using TEEs, we can also avoid prematurely revealing the data to be accessed during an audit, while keeping the communication between the auditor and the audited node to a single request-reply exchange. PoTR has been carefully designed to mitigate the noise introduced by this single message

This work was supported by the Fundação para a Ciência e a Tecnologia (FCT) scholarship 2020.05270.BD, by national funds through FCT via the INESC-ID grant UIDB/50021/2020 and via the SmartRetail project (ref. C6632206063-00466847) financed by IAPMEI, and by the European Union ACES project, 101093126.

2473-3105/23/$31.00 ©2023 IEEE
DOI 10.1109/PRDC59308.2023.00023


(a) Honest Edge Provider. (b) Dishonest Edge Provider.

Fig. 1: PoTR auditing scenarios.

exchange. Our approach minimizes the network impact on our challenge accuracy, eliminating the need to rely on vulnerable and discontinued TEE clocks [18], [23], in contrast with recent related work, listed in Table I.

Compared with previous work [11], [16], [17], we take a step forward and evaluate our proof in the highly challenging edge computing environment. When auditing edge services, the auditor may be located far away from the audited node and the communication network may exhibit large delays and jitter that can affect the accuracy of the proof. Edge computing relies on placing resources physically close to end users, such that applications can offer a latency lower than some SLA-specific threshold $\delta$. By setting the SLA threshold $\delta$ to a small value that can only be satisfied by the provider if data is kept locally, we use PoTR to distinguish the case where the edge node stores data locally (Figure 1a) from the case where it keeps the data in some remote node or in the cloud (Figure 1b). Although we illustrate PoTR in the context of edge computing, our proof is general and can be applied to other contexts.

We have experimentally evaluated the accuracy provided by our proof with nodes placed in two different university campuses and in the cloud. The results show that our proof can be configured to effectively distinguish a fog node that respects the latency SLA from a fog node that does not. We show that, even in the case where the dishonest node stores the file in a nearby fog node (and the observed latency differs by less than $1.5ms$) our PoTR can accurately pinpoint the misbehaviour. Finally, we have experimentally evaluated different alternatives to reduce the overhead induced at fog nodes while executing the proof, by varying the block size and using SGX switchless calls [24].

## II. Background and Related Work

This section provides the essential background for our work.

### A. Distributed Edge Data Storage

Resorting to third-party storage services provides clients with interesting benefits and challenges, especially in the highly demanding edge computing environment.

*1) Edge Storage Benefits:* Many applications strive to reduce their loading times (e.g., social media, entertainment, e-commerce, advertising, and gaming), knowing that even a small difference in milliseconds can significantly impact their revenue [25]. In recent years, a growing number of applications have relied on content delivery networks (CDN) [26], such as Akamai, Amazon, or Verizon, to store static content closer to end users, achieving lower access latencies. In recent years, edge computing has emerged to offer storage and computation even closer to end users, supporting applications for face or object recognition [27], real-time databases [4], and just-in-time video indexing [28], which demand response times below 5-30 milliseconds [29], something that cannot be guaranteed with cloud storage alone or CDNs. The intelligent and resource-aware edge computing environment is composed of a large number of resource-constrained servers known as *fog nodes* or *cloudlets*. Where the self-adapting data placement algorithms will need to consider both the limited resources and the access latency that edge clients suffer, within the distributed and zero-trust environment [7], [30]–[32].

*2) Edge Storage Challenges:* Fog nodes are managed by many local providers, and due to their limited resources, they cannot store copies of all objects stored in the cloud [33]. Instead, they typically keep copies of items that are required by local applications. Moreover, due to their limited storage capacity, edge storage providers may be tempted to oversell their storage and hide this behaviour by fetching, on-demand, data from the cloud or from other servers, instead of serving them from local (edge), resulting in increased latency for data users. One effective approach to address and detect such misbehaviour is through auditing mechanisms. These mechanisms can enhance the intelligent edge environment by providing real measurements to build trust autonomously between the local entities (e.g., through rating schemes) or to penalize non-compliant providers.

Filecoin storage [1] is a concrete example of an auditing mechanism, where a node that fails to prove its ability to store data as required will no longer receive financial rewards and may be expelled from the network. In cloud or edge storage, providers can be compelled to compensate their customers for violations of the defined contract, known as the SLA. Is essecnial to audit that the SLA is being delivered in edge applications such as: 1) Web, EdgeKV [34] (similar to CDNs) charge their clients to store data near clients and reduce latency; 2) Augmented reality apps, they need low-latency access to data to be shown to the user; 3) Autonomous vehicles, they need low-latency access to maps, directions, etc.

### B. Auditing Third-Party Storage Services

When a storage provider is subject to an audit, it must provide a proof that it is applying the storage policies specified in the SLA. This proof is generically called a *proof of storage* [12]. Moreover, since an SLA may cover different aspects of storage implementation, such as the target number of replicas, the location of those replicas, or the latency observed by users when accessing data, it is possible to define different proofs.

*1) Proofs of Storage:* The literature is rich in techniques for obtaining proofs of storage, mostly focused on cloud storage, the most relevant ones are *Proofs of Data Possession* (PDP) [14] and *Proofs of Retrievability* (PoRet) [15] that aim to check if the storage provider keeps at least some copies of the stored data; *Proofs of Replication* [12], [13] that assess if the storage provider has $n$ copies of a data item (they may be placed on the same machine); and *Proofs of Geographic Replication* [9], [10] that test if the provider keeps $n$ data copies on different machines, in distinct geographic locations.

Each of these proofs is issued as a response to a *challenge* [13], that is sent to the storage provider by one or more auditing entities. Typically, a challenge requires the provider to execute a set of reading operations over a subset of the stored data, and return on-time a value that proves the access of the correct data items (e.g. a cryptographic hash of the audited data).

*2) Structure of a Challenge:* A simple way to verify if a storage provider keeps a given data item would be to request that item and then check its integrity. Although this method could, in fact, offer a PoRet, it has several limitations.

First, this approach is inefficient, as it requires the auditor to consume a large amount of bandwidth to obtain the proof. Therefore, to save bandwidth, most challenges require the storage provider to read a subset of the stored data items and compute a cryptographic hash of them. Typically, the response has to be issued within a predefined deadline, determined by the auditor [9], [11]–[13]. Furthermore, when sending the challenge, the data items should be revealed interactively to prevent the storage provider from downloading missing items on demand [13]. If the provider cannot guess in advance the data items to be accessed, it has to keep all items within the constraint access time to build a correct and timely proof.

Second, a single request for data items cannot verify some of the service requirements. For example, it cannot assess whether the storage provider keeps just one or several replicas of the data items. A typical solution is to encode each file with a distinct secret key [12]. Unfortunately, this does not prevent a provider from keeping all replicas in the same machine, which may compromise availability. An option is to design the challenge so that it is impossible to respond on time if all replicas are kept on the same machine, although it can be feasible if the proof is built in parallel [13].

These mechanisms are not enough to guarantee that the proof was generated by the target node. For this purpose, it is possible to leverage a TEE to interactively reveal the challenge and ensure that the operations essential to the proof construction are executed in a given machine [11].

### C. Trusted Execution Environments

A processor with a TEE has two execution modes: *normal mode*, where the operating system and applications run, and *secure mode*, an isolated environment which ensures the integrity and confidentiality of data and code inside [19]–[21], even in the presence of an untrusted operating system.

There is a set of different TEE architectures [21], for example, Intel Software Guard Extensions (SGX), ARM TrustZone, AMD Secure Memory Encryption (SME) and AMD Secure Encrypted Virtualization (SEV). We resort to *Intel SGX* technology, as it is available in Intel systems (PCs and servers), and we use Intel machines in our experimental setup.

*Intel SGX* splits the computing environment in two parts: the secure mode known as *enclaves*, and the normal mode known as the untrusted part. Furthermore, since the processor core can only execute one environment at a time, the exchange of environments occurs through hardware calls, more precisely *ECalls* and *OCalls*. In addition to code and data security, *Intel SGX* provides security guarantees regarding machine identification. In other words, an external machine that communicates with the enclave can get a guarantee that it is communicating with a real enclave, through the *attestation process* [20], [33].

### D. Discussion

We now discuss the limitations of the related work in auditing storage systems, summarized in Table I.

**Efficient Execution.** Both PDP [14] and PoRet [15] are designed to offer efficient cryptographic mechanisms to verify the integrity of cloud-stored data. Benet et al. [12] present a mechanism capable of verifying if there are a certain number of copies of a file, while Li et al. [13] audit if such copies are stored on different physical machines. These proofs depend on a single remote auditor, requiring low cost for deployment and execution. Filecoin [1] offers blockchain-based cooperative digital storage, which requires expensive cryptographic operations. These proofs cannot estimate data location or access latency, making them unsuitable for testing if a provider stores the required data in a specific node.

**Data Locality.** Triangulation mechanisms are one way to estimate data location. Gondree et al. [10] and Benson et al. [9] follow this approach by relying on multiple auditors/landmarks for the estimation. Unfortunately, triangulation can be expensive since it requires the intervention of multiple landmarks for each proof execution, and the accuracy of this mechanism depends on the proximity of the landmarks to the audited node. In this paper, we take a different approach, where we decouple geolocation from data locality. An auditor can first geolocate a node (using techniques as in [10]) and then run our PoTR to check if the files are stored locally at that node. This approach has the advantage that the accuracy of the locality proof no longer depends on the landmarks, and exclusively relies on a single parameter that affects the proof duration, making it practical, particularly in edge environments.

**Delegation Attack.** Despite the use of triangulation, none of these systems is capable of enforcing the proof to be executed on a given machine; this is a critical obstacle when attempting to audit specific nodes, particularly in an edge storage environment. A storage provider, which controls the infrastructure, may capture the challenge request and delegate the production of the response to the remote storage: such behavior may be difficult to detect. Dang et al. [11], and EnclavePoS [16] resort to a TEE to execute their challenge on the correct machine to prevent this attack but, unfortunately, are still vulnerable to a clock delay attack.

**Delay Attack.** Using a TEE to trivially implement the auditor alone is not enough to ensure the generation of correct and honest proofs. These systems assume that the SGX clock can be trusted to measure the duration of the challenge at the audited node [11], [16], [17], but recent research has shown that the enclave cannot read an accurate and reliable system clock, due to the following issues [23]: (1) the storage node system clock is vulnerable to manipulations by the untrusted part; (2) the operation to read trusted timers, such as those provided by *SGX* or, TPM (*Trusted Platform Module*), has

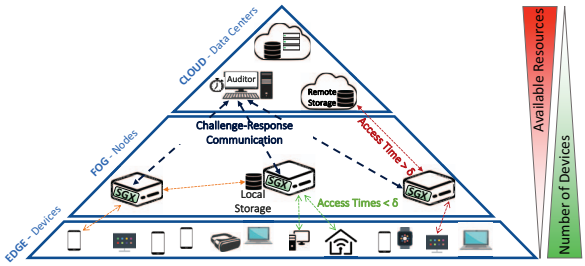



Fig. 2: System architecture.

a large overhead, penalizing the proof accuracy; and (3) the untrusted part can maliciously delay trusted timer messages, to fetch remote data blocks on demand, and thus escape detection. Additionally, since 2020, Intel has excluded SGX trusted time from Linux PSW, leaving Dang et al. [11] and ReliableBox [17] deprecated. As a result, it is not practical to use the enclave for time measurements. Furthermore, as discussed below, ReliableBox focus on the triangulation task and is unable to verify if the audited node keeps the files locally or remotely.

**Geolocation of the Node Being Audited.** Some applications may require the geolocation of the machine being audited. The geolocation problem is orthogonal to the problem of timely retrievability and we advocate that these problems must be addressed by different, complementary, mechanisms. Geolocation typically requires the use of multiple auditors (namely, to perform triangulation) while, as we show in this paper, timely retrievability can be achieved using a single auditor. This separation allows performing the proof of geolocation only sporadically (for instance, when the machine boots) and then run a proof of locality more often.

One way to perform triangulation is by leveraging One-Way Delay (OWD) estimation. ReliableBox [17] measures the duration of the challenge both at a set of remote auditors and inside the enclave. It then computes the time difference between these two measurements to estimate the Round-trip time (RTT) between the auditors and the server, performing geolocation via triangulation. Unfortunately, ReliableBox only captures the network delays and is unable to assess accurately the duration of the proof construction at the audited node. This means that the response to the challenge can take an arbitrarily long duration (e.g. when data is stored in a remote node) without affecting the operation of ReliableBox: triangulation would still be accurate but the result of the challenge cannot guarantee data locality or any other properties. To provide both geolocation and timely retrievability, schemes such as ReliableBox need to be combined with schemes such as the one proposed in this paper.

## III. System Architecture

We now present our system architecture in the context of edge computing that includes an auditor, a set of fog nodes, and a set of remote storage systems connected to the fog nodes (see Figure 2). Our proof, presented in the next section, is computed by a fog node to prove that it can access data with a latency lower than a given agreed threshold $\delta$.

### A. Assumptions

The protocol for obtaining a proof is executed between two nodes: the auditor and the audited fog node. A fog node is said to be *correct* if it can retrieve the files assigned to it with a latency lower than some given threshold $\delta$. Any fog node that cannot satisfy this requirement is denoted *faulty*. The value of $\delta$ is application specific, but can be small and, in some cases, can only be satisfied if data is stored in a storage device directly connected to the fog node. Providers should offer SLAs that define $\delta$ based on their capacity to maintain consistent performance under varying workloads. If a provider cannot meet $\delta$ latency under under high loads, they should consider offering an SLA with higher latency.

Our proof requires the audited node to read a configurable number $N$ of data blocks. $N$ can be conservatively selected to mask worst-case errors that may result from the variability of access to local storage and the variability in round-trip times. We also show that knowledge of the distribution of network delays and the distribution of storage access delays can be used to optimize the value of $N$ when auditing honest nodes. Interestingly, these optimizations cannot be exploited by a rational provider to evade auditing. The configuration of PoTR is discussed in Section V-C.

We assume that each fog node has a processor with *Intel SGX*, as we rely on the guarantees provided by a TEE. We assume that the auditor has the guarantee that it communicates with the expected enclave, due to the attestation process [20], [33], and also that the integrity and confidentiality of the data and code inside the enclave are guaranteed [33]. As explained in Section II-D, the enclave cannot read a reliable system clock [18], [23]. However, the enclave can provide us with the guarantee that the proof is effectively built at the audited node, which is the property we leverage in the solution [20].

Finally, we assume that it is not feasible for the edge storage provider to reallocate enough objects in less than $\delta$ at the beginning of the audit, from a remote storage location. As it will be seen, our audit executes quickly (under 500ms), limiting the number of files that can be downloaded in time, even if the edge node has a high bandwidth link.

### B. Fog Node Storage Organization

The edge storage provider is responsible for storing the files in the fog layer and ensuring that the files are stored in such a way that they can be retrieved with a latency lower than $\delta$, the SLA defined threshold.

In each fog node with *Intel SGX*, local documents are kept in the untrusted part, as the storage capacity of the enclave is limited [33]. Thus, if the processor is running inside the enclave, there must be an exchange between execution environments to read a data item (from the enclave to the untrusted part). Even if the data item is remote, there is also an exchange of execution environments, as the untrusted part is responsible for communicating with remote machines [33].

The set of auditable files is known to both the auditor and audited node and sorted in a deterministic manner. Thus, both parties can use the index of a file in the sorted list as a mutually agreed short unique identifier for that file. We denote this index by *set index*. However, agreeing on the set of files and supporting file modifications is beyond the scope of this work. For simplicity, files assigned to a fog node, including their content, do not change. The modification of files can be trivially supported using versioning.

### C. Enclave Geolocation

The goal of PoTR is to check whether a given node stores data locally. When an enclave is attested, it is possible to uniquely identify the specific enclave, but attestation does not reveal its location. Therefore, PoTR alone cannot ensure that data is stored at a given target location; for that, one also needs to geolocate the node. Several geolocation techniques have been proposed in the literature [35]–[37] and any of them can be combined with PoTR to ensure data locality *and* geolocation. Although geolocation proofs are orthogonal to our work, we briefly sketch two methods to discover the location of the enclave: 1) *Proximity Attestation* – the auditor physically launches the enclave in the local machine at the correct location and exchanges a certificate with that enclave for later authenticating the same specific machine/enclave; 2) *Triangulation* – any technique of triangulation can be applied without requiring heavy cryptographic operations (the goal is to geolocate the machine and not data), so the accuracy is not compromised in any way by the storage proof.

## IV. Proof of Timely-Retrievability

In this work, we introduce the *Proof of Timely-Retrievability* (PoTR) mechanism: a storage proof that aims to assess whether a storage node can access stored data with a latency smaller than some specific threshold $\delta$. The auditor can select the value of the $\delta$ parameter based on the specific requirements of a given application, but it is typically small and, in some cases, can only be satisfied if the audited (edge) node keeps the data locally. With the goal of applying the proof in edge computing scenarios, we assume small values of $\delta$, in the order of the time required to read a data block from the local disk. By supporting such strict values of $\delta$, PoTR is capable of distinguishing the case in which the fog node stores the data locally (Figure 1a) from the case it keeps it in some remote node or cloud (Figure 1b).

*The PoTR is obtained by the storage node in response to an audit*, so our solution requires a machine with auditing capabilities. We do not restrict the placement of these auditing machines, as there may be many geographically distributed fog nodes [38]. Therefore, our proof was designed to be obtainable from an audit machine anywhere on the Internet. This property offers a lot of flexibility regarding the deployment of the auditor and allows a single auditor to perform audits on a large number of fog nodes.

An alternative could be to ask clients to report the latency they observe and use this information to perform the audit. However, this approach raises many privacy challenges, an attacker could infer the client's location given its latency to a fog node. If a PoTR can be extracted independently of the auditor location, we avoid these vulnerabilities.

### A. Challenges

In the design of our PoTR, we face some obstacles, namely: i) the timing information provided by the audited node cannot be trusted [23], thus the time to produce the proof must be measured by the auditor; ii) the network between the auditor and the audited node is subject to variance that introduces errors when estimating the time the audited node took to produce the proof; iii) storage/fog nodes are heterogeneous [38], and the time they require to perform computations and read data (even if the data is local) is not constant, so the proof should be based on average values from multiple readings; and iv) the audited node may attempt to delegate the generation of the proof to another node that has faster access to the data than the audited node itself, so it is required to ensure the proof is produced by the audited node, and not delegated.

### B. Design of the Challenge

The challenge requires the untrusted part of the fog node to access a given number of data objects, in a certain sequence, and return, at the end, a value related to these data objects. *The delay the fog node takes to read these data blocks, and to compute the final value, is used by the auditor to estimate the reading delay observed at the audited node and to check if it matches the target threshold $\delta$.*

Each challenge (implicitly) specifies a sequence of files that must be accessed by the audited node, and each file is uniquely identified by a *set index*. For efficiency reasons, the fog node for each file reads a data block of size $s_b$, instead of the entire file. In practice, the block size should be a multiple of the block size used by the fog node file system.

In each challenge $c$, the fog node has to read a pseudorandom and unpredictable sequence of $N$ data blocks (each of size $s_b$), and return a cryptographic hash of the concatenation of all data blocks accessed. The number $N$ of data blocks is a configuration parameter that influences the accuracy and efficiency of the challenge: the higher the value $N$, the more accurate but less efficient the proof. The approach to configure this parameter is explained in Section IV-D, and if the user is unable to configure $s_b$, it is still possible to execute our challenge defining only $N$, as discussed in Section V-C.

The pseudorandom sequence of the data blocks is determined by a nonce $\eta^c$ (unique per challenge, and generated by the auditor) and the content of the data blocks. For each challenge, the auditor sends the nonce (encrypted with a symmetric key), the number $N$ of data blocks and the size $s_b$ of a block to the enclave. To ensure that the nonce $\eta^c$ is not disclosed to the untrusted part, the nonce never leaves the enclave. The untrusted part only has access to a cryptographic hash of the nonce, we denote this hash as *#index*. With the *#index*, the untrusted part is able to determine the set index of the first file to be accessed. The set index is determined by

applying the modulo function (*mod*) to the hash with the total size of the set of files, i.e., the set index is the remainder of the division of the hash by the number of files.

Since the fog node has to read a data block of size $s_b$, and not the entire file, the auditor sends to the enclave a second nonce $\eta_b^c$ that will determine a data block inside the file, with the computed set index. The enclave, in turn, computes the cryptographic hash of this second nonce and forwards the result to the untrusted part. Both nonces are hashed by the enclave, to ensure that the untrusted part does not know them in plain text. Otherwise, the untrusted part could maliciously predetermine the $N$ data blocks. Now, with the hashed $\eta_b^c$, and the total size of the file to be accessed, the untrusted part can determine the data block, with size $s_b$, inside the file to be retrieved, by applying the modulo function.

When the untrusted part receives the first *#index* from the enclave, it proceeds to determine and read the corresponding initial data block. It then calculates a cryptographic hash of the data block's content, combined with the *#index*, resulting in *result_hash*. Subsequently, the untrusted part returns the value of *result_hash* to the enclave. The enclave, in turn, computes a new *#index* by hashing the response *result_hash* together with the nonce $\eta^c$. This newly generated *#index* is then forwarded to the untrusted part, which uses it to identify the next file.

For each subsequent hash, the untrusted part repeats the execution of the modulo function to obtain the next set index and the index of the data block within the identified file. It reads the corresponding data block and calculates its cryptographic hash together with the file *#index*, yielding *result_hash* that is returned to the enclave. The enclave computes two new hashes (block and file index), and this process continues until all $N$ data blocks are read. Note that each new *#index* is computed using the hash of the previously retrieved data blocks. Finally, the enclave computes a final hash using $\eta^c$ and sends the result back to the auditor as the conclusive proof. The auditor, upon receiving the final hash, repeats all the aforementioned computations to construct a verifiable version and verify the correctness of the proof. If both computations match, the issued proof is deemed correct. Figure 10 in Appendix A provides an overview of the sequential steps involved in the execution of our challenge.

*Security Guarantees:* Each challenge requires two unique nonces that will determine in a sequential and pseudo-randomly faction which $N$ data blocks the untrusted part must read. This interaction with the enclave ensures that the fog node is unable to retrieve the $N$ data blocks in parallel from neighboring nodes. Instead, it must retrieve one block at a time to determine the next block it needs. Additionally, the dependency between data blocks, since the next block index depends on the content of the previous one, ensures that the fog node cannot reuse outputs from previous challenges, thus maintaining challenge freshness. The constant exchange between execution environments (enclave to untrusted part and vice versa) guarantees that the fog node cannot rely on a remote machine for the entire proof computation. The proof is computed solely on the expected machine [20]. It is important to note that through the verification of proof correctness, the auditor can assess the integrity of the stored documents. Any modification to a file will render the proof invalid.

### C. Reading Delay $\delta$ at the Audited Node

In addition to checking *proof correctness*, the auditor has to check the *proof timeliness*. When the challenge is sent to the enclave of the audited node, the auditor starts a timer that is stopped when the proof is received. A proof is valid if it is correct and the reading delay estimation, for a single data block, is acceptable to the auditor.

Having $T_i$ as the time elapsed between the auditor sending the challenge $i$ and receiving the response from the fog node, $T_i$ can be decomposed into different factors, namely:

$$T_i = rtt_i + \alpha_i^1 + \ldots + \alpha_i^N + \delta_i^1 + \ldots + \delta_i^N \tag{1}$$

where $rtt_i$ is the network round trip time for challenge-response messages, $N$ the number of data blocks to be accessed, $\alpha_i^j$ the delay observed at the fog node to compute the cryptographic hash of a given data block $j$ and compute the index of the next block, and $\delta_i^j$ the delay observed to read a data block $j$. Note that for sufficiently large values of $N$, we will have the following:

$$\frac{\alpha_i^1 + \ldots + \alpha_i^N}{N} = \overline{\alpha}_i \approx \overline{\alpha} \qquad \frac{\delta_i^1 + \ldots + \delta_i^N}{N} = \overline{\delta}_i \approx \overline{\delta} \tag{2}$$

However, the auditor is unable to measure accurate values for $rtt_i$, $\overline{\alpha}_i$ and $\overline{\delta}_i$, as they are different and variable in each challenge $i$ and the auditor is only able to measure the total delay $T_i$. Therefore, to estimate $\overline{\delta}$, i.e., the actual mean delay a fog node takes to read a data object, in our work we resort to mean values:

$$T_i = \overline{rtt} + N\overline{\alpha} + N\overline{\delta} \tag{3}$$

$$\overline{\delta}_{\text{estimate}} = \frac{T_i - \overline{rtt} - N\overline{\alpha}}{N} \tag{4}$$

Therefore, the estimate error for a given challenge $i$ will depend on how far the observed values are from the mean values. The $\overline{rtt}$ is calculated independently of our challenge, being obtained by measuring several network samples and dividing them by the number of samples. Experimentally, we verified that the $\alpha$ values are subject to a negligibly small variance, whereby we assumed $\overline{\alpha}_i = \overline{\alpha}$, i.e., we ignore the error introduced by $\alpha$ sampling. Therefore, the error of the estimate $\overline{\delta}_{\text{estimate}}$ depends on two factors: i) the reading error $\varepsilon^{\delta}$ when estimating $\overline{\delta}$ (which depends on the sample size, i.e., the number of data blocks accessed) and ii) the variance between the observed network round-trip time ($rtt_i$) and the expected mean value ($\overline{rtt}$), divided by the number $N$ of data blocks, as Equation 5 describes:

$$\varepsilon^{\text{rtt}} = \frac{|rtt_i - \overline{rtt}|}{N} = \frac{\Delta^{rtt}}{N} \tag{5}$$

### D. Configuring the Challenge

The challenge can be configured by providing two parameters: the maximum error $\varepsilon^{\overline{\delta}}_{\text{max}}$ that he is willing to tolerate, when estimating $\overline{\delta}$, and the reliability of the challenge $\phi$, a parameter specified by the auditor that captures the probability of estimating $\overline{\delta}$ with an error lower than $\varepsilon^{\overline{\delta}}_{\text{max}}$ (in practice we use the 99.99 percentile). As noted before, the error of the estimate $\varepsilon^{\overline{\delta}}_{\text{max}}$ results from the variance of local reads experienced by the audited node ($\varepsilon^{\delta_i}_{\text{max}}$) and by the variance of the network delays ($\varepsilon^{\text{rtt}}_{\text{max}}$), i.e., $\varepsilon^{\overline{\delta}}_{\text{max}} = \varepsilon^{\delta_i}_{\text{max}} + \varepsilon^{\text{rtt}}_{\text{max}}$. For both sources, the error diminishes with the number of samples $N$, thus, $N$ should be chosen so that $\varepsilon^{\text{rtt}}_{\text{max}} + \varepsilon^{\delta_i}_{\text{max}} < \varepsilon^{\overline{\delta}}_{\text{max}}$.

Let $\varepsilon^{\delta^j_i}$ be the difference between the average access latency $\overline{\delta}$ and the latency observed when reading block $j$ during the execution of challenge $i$. The error caused by the variance of storage access in a given challenge $i$ is given by:

$$\varepsilon^{\delta_i} = \frac{\sum_{j=1}^{N} \varepsilon^{\delta^j_i}}{N}$$

Given $\phi$, the expected value of $\varepsilon^{\overline{\delta}}_{\text{max}}$ can be derived from the distribution of the access delays. Similarly, with the inverse cumulative distribution function (ICDF) of the network distribution, we can also calculate the expected maximum difference between the observed value $rtt_i$ and the expected mean $\overline{rtt}$ that matches $\phi$.

### E. Network Delay Distribution

Our proof is capable of finding the correct $\overline{\delta}_{\text{estimate}}$ even if delay distributions are unknown, by choosing worst-case default values that can be used conservatively (see Section V-C). However, it is possible to leverage information regarding the network delay distribution to select a more efficient $N$ value such that the sum of both errors is kept below a target value of $\varepsilon^{\overline{\delta}}_{\text{max}}$. We show this in the concrete edge node scenario in Section V-D2. This optimization, based on measuring network behavior, is safe and cannot be exploited by a malicious provider. The provider can introduce artificial delays when replying to the auditor, but this will not reduce the variance that is introduced by the network itself; on the contrary, it may only increase the variance. In turn, this may force the auditor to use larger values of $N$ than strictly required, increasing the accuracy of the proof at an extra cost to the malicious provider. Therefore, it is always beneficial for the adversary to be honest when the network behavior is measured.

## V. Evaluation

We now present our proof evaluation. We begin by describing our experimental setup and the different benchmark scenarios. Then we show that our proof works even without knowledge of network distribution. Afterward, we discuss how the challenge should be configured, for the block size, and $N$ samples. After finding the correct configuration, we evaluate the performance of the challenge in the different scenarios and show experimentally that our proof can accurately distinguish a misbehaving server from a correct server, in the context of edge computing. Finally, we evaluated different alternatives to reduce the impact of our proof on the audited node.

### A. Experimental Setup

We resorted to two different types of machines: 1) Intel NUCs to represent fog nodes, and 2) virtual machines in the Windows Azure cloud to represent remote entities. For NUCs, we have resorted to the Intel NUC10i7FNB, it has an Intel i7-10710U CPU that supports Intel SGX, 16GB RAM, 240GB SSD M.2, and Ubuntu 20.04 LTS. We run the Intel SGX SDK Linux 2.13 Release and OpenSSL 1.1.1k. An Intel NUC is an example of what a fog node might be, as it possesses modest computational resources but is relatively inexpensive for large-scale deployments. For the cloud deployment in Azure, we resort to Standard D2ads v5 with two vcpus, and 8 GB RAM in west Europe (Netherlands) the closest data center to our laboratory; our laboratory is located in a European capital. We leverage these two types of deployment to create different scenarios and evaluate our storage proof; we explain our scenarios in more detail in the next section.

For the stored data that our proof is responsible for auditing, we generated 150GB with random data divided into files of 1GB, our proof leverages a hash function to choose the next block to be accessed; this offers uniform distribution access over the 150GB of data. In our code implementation, we used the SHA-256 function as a cryptographic hash function and AES in GCM mode with 128 bit keys to cipher the nonces, both currently considered secure [39].

### B. Evaluation Scenarios

In our experiments, we leverage three different entities to evaluate our PoTR storage proof: an auditor; the audited node, which represents a fog node that stores data; and the remaining one is the remote storage node, which is used by the audited node to access data in configurations where it does not store the files locally. In our experiments, the fog node stores all files at the same location, i.e., all files are stored locally, or all files are stored in the remote node.

We deployed four different scenarios to evaluate PoTR; the difference between the scenarios is the location of the remote storage node, since the closer the remote storage is to the audited node, the more difficult it is to detect malicious behavior. The first scenario is represented in Figure 1a: the fog node has honest behavior and stores all data locally. Figure 1b presents the malicious behavior that the edge provider can adopt by resorting to remote storage. Figure 1b represents the three remaining scenarios in which we vary the location of the remote storage from: a) a virtual machine on the Azure cloud; b) a NUC on a different campus of our university; c) a NUC located in our laboratory and connected through a switch to the audited node, with a mean network delay of $0.1ms$. Regardless of the scenario, both the auditor and the fog node/audited node do not change their location, the auditor is running in the Azure cloud, and the audited node in an NUC at our laboratory. By deploying different machines at different

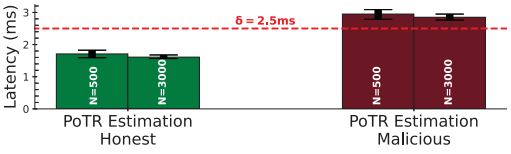



Fig. 3: Overview of PoTR $\delta$ estimative without network knowledge, picking a large N value of 3000, 64KB block size.

geographic locations, we capture the different variations and delays of wide-area links.

### C. PoTR Without Network Knowledge

As stated in Section III-A, the efficiency of PoTR can be improved given knowledge on the distribution of network delays and access delays. When executing PoTR without this knowledge, a large default value for $N$ should be used, enough to compensate worst-case network variance; we show that a default value $N = 3000$ can be safely used in most scenarios. We have selected this default value based on latency measurements from our laboratory to the farthest Azure datacenter[1], the Australia Central 2. We have observed an average RTT of $\approx 286ms$ and a maximum latency of $\approx 1143ms$. When considering a network variance with this order of magnitude in Equation 5, by setting $N = 3000$ we can reduce $\varepsilon^{\text{rtt}}$ to $\approx 0.28ms$. In this scenario, the variance in storage access delay is negligible compared to the network variance.

Figure 3 shows the results of running PoTR using the default value of $N = 3000$ in different scenarios. We can observe that it is possible to set the SLA threshold to a small value such as $\delta = 2.5ms$, to ensure that the audited node has the data stored locally (the local delay to read a data block is $\approx 1.7ms$), and that our $\overline{\delta}_{\text{estimate}}$ achieves high accuracy in both scenarios.

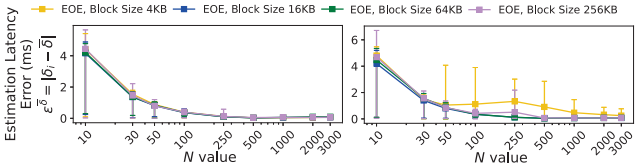
However, with $N = 3000$ our challenge takes approximately $5.2s$ to execute. When the challenge is executed using more stable networks, it is possible to fine-tune the value of $N$ to achieve similar accuracy with a lower cost for the audited node. In the next section, we demonstrate results for various values of $N$. For instance, with $N = 500$, we achieve accurate results with a challenge executed in just $0.9s$.

### D. PoTR Configuration

We now derive, based on experimental results, the best configuration for our storage proof, by setting the block size and the $N$ value for PoTR. Note that in a situation where the user is unable to configure the block size, they can simply use a high value for $N$, as demonstrated in the previous Section V-C.

*1) Selecting the appropriate Block Size:* To find the appropriate value for the size of the block to use in our challenge, we evaluated the performance of the challenge with different block sizes and different $N$ values. The results are presented in Figure 4. We executed our challenge to estimate the $\delta$ value and calculated the error relative to the real reading delay in the fog node, we present the results for the honest fog node scenario in Figure 4(a) and for the nearby fog storage scenario

[1]We leveraged this website: https://cloudpingtest.com/azure




(a) Honest Fog Node. (b) Nearby Fog Storage.

Fig. 4: PoTR experimental error for different scenarios and block size, EOE is Experimentally Observed Error.

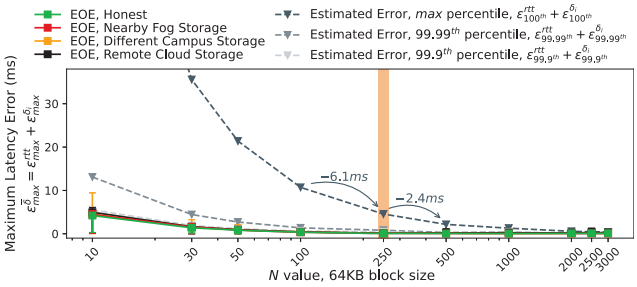



Fig. 5: Theoretical and experimental error for the PoTR challenge (EOE means Experimentally Observed Error).

in Figure 4(b). We observe that for small values of $N$ the results obtained exhibit a large variance and only by increasing $N$ do we obtain more accurate results. In both scenarios, we observe that the size of the block has a relatively smaller impact on the accuracy of the result than the value of $N$. Still, it is possible observe that, when using larger block sizes one can obtain more accurate results. This is clearly noticeable in the scenario where files are stored in a nearby fog node, where the configuration using blocks of 4KB takes longer to converge than configurations with larger block sizes. Furthermore, in Figure 4(b), it is possible to observe that when the $N$ value goes above 100, blocks of 64KB offers a slightly smaller error than blocks of 16KB. For these reasons, we opted to set our challenge with blocks of 64KB.

*2) Finding the appropriate value for $N$:* As described in Section IV-D, the value $N$ can be determined by providing $\varepsilon^{\overline{\delta}}_{\text{max}}$, which depends both on the network error and the reading error, this network knowledge allows us to find an optimal value for $N$. Figure 5 presents the estimated error for our challenge under different percentile values, of the reliability $\phi$, for each error $\varepsilon^{\text{rtt}}$ and $\varepsilon^{\delta_i}$ (these two were measured experimentally). Note that $\varepsilon^{\overline{\delta}}_{\text{max}}$ decreases as we increase the value $N$, this results from the impact that $N$ has on $\varepsilon^{\text{rtt}}$ reducing its effect on the challenge, on the other hand, $\varepsilon^{\delta_i}$ has a negligible impact on $\varepsilon^{\overline{\delta}}_{\text{max}}$, $\varepsilon^{\delta_i}$ is mostly below $1ms$ reaching at most $5ms$. Figure 5 also presents the measured error of our challenge in real experiments in the four different scenarios presented earlier. We observe that the error increases when we also increase the distance between the data and the audited node.

Both $\phi$ values of $100^{th}$ and $99.99^{th}$ percentiles are able to predominantly capture the real observed error of our challenge.

We use the estimated error to choose a desirable value for $N$; in this case, we chose $N = 250$ since this is where we observe the last most significant reduction in the estimated error of $-6.1ms$, versus the increase in the value of $N$, of 150. As expected, increasing the value of $N$ also improves the accuracy of our challenge, for $N = 10$ we obtain an error of $\approx 5ms$, which can be undesirable for edge environments, while for $N = 250$ the observed error is only $\approx 0.1ms$, such a small error can provide a $\overline{\delta}_{\text{estimate}}$ with high reliability.

### E. PoTR Accuracy

Using the configuration parameters derived from the previous analysis, i.e., using a block size of 64KB and by setting the number of samples $N$ to 250, we have executed our challenge in multiple scenarios and captured the distribution of the data access latency estimated by our challenge. We compare the estimated values with the real values, observed at the audited node itself. The results are depicted in Figure 6. Each point in the figure represents either the real latency to access a data block or the $\overline{\delta}_{\text{estimate}}$ to access a block resulting from the execution of our challenge.

It is possible to observe that our challenge estimate closely approximates the real value with a small error, as discussed in Section V-D. The error between the real value and $\overline{\delta}_{\text{estimate}}$ increases as the audited data move further from the audited node. This happens because the farther away the audited node is, the more likely the network delay exhibits a larger variance. Nonetheless, our challenge can still distinguish whether the data is local to the fog node or at a remote site. The accuracy of the PoTR is sufficient to differentiate between scenarios where files are stored locally or at a nearby fog node. Even with a latency difference of less than $1.5ms$, our PoTR accurately identifies the configuration used by the audited node.

Looking at the results in Figure 6, our PoTR enables the establishment of a threshold to accurately differentiate between a correct node (storing all files locally) and a faulty node that stores files elsewhere (even if they are kept in nearby nodes). This threshold is determined by setting a very strict value for $\delta$ in the SLA, such as the $100^{th}$ percentile for reading delays measured locally at the fog node. The $100^{th}$ percentile line, located at $2.2ms$, corresponds to 0.005% false positives and 0% false negatives (we define as positive the detection of malicious behavior of the audited node).

### F. Overhead Imposed by a PoTR Challenge

We now assess the overhead imposed on the audited node while it responds to a challenge. For this, we run a local client that performs read/write access to the data stored on the edge node, and then we measure the loss in throughput of that client during challenge execution. The results are shown in Figure 7.

The throughput of the concurrent client decreases by approximately $57\%$ during the processing of the challenge by the audited node. This is expected as our resource-constrained edge nodes need to access multiple files and compute their digest to respond to the challenge. Despite this non-negligible overhead, we argue that its impact on the overall operation of an edge node is not significant in practice. The challenge typically takes less than $500ms$ to complete, and auditing occurs sporadically in most scenarios (e.g., once per day). Nonetheless, in the next section, we discuss and evaluate strategies to further reduce this overhead.

### G. Strategies to Reduce PoTR Overhead

We considered three complementary avenues to attempt to reduce the overhead of running a challenge in an audited node. The first was to1modify the implementation, taking advantage of switchless calls [24] that reduce the ECall/OCall overhead. The second was to use blocks size smaller than 64K. As discussed earlier, smaller sizes can reduce the accuracy of the test but can have an impact on the overhead. Finally, one can simply reduce the number of samples $N$. By reducing the number of samples, one would reduce the challenge duration and therefore, the period during which clients would be affected by the concurrent execution of a challenge. However, as we show in this section, using switchless calls or reducing the size of the blocks has a minor impact on the overhead.

Figure 8 shows how the use of switchless calls, different block sizes, and different values of $N$, affect the drop in throughput observed by clients during the execution of the challenge and also the duration of the challenge.

It is clear from Figure 8a that the drop in throughput during the execution of the challenge is roughly the same regardless of the block size used or the use of switchless calls. Switchless calls offered a small performance improvement, just a latency reduction of $\approx 1.6\%$ in the best case. When considering block size, the largest difference is observable from 256KB block size, with a throughput reduction of $-55\%$, versus 4KB block size alternative, with a reduction of $-38\%$, relative to the parallel client. This is not a large improvement considering that the 256KB block size is 64 times the 4KB block size. This is because, during the execution of our challenge, our disk access is continuous, independently of the block size, and therefore the impact on other clients will be similar; however, our challenge duration may vary. Additionally, as discussed in Section V-D1, having a very small block size can still affect our storage proof error.

These results suggest that the only effective way to reduce the overhead of the challenge is to reduce its duration. Figure 8b shows that, although block size has some impact on the duration of challenge executions, its impact is relatively minor compared to the effect of reducing the number of samples $N$. This indicates that the most effective strategy to reduce the overhead of the challenge is to reduce the number of samples, as this reduces the duration of the period in which clients are affected. Of course, as discussed before, reducing the value of $N$ will degrade the accuracy of the test, increasing the chances of producing false positives and false negatives.

Figure 9 shows the effect of $N$ on both false positive and false negative rates when using the threshold $\delta = 2.2ms$, as discussed in Section V-E. In this analysis, we consider the scenario where a faulty node stores data at the nearby fog storage, as this is the scenario where it is most challenging

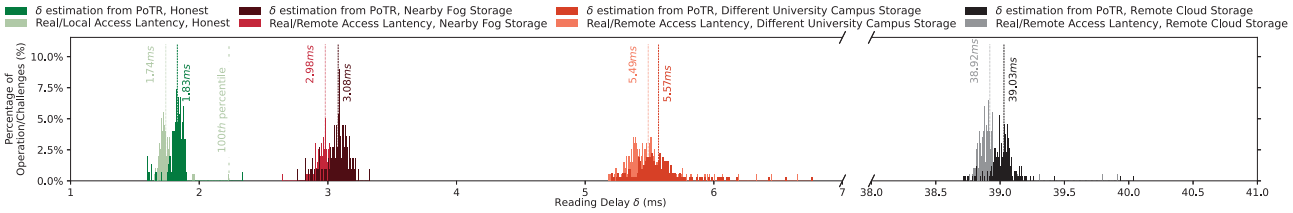



Fig. 6: Distribution and correspondent averages for reading delay $\delta$, in milliseconds, for different scenarios where PoTR is configured with $N = 250$ and 64KB block size.

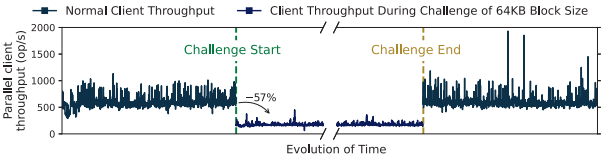
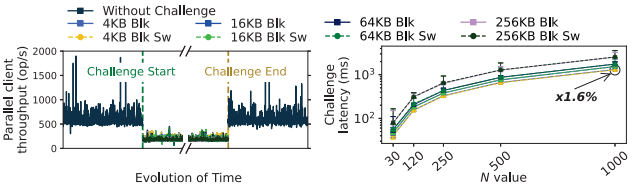



Fig. 7: Challenge impact on concurrent reads.




(a) Throughput Impact. (b) Challenge Duration.

Fig. 8: PoTR duration and throughput impact on concurrent reads, Blk is block, and Sw is switchless.

to accurately detect a misbehavior, given the small difference between the latency offered by correct and faulty nodes. It is interesting to observe that our PoTR offers a false negative rate close to $0\%$ for most values of $N$. Therefore, even where the values of $N$ are smaller than 250 (the required to obtain the best accuracy), our PoTR will detect a misbehaving client. Unfortunately, the true positive rate can increase significantly when using low values of $N$, risking identifying a correct node as faulty. More precisely, the false positive rate increases slowly as we decrease the number of samples from 250 to 100 (at this point, our PoTR offers a false positive rate of approximately $1\%$), but then increases sharply. For values of $N$ smaller than 100 the challenge no longer provides acceptable values for the false positive rate.

In summary, by tolerating a small fraction of false positives, the auditor can reduce the challenge overhead by decreasing the number of samples from 250 to 100. This roughly halves the period during which clients are affected by our challenge.

### H. Combining Multiple PoTR Configurations

As seen in the previous section, using a reduced number of samples, 100 instead of 250, our challenge imposes significantly less overhead while still obtaining a negligible false negative rate and a very small positive rate ($\approx 1\%$). Faced with these observations, we conclude that in some deployments the auditor should combine different configurations of the PoTR:

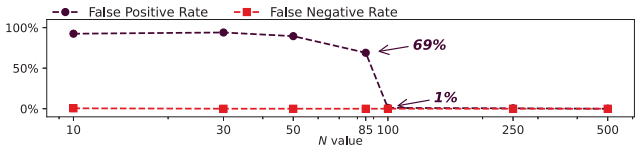



Fig. 9: False positive and negative rates of PoTR under different N values for honest and nearby fog node scenarios.

one that is cheaper and works $99\%$ of the time; another that is more expensive but provides extreme accurate results.

For example, the auditor might use $N = 100$ to challenge the audited node. If the node was flagged as malicious (a positive), the auditor could then launch a second challenge with $N = 250$, to filter out false positives. Note that in such a case, the combined challenge would read on average 102.5 $(0.01 \times 100 + 0.01 \times 250 + 0.99 \times 100)$ blocks instead of the 250 in the normal auditor, a reduction of $41\%$ on average for the number of blocks required to execute our challenge. Notice that, as hinted before, for $N = 85$ this strategy would no longer be worth it since it would require reading 257.5 $(0.69 \times 85 + 0.69 \times 250 + 0.31 \times 85)$ blocks on average, which is larger than the 250 required by the normal auditor solution.

## VI. Conclusions

We present a novel auditing mechanism that is capable of extracting a proof of timely-retrievability, that is, a proof that a given storage node is able to serve requests without violating some given data access latency constraint $\delta$. The proof is designed in a way that, if the storage node does not store locally a significant fraction of the objects, it will be unable to respond in time. We enforce our proof to be executed in the audited node by leveraging SGX enclaves, for iteratively revealing the next data block to be read to the untrusted part.

Our evaluation shows that our proof can accurately detect a node that cannot satisfy the target latency constraint $\delta$ under different edge computing scenarios, resulting in the detection of a misbehaving storage provider.

Next, we aim to extend our evaluation to demonstrate the behavior of our proof with a more sophisticated attacker who varies the percentage of data stored remotely (i.e. some of the files are stored locally, and some are stored remotely), and introduce adjustments to our proof if necessary.

## References


[1] Filecoin, "A blockchain based storage network," https://spec.filecoin.io/.

[2] J. Benet, "IPFS: content addressed, versioned, P2P file system," *arXiv preprint arXiv:1407.3561*, 2014.

[3] Swarm, "Distributed storage platform," https://github.com/ethersphere/.

[4] Akamai, "Real time key value store for edge computing." https://www.akamai.com/products/edgekv.

[5] Cloudflare, "CDN global network," https://www.cloudflare.com/cdn/.

[6] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, 2017.

[7] Autopoietic Cognitive Edge-cloud Services (ACES), "Cloud-edge service for data management." https://www.aces-edge.eu/.

[8] H. Gunawi, M. Hao, R. Suminto, A. Laksono, A. Satria, J. Adityatama, and K. Eliazar, "Why does the cloud stop computing? lessons from hundreds of service outages," in *ACM Symposium on Cloud Computing*, Santa Clara, California, 2016.

[9] K. Benson, R. Dowsley, and H. Shacham, "Do you know where your cloud files are?" in *ACM Workshop on Cloud Computing Security Workshop*, Chicago, IL, USA, 2011.

[10] M. Gondree and Z. Peterson, "Geolocation of data in the cloud," in *ACM Conference on Data and Application Security and Privacy*, San Antonio, TX, USA, 2013.

[11] H. Dang, E. Purwanto, and C. Chang, "Proofs of data residency: Checking whether your cloud files have been relocated," in *ACM on Asia Conference on Computer and Communications Security*, Abu Dhabi, United Arab Emirates, 2017.

[12] J. Benet, D. Dalrymple, and N. Greco, "Proof of replication," *Protocol Labs, July*, vol. 27, 2017.

[13] L. Li and L. Lazos, "Proofs of physical reliability for cloud storage systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, 2020.

[14] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *ACM Conference on Computer and Communications Security*, Alexandria, Virginia, USA, 2007.

[15] F. Armknecht, L. Barman, J. Bohli, and G. Karame, "Mirror: Enabling proofs of data replication and retrievability in the cloud," in *USENIX Security Symposium*, Vancouver, Canada, 2016.

[16] Y. Zhang, W. You, S. Jia, L. Liu, Z. Li, and W. Qian, "EnclavePoSt: A practical proof of storage-time in cloud via intel SGX," *Security and Communication Networks*, 2022.

[17] T. Jiang, W. Meng, X. Yuan, L. Wang, J. Ge, and J. Ma, "Reliablebox: Secure and verifiable cloud storage with location-aware backup," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 12, 2021.

[18] F. Alder, G. Scopelliti, J. Van Bulck, and J. T. Mühlberg, "About time: On the challenges of temporal guarantees in untrusted environments," in *6th Workshop on System Software for Trusted Execution (SysTEX 2023)*, 2023.

[19] J. Ekberg, K. Kostiainen, and N. Asokan, "The untapped potential of trusted execution environments on mobile devices," *IEEE Security Privacy*, vol. 12, no. 4, 2014.

[20] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun, "ROTE: Rollback protection for trusted execution," in *USENIX Security Symposium*, Vancouver, Canada, 2017.

[21] Z. Ning, J. Liao, F. Zhang, and W. Shi, "Preliminary study of trusted execution environments on heterogeneous edge platforms," in *IEEE/ACM Symposium on Edge Computing*, Bellevue, WA, USA, 2018.

[22] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *International Workshop on Hardware and Architectural Support for Security and Privacy*, Tel-Aviv, Israel, 2013.

[23] F. Anwar, L. Garcia, X. Han, and M. Srivastava, "Securing time in untrusted operating systems with TimeSeal," in *IEEE Real-Time Systems Symposium*, York, England, 2019.

[24] H. Tian, Q. Zhang, S. Yan, A. Rudnitsky, L. Shacham, R. Yariv, and N. Milshten, "Switchless calls made practical in Intel SGX," in *Workshop on System Software for Trusted Execution*, Toronto, Canada, 2018.

[25] Akamai, "Online retail performance report: Milliseconds are critical," https://www.akamai.com/newsroom/press-release/akamai-releases-spring-2017-state-of-online-retail-performance-report.

[26] Grand View Research, "CDN market report," https://www.grandviewresearch.com/industry-analysis/content-delivery-networks-cnd-market.

[27] C. Streiffer, A. Srivastava, V. Orlikowski, Y. Velasco, V. Martin, N. Raval, A. Machanavajjhala, and L. Cox, "ePrivateEye: To the edge and beyond!" in *ACM/IEEE Symposium on Edge Computing*, San Jose, CA, USA, 2017.

[28] M. Satyanarayanan, P. Gibbons, L. Mummert, P. Pillai, P. Simoens, and R. Sukthankar, "Cloudlet-based just-in-time indexing of IoT video," in *Global Internet of Things Summit*, Geneva, Switzerland, 2017.

[29] G. Ricart, "A city edge cloud with its economic and technical considerations," in *IEEE International Conference on Pervasive Computing and Communications Workshops*, Kona, HI, USA, 2017.

[30] I. Lujic, V. De Maio, S. Venugopal, and I. Brandic, "Sea-leap: Self-adaptive and locality-aware edge analytics placement," *IEEE Transactions on Services Computing*, vol. 15, no. 2, pp. 602–613, 2022.

[31] Z.-L. Shao, C. Huang, and H. Li, "Replica selection and placement techniques on the iot and edge computing: a deep study," *Wireless Networks*, vol. 27, no. 7, pp. 5039–5055, 2021.

[32] L. Epifâneo, C. Correia, and L. Rodrigues, "Cathode: A consistency-aware data placement algorithm for the edge," in *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*, 2021, pp. 1–10.

[33] C. Correia, M. Correia, and L. Rodrigues, "Omega: a secure event ordering service for the edge," in *IEEE/IFIP International Conference on Dependable Systems and Networks*, València, Spain, 2020.

[34] Akamai, "Supercharge your edgeworkers apps with a serverless key-value store." https://www.akamai.com/products/edgekv.

[35] J.-H. Choi and C. Yoo, "One-way delay estimation and its application," *Computer Communications*, vol. 28, no. 7, pp. 819–828, 2005.

[36] A. Vakili and J.-C. Gregoire, "Accurate one-way delay estimation: Limitations and improvements," *ieee transactions on instrumentation and measurement*, vol. 61, no. 9, pp. 2428–2435, 2012.

[37] A. Abdou, A. Matrawy, and P. C. van Oorschot, "Accurate one-way delay estimation with reduced client trustworthiness," *IEEE Communications Letters*, vol. 19, no. 5, pp. 735–738, 2015.

[38] M. Mukherjee, R. Matam, L. Shu, L. Maglaras, M. Ferrag, N. Choudhury, and V. Kumar, "Security and privacy in fog computing: Challenges," *IEEE Access*, vol. 5, 2017.

[39] E. Barker and A. Roginsky, "Transitioning the use of cryptographic algorithms and key lengths," National Institute of Standards and Technology, NIST Special Publication 800-131A Revision 2, Mar. 2019.


## Appendix A
## PoTR communication protocol

In Figure 10, we present in detail all the steps of our communication protocol in the execution of our PoTR challenge. We separate the steps on the auditor's side between the untrusted part and the enclave, where the sensitive parts of our challenge, such as the nonce, are always securely stored inside the enclave.

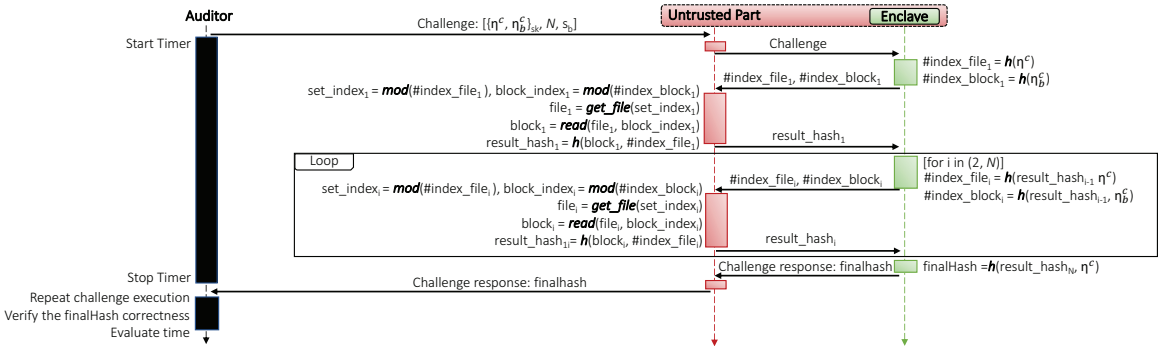



Fig. 10: PoTR challenge execution. $set\_index_i$ is the index of a file, and $block\_index_i$ is the index of a data block with size $s_b$ inside $file_i$. $block_i$ is the read data block. $sk$ is a secret key between the auditor and the enclave. $\#index$ is a hash.