TÉCNICO LISBOA

Proof of Timely-Retrievability for Storage Systems at the Edge

Rita Alexandra Rodrigues Prates

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Luís Eduardo Teixeira Rodrigues
Prof. Miguel Nuno Dias Alves Pupo Correia

Examination Committee

Chairperson: Prof. Pedro Tiago Gonçalves Monteiro
Supervisor: Prof. Luís Eduardo Teixeira Rodrigues
Member of the Committee: Prof. Filipe João Boavida Mendonça Machado Araújo

November 2021

Acknowledgments

First, I would like to thank my dissertation supervisors Prof. Luís Rodrigues and Prof. Miguel Correia for the opportunity to work on this thesis under their supervision. Thank you for all the meetings, all discussions, and all the provided feedback during the execution of this work. This thesis was only possible, due to their insight, support, and share of knowledge.

I also would like to thank Cláudio Correia for his help, and discussions throughout this thesis, and Prof. Rui Henriques for the insight in statistics.

I would like to thank my family, specially my parents for all the support, encouragement and caring over all these years. Thank you for allowing me to choose my path, without stopping to supporting me.

Last but not least, I would like to thank to all my friends and colleagues, for helping me to grow personally and professionally. I would like to leave a special thanks to *Lo Quarteto* and *Pseudo-mestres* for always having my back, and supporting me when rough times were around. This work would be impossible without your continuous support.

To each and every one of you – Thank you.

Abstract

Edge computing is a model that places servers close to the edge of the network, in order to assist applications that run in resource-constrained devices. Edge servers may be used to store files such that clients can access data with low-latency.Yet, as the capacity of edge nodes is limited, providers of edge storage may be tempted to oversell their capacity and to hide this behavior by fetching, on-demand, data from the cloud instead of serving it with the required low latency.

In this thesis, we study techniques that have been proposed in the literature that can help in detecting this type of misbehavior, and design and implement a new proof of storage to detect this rational behavior. Our *Proof of Timely-Retrievability (PoTR)* aims to assess whether a storage node at the edge is able to retrieve a data object with a latency smaller than some specified threshold δ. We leverage the availability of Trusted Execution Environments (TEEs), more precisely *Intel SGX*, to ensure that the proof is produced by the fog node being audited and to reduce the communication between the auditor and the audited node. Our proof is issued in response to a challenge: we describe how a challenge may be designed and configured, so that a PoTR is obtained. We have evaluated our design experimentally, and we show that the auditor is able to distinguish a fog node that respects the target limit on data access latency δ from a fog node that does not.

Keywords

Edge Computing; Quality of Service; Proof of Storage; Trusted Execution Environment.

Resumo

A computação na periferia da rede é um modelo que coloca servidores na periferia da rede, com o objetivo de auxiliar aplicações que correm em dispositivos com recursos limitados. Os servidores na periferia da rede podem ser usados para armazenar dados de forma a que os clientes possam aceder aos dados com baixa latência. No entanto, como os servidores na periferia da rede têm capacidade de armazenamento limitada, entidades que fornecem serviços de armazenamento na periferia da rede podem ser tentadas a vender maior capacidade do que a que realmente possuem, mascarando este comportamento ao descarregar os dados de servidores na nuvem, a pedido, em vez de os servir com baixa latência.

Nesta tese, estudamos técnicas, propostas na literatura, que permitem ajudar na deteção deste tipo de mau comportamento e desenhamos e implementamos uma nova prova de armazenamento que permite detetar este comportamento racional. A nossa *Prova de Resposta Pontual* pretende avaliar se um servidor na periferia da rede é capaz de aceder a um bloco de dados com um latência inferior a um limiar δ. Tiramos partido da existência de ambientes de execução confiável, mais precisamente do *Intel SGX*, para garantir que a prova é produzida pelo nó que está a ser auditado e para reduzir a comunicação entre o auditor e o nó auditado. Como uma prova é genericamente obtida em resposta a um desafio, descrevemos como é que um desafio pode ser desenhado e configurado para que uma prova de resposta pontual seja obtida. Avaliámos a nossa solução experimentalmente e mostramos que o auditor consegue distinguir um servidor na periferia da rede que respeita o limiar δ no tempo de acesso aos dados de um servidor que não respeita este limiar.

Palavras Chave

Computação na Periferia; Qualidade de Serviço; Prova de Armazenamento; Ambiente Confiável.

Contents

List of Figures

List of Tables

Acronyms

DT	Detection Threshold
FPR	False Positive Rate
FNR	False Negative Rate
PDP	Proof of Data Possession
PoRet	Proof of Retrievability
PoRep	Proof of Replication
PoGR	Proof of Geographic Replication
PoTR	Proof of Timely-Retrievability
SLA	Service Level Agreement
SSHFS	Secure Shell File System
TEE	Trusted Execution Environment
TPM	Trusted Platform Module
ULT	Unacceptable Latency Threshold
P2P	Peer-To-Peer

1

Introduction

Contents

Today, there are many scenarios where an end-user, or an organization stores data in machines run by third-parties, either to ensure durability and availability, or to provide others with low latency when accessing the data. Relevant examples include cloud storage (such as Dropbox, iClould, Google Drive, and many others), peer-to-peer storage (including the storage services used for decentralized applications [1, 2]), content distribution networks (for instance, Akamai), and, more recently, edge storage [3].

1.1 Motivation

When using a storage service, the user has some expectations regarding the quality of service to be provided by the third-party. This, naturally, includes the expectation that the third-party will not discard, or corrupt the stored data, but also that it stores the data while preserving some additional Service Level Agreement (SLA) properties. Those properties may be data storage in multiple distinct machines, data storage in specific geographic locations, or serve data clients with some bounded delay.

Unfortunately, a rational [4–6] third-party may opt to avoid complying to some of these expectations, if it can gain some benefit and pass unnoticed. For instance, the third-party can keep the data in fewer replicas and/or fewer locations than defined in the SLA, assuming that it can be impossible for the customer to audit how many replicas are used or where these replicas are located. This motivated the development of auditing techniques, able to extract proofs of storage, i.e., evidences that the third-party is complying with (or violating) the SLA.

In this work, we address the problem of deriving audit techniques tailored for edge-storage scenarios, in particular, that are able to verify that data is placed by an edge-storage provider in locations that are able to guarantee that data is served with low-latency to end-users [7]. As the capacity of edge nodes is limited [8], providers of edge storage may be tempted to oversell their capacity and to hide this behavior by fetching, on-demand, data from the cloud instead of serving it from local storage.

1.2 Contributions

In this thesis, we survey the main auditing mechanisms that have been proposed in the literature to cope with rational behavior in distributed storage systems and propose a new proof of storage for edge scenarios, that we name *Proof of Timely-Retrievability*. In detail:

- The thesis describes a novel auditing mechanism that is able to extract a proof of timely-retrievability, i.e., a proof that a given fog node is able to serve requests without violating some given data access latency constraint δ.

Our auditing mechanisms is based on a challenge that requires the fog node to access, in sequence, a pseudo-random set of data items and to respond to the challenge in a timely manner. The challenge is

designed in a such a way that, if the fog node does not store locally a significant fraction of the objects, it will be unable to respond in time. To ensure that the challenge is executed by the fog node being audited, and not by some other, we leverage on Trusted Execution Environments (TEEs) supported by modern hardware, namely on the *Intel Software Guard Extensions* (*Intel SGX*) .

1.3 Results

This thesis has produced the following results:

- An implementation of our mechanisms to extract the proof of timely-retrievability for nodes that support the *Intel SGX*;
- An extensive experimental evaluation to assess the accuracy of the produced proofs.

1.4 Research History

This work has been developed in the context of the project Cosmos, that aims at developing a storage system to support edge-computing. In the development of the work I benefited from the discussion and collaboration with other members of the Cosmos team, in particular from working with Cláudio Correia, that is addressing on the security aspects of the project.

Parts of the work described in this thesis have been published as:

- R. Prates, C. Correia, M. Correia and L. Rodrigues. Prova de Resposta Pontual no Acesso ao Armazenamento Contratado na Periferia da Rede. In *Actas do décimo segundo Simpósio de Informática (Inforum)*, Lisboa, Portugal, September 2021.

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) as part of the projects with references UID/CEC/50021/2019 and COSMOS (financed by the OE with ref. PTDC/EEICOM/29271/2017 and by Programa Operacional Regional de Lisboa in its FEDER component with ref. Lisbon-01-0145-FEDER-029271).

1.5 Organization of the Document

The rest of the thesis is organized as follows. In Chapter 2 we present all the background related with our work. Chapter 3 describes the proposed architecture to be implemented, and Chapter 4 describes how we evaluated our auditing mechanism. Finally, Chapter 5 concludes the thesis and highlights directions for future work.

2

Background and Related Work

Contents

In this chapter, we introduce data storage on third-parties, and existing storage architectures, in Section 2.1. At the same time, in Section 2.2, we provide an overview on how an auditing mechanism may be designed, and survey some techniques proposed in the literature that aim to assess the quality of a service of storage, in a cloud architecture. Finally, in Section 2.3, we introduce TEE, as it is a main component of our solution.

2.1 Data Storage on Third-Parties

There are several scenarios where a user may opt to store data on machines controlled by a third-party. In this thesis, we consider three main examples, namely, peer-to-peer storage, cloud storage, and edge storage. Using third-parties for data storage can bring several advantages, including reduced cost (it may be more cost efficient to run a large shared storage service than multiple small storage devices), fault tolerance (by keeping multiple copies of the data, possibly in different geographical locations), and reduced latency when accessing the data (copies may be placed in locations near to the users). However, it also brings challenges, given that the third-party may not comply with the service requirements agreed. In this section, we introduce different scenarios, where storage by third-parties is used, and list their main challenges.

2.1.1 Peer-To-Peer Storage

Peer-To-Peer (P2P) storage is a storage model where different parties coordinate to keep redundant copies of each other files. P2P storage is driven by two main observations [9–11]. In first place, the capacity of storage devices, such as hard-disks, has increased significantly over the years, and many users consume just a fraction of their local storage capacity (the excess capacity remains underused). In second place, it may be impractical for many users to deploy their own devices in multiple locations, a requirement to tolerate the loss of local storage due to theft, fire, or other disasters.

The idea of P2P storage is that users can cooperate to store copies of files from other users, leveraging the spare capacity of each individual storage. This approach has the advantage of being inexpensive and has the potential to provide a high level of reliability. If the users are located in different geographical locations, and multiple copies of a file are created, the likelihood of loosing files can become arbitrarily small. However, as P2P storage relies on reciprocity: one party stores data for a third party under the assumption that the third party will also store data for other parties, free-riding [9–11] becomes a vulnerability to the system. A party benefits from the contribution of other nodes without doing its part, for instance, by failing to locally store copies of files that have been assigned to it and, therefore, data loss.

2.1.2 Cloud Storage

Cloud storage is a model where large service providers offer storage services to the end users for a fee. Cloud storage is widely used today, and there is a large variety of companies that sell this type of service (Amazon, Google, Microsoft, and others). The main drive of cloud storage is the cost savings that can be achieved when the storage servers, and the infrastructure to maintain those servers (power supply, cooling, etc) are shared by a large number of users. This allows cloud storage providers to maintain multiple copies of the data, in different servers, possibly in different locations, for a fraction of the cost that would need to be payed by individual users, if they attempted to build a similar infrastructure for their own use only.

Users of cloud storage pay for the service under the assumption that the cloud provider implements the necessary fault-tolerance techniques to make highly unlikely that data can be lost due to the failure of individual components, such as a hard-drive, or a server. However, it is not easy to the customer to audit if the expected levels of redundancy are actually implemented by the cloud provider [4]. A dishonest provider could be tempted to maintain less copies of the data than advertised, to reduce costs [4, 6, 12].

2.1.3 Edge Storage

There is a range of new tasks, such as image processing for face or object recognition [13, 14], and just-in-time video indexing [15], that require a response time below 5–30 milliseconds [16]. For these applications, it is essential to access data with low-latency, something that cannot be guaranteed with cloud storage alone. In a matter of fact, for a response time below 5–30 milliseconds, a storage node needs to be placed at the center of every circular area with 174Km of radius [17].

Edge computing, that consists in placing computing resources closer to the edge of the network, emerges as a strategy to offer low latency access to processing, and storage resources to these applications. Edge servers, also known as *fog nodes* or *cloudlets*, extend cloud services closer to users. Fog nodes, that can be seen as *micro datacenters in a box* [3], are resource constrained, and are not able to keep a copy of all objects maintained in the cloud [18]. Instead, they will typically keep copies of items that are primarily stored in the cloud. Moreover, due to their limited storage capacity, providers of edge storage may be tempted to oversell their storage and hide this behavior by fetching, on-demand, data from the cloud or from other servers, instead of serving it from local storage. At the same time, due to their geographic distribution, fog nodes are more vulnerable to attacks by malicious players [19] than cloud storage datacenters, and raise concerns about the services reliability due to the different edge storage providers in the network [8]. Figure 2.1 provides an overview of edge storage, where documents are, in the first place, outsourced to the cloud and, then, replicated to fog nodes.

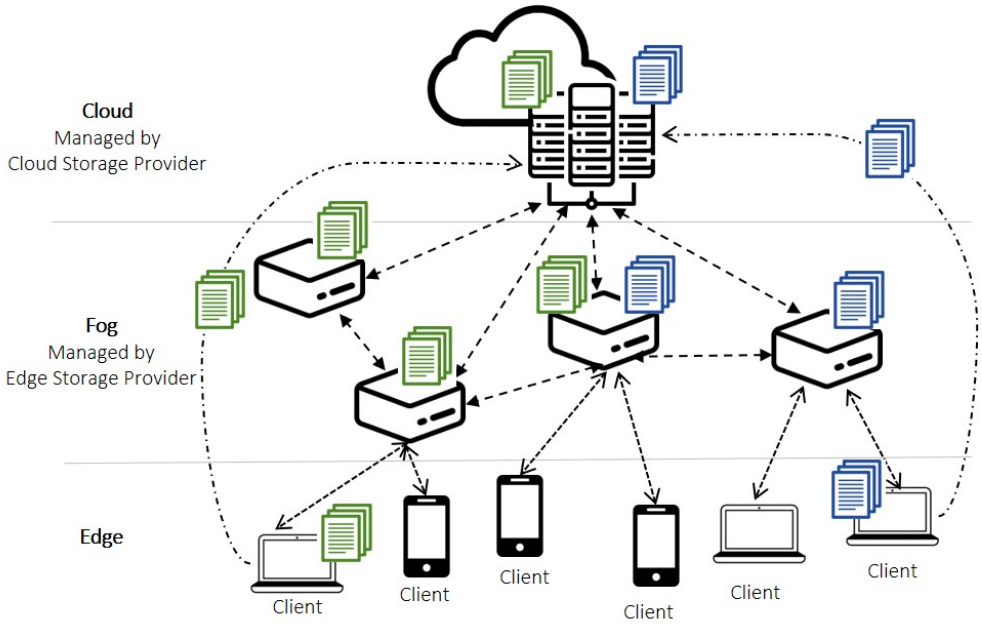

Figure 2.1: Edge storage overview.

2.1.4 Service Level Agreement

In any case, regardless of the type of storage used, the client has some expectations regarding the properties of the service provided by the third party. These expectations are captured by a (implicit or explicit) *Service Level Agreement* (SLA) [4, 5, 20], that lists the requirements the service provider must comply with. In the context of third-party storage services, the SLA may cover the following aspects:

- **Integrity**: the stored documents have not been modified;
- **Retrievability**: the documents are retrievable;
- **Replication**: the provider keeps at least n replicas of an item, in independent storage servers/disks;
- **Geo-replication**: the provider keeps n replicas of an item, in different storage nodes, in different geographically distributed locations;
- **Timely Retrievability**: documents are retrieved within some (typically small) latency δ.

2.1.5 Addressing Rational Behavior

Like any local storage system, third-party storage can be affected by faults that may compromise data availability [5, 6, 20]. Disks can malfunction and loose the data stored on them. A faulty controller can corrupt data when it is stored on disk. A malicious intruder may modify or delete stored data.

Natural disasters may damage the storage equipment. Therefore, these faults can be masked with the help of replication, by keeping multiple copies of each data item on different machines, ideally in different locations, using diverse hardware maintained by different teams, such that the chances of multiple correlated faults happen in a short time interval becomes small.

A significant difference between privately-owned storage and third-party storage is that, with the later, it may be difficult for the user to assess if the desired fault-tolerant and data placement policies are, in fact, deployed by the storage provider. In the absence of appropriate auditing mechanisms, the storage provider may rationally [4] deploy lower redundancy levels than expected, to reduce the costs associated with the additional storage and replica coordination.

As we have noted, in P2P storage, a node can simply not store the data it is supposed to store. When the data is requested, it may attempt to fetch it from another peer. Naturally, if other peers also do this, the chances that data is lost become high. This behavior is unlikely to occur in a cloud service provider, as it will be easily detected as soon as a client would attempt to access the data. However, the cloud provider may be tempted to keep less replicas of the data, in fewer locations, than the advertised [6, 12]. In turn, edge storage providers may have incentives to keep copies of data item in just a fraction of their fog nodes, to augment the capacity they can sell to their customers, at the cost of providing increased latency to data users.

One way to avoid and detect this rational behavior consists in deriving auditing mechanisms that can assess if the third-party storage service keeps the desired number of data copies in the desired locations. When the auditing detects a misbehavior, this can be used to penalize the provider. In a P2P system, a faulty node may be prevented from further using services provided by other nodes [9–11], cloud, or edge storage providers may be forced to compensate clients for violations of the SLA.

2.2 Auditing Third-Party Storage Services

When a storage provider is subject to an audit, it has to provide a proof that it is applying the data replication and data placement policies specified in the SLA. In this thesis, such proof is generically called a *proof of storage* [12]. Moreover, as 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 of storage, each one covering a sub-set of the aspects enumerated above.

In this section, we survey the main proofs of storage, and respective implementations, that have been proposed in the literature. First, in Section 2.2.1, we describe an overview of the different proofs of storage, and SLA aspects that they cover. Then, from Section 2.2.2 to Section 2.2.4, we provide an overview of the components and steps, surveyed from the literature, that build an auditing mechanism.

This overview allows a generic introduction on how an auditing mechanism may be designed, for a better understanding of the mechanisms proposed in the literature, described in the following Section 2.2.5. We have been inspired by the mechanisms proposed in the literature for the construction of our *Proof of Timely-Retrievability*.

2.2.1 Proofs of Storage

Each one of the following proofs of storage as been conceived to prove a different subset of the aspects that can be covered by a SLA, as described in Section 2.1.4.

- **Proof of Data Possession (PDP)** A proof of data possession proves integrity of the stored data;
- **Proof of Retrievability (PoRet)** A proof of retrievability proves the stored data is available to download and recoverable;
- **Proof of Replication (PoRep)** A proof of replication proves that data copies are stored at different servers/storage devices;
- **Proof of Geographic Replication (PoGR)** A proof of geographic replication proves that data copies are stored at different geographic locations.

Although a storage node to build a PDP [21] needs to have available the requested data items, the result of the proof may be a tag, or a cryptography hash over a data item, not proving files full storage. Most files may be stored, but not necessarily all [20, 22]. Therefore, a PoRet [23] proves files full storage [4], and with the use of error correction codes, a PoRet allows files recovery, in case of corruption [24].

	Integrity	Retrievability	Replication	Geo-replication	Timely Retrievability
PDP	✓	–	–	–	–
PoRet	✓	✓	–	–	–
PoRep	✓	–	✓	–	–
PoGR	✓	–	✓	✓	–

Table 2.1: Proofs of Storage vs SLA aspects

Table 2.1 matches the different proofs of storage with the respective covered SLA aspects. As highlighted in the table, none of the proofs of storage previously proposed in the literature addresses timely retrievability.

2.2.2 Obtaining Proofs of Storage

On a first approach, one might think that, in order to perform an audit, one could just read the objects stored and check if the provider is able to return their correct value. This simple method could, in fact, offer a PoRet. Unfortunately, this method has several limitations. In the first place, this approach is very expensive, as it requires the auditor to consume a large amount of bandwidth to obtain the proof. In second place, this technique is unable to verify some of the service requirements, for instance, it cannot assess if an object is kept locally by the target storage node or fetched on-demand from another location, nor is able to detect if the provider keeps just one or several replicas of the object [4, 12, 20]. Therefore, more sophisticated approaches are needed, not only to save bandwidth, but to check the different aspects of the SLA.

Most audits are based on the notion of a *challenge* [4–6, 12, 20], a sequence of tasks that the storage providers need to execute in order to produce a proof, i.e., the content and the timeliness of the response to the challenge serves as a proof. Each task typically requires the provider to perform some cryptographic operation on the content of (a part of) a file, such that it can only obtain a correct and timely proof, if the data is stored locally. A challenge may be set-up in such a way that, in order to respond in a timely manner, the provider must perform several tasks in parallel, which in turn, requires data replication in different media [4]. The number and the sequence of tasks in a challenge depends on the type of proof the auditor aims at obtaining. Thus, the auditor has to design a challenge that ensures the following *security guarantees*: i) the provider cannot re-use outputs from previous audits; ii)the provider cannot pre-compute an answer before the audits starts; iii) the proof is produced by the target storage node; and iv) the proof proves the required SLA aspects.

2.2.3 Components of a Challenge

As a challenge is issued to obtain a proof of storage, a challenge is typically characterized by four main components: i) the set of objects that the nodes need to access; ii) the sequence by which these objects should be accessed; iii) the output that the node should produce (a function of the content of the objects accessed); and iv) a deadline, by which the proof must be produced. We briefly discuss each one of these components next.

2.2.3.A Set of Objects

A challenge could force the storage node to access all objects that it is required to store. However, in most cases, this would be extremely expensive. Therefore, most challenges resort to sampling, i.e., to produce the proof the node only needs to access a subset of the data objects and, from these objects, it may also be required to access some sample blocks. Then, each challenge has to select a different set

of objects/blocks, using some pseudo-random function unknown to the node being audited, such that the node can not predict which objects it needs to maintain to answer a given challenge. The number of objects included in the challenge depends on different aspects, that we will discuss later in this thesis. In any case, this number should be selected such that the probability of producing a proof without having the entire set of objects is negligibly small [4, 20].

2.2.3.B Sequence of Objects

In some cases, the challenger may want that the node being audited accesses the objects in a specific sequence, in particular, the challenger may want to prevent the audited node from knowing which is the next object in the sequence, unless it has already accessed the previous objects. To avoid excessive communication between the auditor and the node being audited, the identifiers of objects to be accessed, except the first object, may be sent encrypted. Then, it is possible to use the content of the i^{th} object to decrypt the identity of the $(i+1)^{th}$ in the sequence, such that the audited node is forced to respect the desired sequencing [4, 6].

2.2.3.C Cryptographic Hashes

For each object, or data block included in the challenge, the audited node may be forced to include the entire content in the proof of storage. Again, this can be very expensive, namely, it may consume a significant amount of bandwidth to transfer the proof. Therefore, most challenges require the audited node to send just a cryptographic hash of the accessed content. The auditor can then check if the hash that is produced as part of the proof matches the genuine content of the data. Naturally, it is important to ensure that the audited node has the original data, and not only a set of pre-computed hashes. For this purpose, the challenge includes a random nonce η, that is unique and that cannot be guessed by the audited node. The hash to be returned as part of the proof is a function of the data stored and the nonce η, and can only be computed in response to the challenge [4, 20].

2.2.3.D Deadline

Most challenges need to be answered by a given deadline, otherwise the audited node would have enough time to download, from another source, the data required to produce the proof. The deadline should be defined such that it can only be met if the audited node has the data, possibly stored in different servers [4]. To define the right deadline for a challenge, the auditor needs to have models that characterize the delays involved in the production of the proof. These include the delays associated with reading data from persistent storage, computing hashes, and communicating with the auditor. As we will discuss, as non-negligible part of defining a challenge, and a proof of storage is related with the correct

definition of such models.

For the time to read a data object from disk, and because storage nodes may be heterogeneous (have different storage media), the auditor may bound this metric with estimation measurements from the storage media available in the market [4]. The same approach could be used to estimate the time required to compute the cryptographic operations required to produce the proof.

Unfortunately, to measure experimentally the latency of the communication between the auditor and the audited node can be difficult without the participation of the target node, which can artificially introduce delays in the experiments to lead the auditor to over-estimate the communication latency which, in turn, would create a slack that the node could exploit to defeat the challenge. Therefore, this latency is often not obtained experimentally, but inferred from other parameters.

A technique that is often used consists in estimating the network latency from the physical location of the nodes involved in the communication [6]. In a first step, a metric of physical distance between the two nodes is obtained. Then, the physical distance is used to predict the network latency. These predictive models are built with the help of a set of storage nodes in known locations to the auditor, referred as *landmarks*.

The distance between two nodes in the network may be determine with the the following distance measurements [6]:

- **Haversine Distance**: distance between two points in a sphere;
- **Driving Distance**: distance between two cities, using Google Maps API;
- **Topology-based Distance**: distance between hosts in the Internet based on the network topology.

Then, the distance can be used to estimate the latency, using techniques such as [6]:

- **Speed of Light**: $\frac{4}{9}\times$(speed of light) as an upper bound for Internet latency;
- **Best Fit Line**: with the set of sampling measurements (distance, latency) between landmarks, use a linear regression function to obtain the best fit line that better accommodates these values;
- **Site Expected**: with latency measurements, for a given node, calculate the linear regression of these points.

Note that this estimated network latency is a lower bound on the communication delay, as this delay as other components, such as the delay introduced by buffering routers.

2.2.4 Phases of a Challenge

Typically an audit is composed by the following three phases: i) challenge setup; ii) challenge execution; and iii) response verification. We now describe in more detail each one of this phases.

2.2.4.A Challenge Setup Phase

Before any storage node executes a challenge, the responsible auditor has to prepare the challenge, according to the requirements to be proven. Thus, the auditor selects set of objects to be read, together with their retrieval sequence. The auditor also generates one, or more nonces to be used in the cryptographic hashes, and selects the required parameters to set an accurate deadline.

2.2.4.B Challenge Execution Phase

With the challenge ready, the auditor sends it to the target storage nodes. Each node will read, from local, or remote disk, the requested data objects, compute their cryptographic hashes with the nonce, or nonces. Then, it will send the final response to the auditor. The auditor will measure the delay the storage nodes take to compute, and send the final response.

2.2.4.C Response Verification Phase

As soon as the auditor receives a response to the challenge, the auditor verifies its correctness, and timeliness. To verify the proof correctness, the auditor executes the same sequence of steps that a storage nodes had to execute during a challenge (Section 2.2.4.B), i.e., read the sequence of required data objects, and compute their cryptographic hashes, etc, to obtain a verification version. With this verification version, the auditor checks the proof correctness. If the received proof, and the verifiable version match, the proof is assigned as correct.

Then, for a correct proof, the auditor checks the proof timeliness. If the proof was received within the defined deadline, the proof is considered valid and, accepted. Accordingly, the storage provider is assigned as complying with the agreed SLA.

It is important to highlight that for any audit, it is mandatory that the clients data files are outsourced to the storage nodes, in the first place, independently of SLA compliance, and that the auditor has knowledge of the outsourced data files. The auditor can have a copy of the data files, a summary of them, a list of tags, etc. It is also required that the cryptographic hashes to be used, and the challenge procedures are known to the storage nodes. The procedures for agreement may happen during the setup phase, and be the same for all challenges, or can happen every time there is a challenge, and may be different per challenge. In this thesis, we define this two steps as part of the challenge setup phase (Section 2.2.4.A).

2.2.5 Challenges Previously Proposed in the Literature

In the following sections, we describe a set of different challenges proposed in the literature. For a better understanding, we subdivided each implementation based on the phases surveyed in Section 2.2.4.

2.2.5.A Time Bounded Challenge with *Intel SGX*

Dang et al. [20] describe a time bounded challenge with *Intel SGX* to check *data residency*, i.e., to assess if the outsourced data is retrievable *in its entirety from local drives of a storage server in-question* [20]. In other words, *Dang et al.* provide a scheme to obtain a PoRet, and their solution is divided in two architectures. At the same time, if their solution is applied to different storage nodes, in different geographic locations, a PoGR can be obtained.

At challenge setup, a file F is divided into n data blocks, with a given size. Each data block is, later, encoded with a standard error correction code, so that from any set of n data blocks, the file F may be reconstructed. Then, each encoded data block is concatenated with a unique identifier (a hash of the block index, together with a secret key s, only known to the auditor, and the unique file F identifier). In turn, the auditor outsources the hashed data blocks to the storage node, discards the original file F, and keeps the secret key s, together with some metadata for the verification process.

A storage node, when executing the challenge, has to read the challenged data blocks, and compute, and send the response within the deadline, for a valid proof. As *Dang et al.* discuss a audit based on a set of challenges, the auditor sets a second threshold, beyond the challenge deadline. This second threshold defines the number of challenge responses that can be received after the deadline. In case this threshold is reached, the obtained proofs are rejected and the storage node is assigned as not complying with the SLA.

The deadline is defined based on the data fetching time, together with the delay to transmit the challenges and the proofs. However, *Dang et al.* diverge on the deadline definition for each one of their two architectures. In the first one, the auditor and the storage node are distant to each other, which means the challenge and the response are transmitted via the network. As for the second architecture, both the auditor and the target node are co-located in the same machine. This is only possible due to TEE (*Intel SGX* in this case), since the auditor data and code must be secure, otherwise, the storage node could know the selected blocks to be challenged a-priori and, download then from a remote storage, hiding its dishonest behavior. Therefore, in this second architecture, the response deadline is defined based on the delay the storage node takes to exchange the execution environment between the *Intel SGX* components.

To obtain, a PoGR, the auditor requires to previously know the location of the storage node. Since the second architecture uses a trusted execution environment, when setting the auditor code and, due to the attestation process, the auditor has guarantees that the challenge is executed at the expected

storage node, and not by another one [8, 25].

2.2.5.B Time Bounded Encoding Challenge

Benet et al. [12] provide a time bounded encoding challenge to check data replication across different storage nodes, i.e., to build a PoRep.

At the setup phase, to guarantee independent storage of n replicas of data D, each storage node (that belongs to the hired cloud storage provider) encodes the data replica stored locally with a different key, creating n unique encoded replicas between all nodes. The used key may be randomly chosen, or derived from the storage node and replica identifiers.

Then, when executing the challenge, each storage node has to prove compliance with the pre-defined encoding, leading all nodes together to prove local storage of the n unique replicas. However, as a storage node may discard the assigned replicas, and fetch them on-demand from a neighbor node, or re-encode them from D, the PoRep must be produced on time. A PoRep is only valid if it is correct, i.e., proves data storage at the expected replica out of the n existing ones, and it was built within a deadline. The correctness and timeliness of the proof is verified by the auditor, during the response verification phase.

In this challenge, *Benet et al.* set the deadline as the network delay plus the delay to retrieve the already encoded replicas. As the n replicas may not be stored locally, or the storage node needs to re-compute them by re-encoding D, the extra network, or computation delay will exceed the deadline, and the proof will be considered invalid. Therefore, the encoding function is the main point to detect a rational cloud storage provider. The encoding function must be slow enough, to detect re-encoding on-demand, by the target node or by a neighbor one. According to *Benet et al.*, block cipher with block chaining and layering is the more accurate encoding function: it provides sequential encryption, where each cipher block in each iteration depends on the ciphered blocks from previous ones. By using this function, the auditor guarantees that data encryption can not be parallelized.

2.2.5.C Parallel and Sequential Time Bounded Challenge

Li et al. [4] introduce a parallel and sequential time bounded challenge to build a proof of physical reliability (PoPR). A PoPR is equivalent to PoRep, as it proves data replication in independent storage. At the same time, a PoPR may be extended to prove data geo-replication and obtain a PoGR.

During challenge setup, each file F is broken into n_s subfiles and encoded with an error correction code, producing a verifiable version, which is later subdivided in n l-bit blocks, referred as *symbols*. Later, symbols are distributed across the cloud storage provider n storage nodes, defining a symbols layout L, that must be followed by the cloud storage provider. As each storage node will keep one symbol of each subfile, when retrieving a document, the cloud storage provider must access all storage nodes

in parallel, and retrieve serially symbols in the same storage node. Figure 2.2 provides an overview a storage layout.

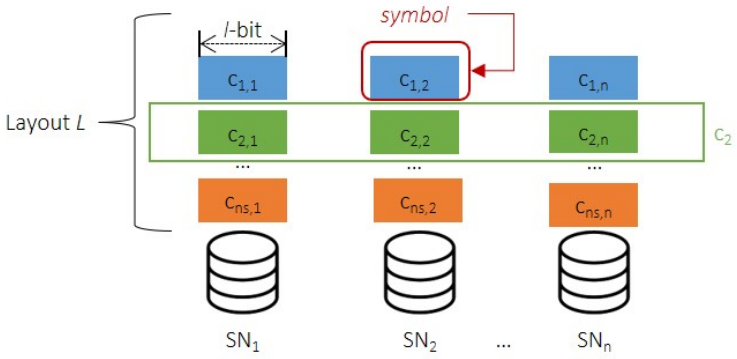

Figure 2.2: Storage layout L between n storage nodes. SN_i is a storage node; c_i is the verifiable version of a subfile f_i, and $c_{i,j}$ is a l-bit block, also described as *symbol*.

Once a rational cloud storage provider may store symbols in fewer nodes than the defined in the layout L, when executing the challenge, more symbols need to be read serially (i.e., in the same storage node) than in parallel, leading the challenge to take longer than expected by the auditor, and allowing it to detect a misbehavior. Therefore, for each group of requested symbols, per storage node, the auditor sets a deadline based on the symbols processing delay, and the network delay between the auditor and the storage node.

The processing delay is defined as the delay to read from disk the symbols expected to be stored locally plus the computation delay to hash each symbol, and build the final proof. However, due to hardware characteristics, the processing delay is mainly bounded by the delay to read data from disk. At the same time, as storage nodes may be heterogeneous, the auditor can not set a strict deadline for the reading delay. Instead, the auditor sets the delay between loose boundaries: the upper bound is the fastest data access rate known so far, plus an error margin, and the lower bound, the lowest reading delay registered from that storage node, plus a margin. These loose boundaries guarantee that the processing delay is hardware independent, and, thus, even a storage node with a fast disk access is unable to mask non-compliance with the storage layout L. As for the limit network delay, the auditor, based on given samples distribution of the network delay, bounds this parameter between a low and high standard deviations from expected values, or a minimum and maximum values from the samples, plus an error margin.

At challenge setup, each storage node is assigned with a different deadline, due to the different number of symbols to be retrieved from each node. Therefore, when issuing a challenge, the auditor must mask the challenge length. Otherwise, the cloud storage provider detects which is the smallest challenge, to be the one to be executed first, and just then executes the larger challenges. As the larger challenges will have a larger deadline, even if the cloud storage provider processes the smallest

challenge first, it could compute valid proofs (correct and on-time) for both challenges. The challenge length can be hidden by creating a dependency between the symbols to be retrieved (the next one to be retrieved is only known after the previous one is retrieved), and by hashing the last symbol to be retrieved with a nonce η. A challenge is only finished when the cryptographic hash of the retrieved symbol with η matches the cryptographic hash of the last one. With the challenge length masked, a dishonest cloud storage provider would randomly choose the challenge to be selected first, which increases the probability to detect a misbehavior cloud storage provider, for a set of challenges.

Li et al. extend their PoPR to prove data geo-replication, through applying the above mechanism to storage nodes in different known geolocations, and obtain a PoGR. If the response to a challenge is not received within the deadline, it is assumed that the client files are not kept locally, and the data is not in that location. To minimize the network latency, the auditor may be placed geographically close to the target nodes.

2.2.5.D Data Geolocation and Time Bounded Challenge

Benson et al. [6] introduce a geolocation and time bounded challenge to obtain a PoGR.

As each data block is stored together with an authentication tag, storage nodes are challenged to prove data storage of both elements in less than T, where T is the execution time of some challenge plus the network delay. With a low enough deadline T, the auditor ensures the target storage node does not reach to a neighbor node to answer the challenge, i.e., the challenged storage node must be the one computing the proof. At the same time, as the storage nodes have to retrieve the maximum number of random blocks that can be retrieved in T, any extra network delay to read the data blocks from another node will exceed the deadline. Note that the maximum number of data blocks to be retreived, and the deadline T are chosen by the auditor at the challenge setup phase.

On the other hand, to assess data geo-replication, the auditor, also at challenge setup phase, with a set of landmarks, builds a latency-distance predictive model (Section 2.2.3.D), that is later applied to storage nodes in unknown locations. *Benson et al.* discuss the three models, previously introduced, with Haversine and Driving Distance, and according to them, and as expected the Best Fit Line and Site Expected models with Haversine distance outperform the Speed of Light model, when retrieving content from 40 universities hosts in North America. The $\frac{4}{9}$ the speed of light is a large upper bound for latency, leading to inaccurate distances and geolocations.

Benson et al. extend their PoGR by providing a geolocation model landmarks independent. Note that the above predictive models require a set of trustworthy landmarks to communicate between each other to build a latency-distance model. Therefore, *Benson et al.* discuss a new approach based on IP analysis of the retrieved data blocks, to detect datacenters possible location: if for a set of data requests the storage nodes IPs are geo located to the same region, then, most likely in that region there is a

datacenter.

2.2.5.E Constraint-based Data Geolocation Challenge

Gondree et al. [5] provide a constraint-based data geolocation (CBDG) challenge that combines a PDP with constraint-based geolocation [26], to build a PoGR.

At setup phase, documents are broken into n blocks and tagged with a message authentication code (MAC), and later stored at cloud storage provider storage nodes. Before executing an audit, the auditor to assess data integrity and ensure the proof output is not reused from a previous audit, selects c random indices referring the stored blocks, and tags to be retrieved. At the same time, the auditor requests a set of trusted landmarks to build a latency-distance predictive model between them that they will later apply. Due to its simplicity, *Gondree et al.* considered the Best Fit Line model, from Section 2.2.3.D, as the latency-distance predictive model.

Next, when starting the challenge execution, the auditor forwards a challenge (the c random indices) to the landmarks that will, in turn, challenge the target storage node. For each challenge, each landmark will record the response delay, and together with the pre-built predictive model estimate its distance d to the target storage node. Then, already at the response verification phase, with the distance d, each landmark computes the circular constraint region, with radius r that forwards to the auditor, together with the storage node response to the challenge.

With the radius r from all landmarks, and the responses, the auditor checks responses correctness, and if they are correct, computes the intersection between the different circular regions. This intersection will result in the most feasible region for the storage node geolocation. Therefore, this method allows to geo locate storage nodes in unknown locations. Figure 2.3 provides an overview of the computed most feasible region for a storage node location.

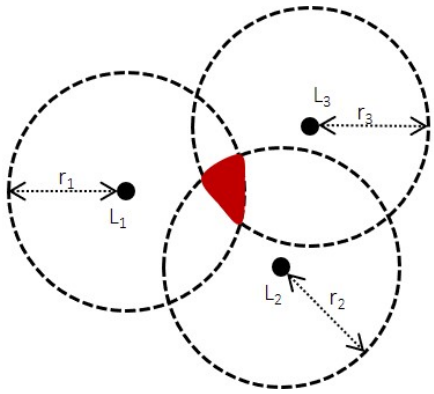

Figure 2.3: Most feasible region (red area) for a storage node location from radius measurements of three landmarks. L_i is a landmark and r_i the radius of the circular constraint region.

2.2.6 Discussion

In this section, we summarize and analyze the different challenges proposed in the literature, to help us to understand how we can design an accurate challenge to obtain a Proof of Timely-Retrievability (PoTR).

Challenge	PDP	PoRet	PoRep	PoGR
Time bounded with *Intel SGX* [20]	✓	✓	–	–
Time bounded encoding [12]	✓	✓	✓	–
Parallel and sequential time bounded [4]	✓	–	✓	–
Data geolocation and time bounded [6]	✓	–	✓	✓
Constraint-based data geolocation [5]	✓	–	✓	✓

Table 2.2: Proofs of storage provided by each challenge in the literature.

Table 2.2 provides an overview of the proofs of storage provided by each implementation, in the literature. Note that the surveyed implementations apply to a cloud storage architecture, that is unable to satisfy clients data requests with low latency. Therefore, none of the previous challenges proves timely retrievability.

Challenge		Set of Objects	Sequence of Objects	Cryptographic Hashes	Deadline
Time bounded with *Intel SGX* [20]	Auditor distant to target node	all encoded data blocks	sequential but in random order	encoded data blocks	reading delay plus computational delay plus network delay
	Auditor co-located to target node	all encoded data blocks	sequential but in random order	encoded data blocks	reading delay plus computation delay plus hardware calls delay
Time bounded encoding [12]		all replicas	sequential	encoded replicas	reading delay plus network delay
Parallel and sequential time bounded [4]		random number	sequential at the same node, in parallel between nodes	data block with nonce	reading delay plus computational delay plus network delay
Data geolocation and time bounded [6]		random number	sequential	data block and tag	reading delay plus network delay
Constraint-based data geolocation [5]		random number	sequential	data block and tag	–

Table 2.3: Components of the challenges proposed in the literature.

Table 2.3 summarizes on how each implementation in the literature satisfies each one of the components of a challenge, introduced in Section 2.2.3.

In the first place, note that in [12, 20], as the cloud storage provider has to prove storage of all data blocks/replicas, the storage nodes are forced to retrieved all data items. However, as in [4–6] the auditor selects a random number of data blocks to retrieve, a challenge will only address a sample of the remotely stored data. Nevertheless, in these three cases the number of challenged data blocks will determine the probability to detect a rational cloud storage provider: larger the challenge, longer the storage nodes will take to respond. Then, a proof will only be within the deadline if the accessed data items are kept in local storage.

Second, in [12, 20], the auditor requests all encoded data blocks/replicas, and in [5, 6] the storage nodes have to answer the data items, together with their tags, these audits bring extra overhead to the network, and are more expensive. Therefore, the cryptographic hash computed by the storage nodes in [4] is a more accurate solution, as it is cheaper, and does not overload the network, at the same time it allows to assess data integrity.

Finally, note that all implementations define the deadline based on the reading delay, and the communication delay between the auditor and the audited nodes. In [20] first architecture and [4–6, 12] the communication is done through the network, thus, time measurements are subject to the network variance and noise. Nevertheless, as in the second architecture of [20] the auditor is co-located with the audited node, the communication delay is much lower, as it is done inside the same hardware. As a consequence, the delay measurements are expected to be more accurate, due to the absent of network noise and variance.

Security Guarantees: in the implementations surveyed form the literature, storage nodes have to access a set of random data blocks, or have to comply with a given encoding. Therefore, the auditor has guarantees that the storage nodes cannot reuse outputs from previous audit, and that the proofs are not pre-computed before an audit. Simultaneously, as the auditor defines a deadline, or estimates the node geolocation, the auditor ensures the storage node answering the challenge is the audited one. Note that, although, storage nodes geolocation may not determine nodes location with high accuracy, it typically allows to distinguish storage nodes in different countries, which in most cases is enough to comply with geographic replication aspect.

2.3 Trusted Execution Environments

In Section 2.2.5.A, we briefly introduced the notion of a Trusted Execution Environment (TEE). We now present the concept in more detail, with a focus on the TEE technology we use in this thesis, *Intel SGX*.

A TEE is an isolated environment that can offer stronger security properties for code and data, such as integrity, and confidentiality [25, 27]. A processor with a TEE has two execution modes: the *normal mode*, where the operating system, and applications run, and the *secure mode*, that ensures the integrity and confidentiality of data and code inside the secure mode, even in the presence of an untrusted operating system. Typically, a TEE comes with special cryptographic material protected at the hardware level such as unique keys produced during manufacturing [19, 27], usually used to correctly identify a TEE. These unique keys can also be used to uniquely identify devices, or CPUs [25]

There are a set of different TEEs architectures, for instance, *Intel Software Guard Extension* (*Intel SGX*), and *ARM TrustZone* [7, 19]. In this thesis, we resort to *Intel SGX* technology, as it is the TEE technology available in Intel systems (PCs, and servers), and we use Intel machines in our experimental

setup.

2.3.1 Intel Software Guard Extension

Intel SGX is a TEE technology available in Intel processors. *Intel SGX* splits the computing environment in two parts: the trusted parts, a set of TEEs or *enclaves*, and the untrusted part. Applications in secure mode run inside the enclave, while applications in normal mode run in the untrusted part. Furthermore, as the processor core can only execute one environment at a time, the exchange of environments happens through hardware calls, more precisely *ECalls* and *OCalls*. A *ECall* exchanges the execution environment from the untrusted part to the enclave, while an *OCall* exchanges the execution environment from the enclave to the untrusted part [7, 8]. Figure 2.4 provides an overview of *Intel SGX* components, and the executed calls for execution mode exchange.

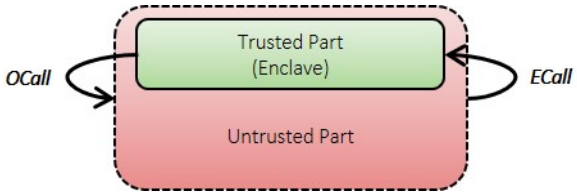

Figure 2.4: *Intel SGX* components and execution mode switching calls.

Apart from code, and data security, *Intel SGX* provides security guarantees about the machine identification. In other words, any external machine that communicates with the enclave has guarantees that is communicating with a real enclave, due to the *attestation process* [8, 25].

The attestation process brings security guarantees that an enclave has been correctly instantiated in an Intel platform with *SGX*, with a correct software version [8, 28]. In other words, the attestation process ensures a client that it is communicating with a given code, inside an enclave, and not an impostor. *SGX* has a security module that manages short-term, and long-term cryptographic keys. With these keys, the enclave is able to produce messages authentication codes (MACs) and digital signatures that uses to authenticate its messages. Thus, an external machine that communicates with the enclave can verify the authenticity of the MACs, and the digital signatures with an Intel server and gain security guarantees that is communicating with a trustworthy enclave [8, 29].

With *Intel SGX* we can ensure data, and code security in fog nodes. *SGX* enclaves have already been used to provide secure storage at the edge [8], however, they incur additional latency overhead and require complex solutions to avoid the memory limitation impose by the *SGX* technology. In fact, storage solutions with *SGX* still require the data itself to be stored in the untrusted zone [8, 30], and, therefore, are still subjective to rational behavior, as described in Section 2.1.5.

Summary

In this chapter, we introduce three architectures for third-party storage: P2P, cloud storage, and edge storage. We discuss their advantages, and disadvantages, more precisely, that a third-party storage provider may have a rational behavior, to save resources, and reduce costs, at the cost of not complying with the agreed quality of service.

We also surveyed a set of mechanisms that allow to audit a third-party storage provider, and, thus, detect a rational behavior. We described a set of existing proofs, from the literature, for different types of rational behavior, and explain on how an auditing mechanism may be designed, so that a given proof can be obtained. Moreover, we discussed in more detail a set of auditing mechanisms that aim to access service compliance by a cloud storage provider.

Finally, we introduce the notion of TEEs, and *Intel SGX* technology, as it is an important component in our PoTR.

In the next chapter, we introduce our PoTR, and the designed auditing mechanism. We have been inspired by the schemes introduced above to design this new auditing mechanism.

3

Proof of Timely-Retrievability

Contents

In this chapter, we introduce our *Proof of Timely-Retrievability*. This mechanism allows users to assess whether a storage node at the edge is able to return their documents with latency below a certain threshold. We begin by introducing the design goals in Section 3.1 and the main obstacles we need to face in Section 3.2. Section 3.3 describes the system assumptions. Finally, in Section 3.4, we describe how a proof of timely-retrievability is obtained, more precisely, the system architecture, the challenge design and configuration, and the reading delay estimate.

3.1 Design Goals

The goal of a proof of timely-retrievability is to assess whether a node can retrieve stored data with a latency smaller than some specified threshold δ. The value of the δ parameter can be selected by the auditor 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 node keeps the data locally. For instance, many edge services and applications require response times below 5–30 milliseconds [16], thus, a fog node that stores data to serve these applications needs to be able to retrieve data with a latency in the order of the milliseconds. More precisely, our proof covers the SLA aspects of:

- **Integrity**: the outsourced documents are effectively stored by the audited node and they have not been modified;
- **Timely Retrievability**: the outsourced documents can be retrieved by the audited node within some (typically small) latency δ.

As our proof of timely-retrievability is obtained as a response to an audit, it is necessary to deploy a machine with auditing capabilities, i.e., with the ability to, based on a set of measurements, conclude whether the edge storage provider complies with the expect SLA. However, as fog nodes are in a large number, and geographically distributed [31, 32], we also aim at achieving:

- **Independence from the Auditor Location**: The proof can be extracted by any node in the internet, regardless of its location.

This property offers a lot of flexibility regarding the deployment of the auditor and allows for a single auditor to perform audits on a large number of fog nodes.

Note that we could have opted to design a proof of timely-retrievability requiring the auditor to be placed near the audited node. For instance, if the auditor is placed in the same network of the data clients, it can observe directly the actual latency experience by the clients. Unfortunately, the cost of auditing a system with a large number of fog nodes in this way can be very high, as it requires to move the auditor or to deploy a large number of auditors. We could also ask clients to report the

latency they observe and use this information to perform the audit. However, this approach raises many challenges as client privacy needs to be preserved [33]. If a proof of timely-retrievability can be extracted independently of the auditor location, we avoid these limitations.

3.2 Obstacles

In the design of our Proof of Timely-Retrievability we face the following obstacles:

- Timing information provided by audited node cannot be trusted [34]; the time to produce the proof must be measured by the auditor;
- The network between the auditor and the target fog nodes is subject to variance; this may introduce an error when estimating the time the audited node took to produce the proof;
- Fog nodes are heterogeneous [31, 32] and the time they require to perform computations and read data (even if the data is local) is not constant; this suggests that the proof should be based on average values of multiple readings;
- 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; we need to ensure that the proof is produced by the audited node and not by some other node.

3.3 Assumptions

The protocol to extract a Proof of Timely-Retrievability is executed among two nodes, an auditor and an audited fog node. We make the following assumptions regarding these nodes and the network that connects them.

The auditor knows exactly which files are assigned to the fog node and is also aware of the content of those files. Our algorithm is executed over a set of immutable files. Mutable files can be supported by creating new versions of a given file (where each version is immutable).

The fog node is considered correct if it can retrieve the files assigned to it with a latency smaller than some given threshold δ. A fog node that cannot satisfy this requirement is denoted to be faulty. The value of δ 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 (for instance, via a SATA bus).

The auditor is aware of the latency distribution observed by a correct node, when accessing data items. The auditor is also aware of the distribution of the time it takes for a correct fog node to perform the computations required to produce the proof. As we will see in Section 3.4.4, these distributions are

used by the auditor to decide how many samples it needs to perform (i.e., how many objects the fog node is required to read), such that the resulting average is accurate.

The auditor is able to obtain a characterization of the network connecting it to the fog node, and has access to the latency distribution of the messages exchanged between the two nodes. This can be achieved at reboot by a process, where the auditor makes multiple requests to the audited node and records the round-trip times observed.

We also assume that each fog node has an Intel processor with *SGX* extension: we use guarantees provided by a trusted execution environment (Section 2.3) to support the execution of the challenge at the fog node. We assume that, before any audit, the enclave authenticates towards the auditor, through an attestation process [19], and that the auditor has guarantees that is communicating with the expected enclave [25]. At the attestation process, the enclave and the fog node exchange a set of secrets: a symmetric key (unique per fog node) and the cryptographic functions to be used during an audit. At the same time, we assume that the integrity and confidentiality of the data and code inside the enclave are guaranteed [8].

We equally assume that the fog node, more precisely the enclave, is unable to read an accurate and reliable system clock, due to the following issues [34]: (1) the fog node system clock is vulnerable to manipulations by the untrusted part; (2) the operation to read trusted timers, like the ones provided by *SGX*[1] or, Trusted Platform Module (TPM), has a huge overhead; and (3) the untrusted part may maliciously delay trusted timer messages, to fetch on-demand remote data blocks and, thus, escape detection. As this being said, it becomes impracticable to use the enclave for time measurements. Nevertheless, the enclave provides us with security guarantees that the proof is effectively built at the challenged node and not at a neighbor one [25].

Finally, we assume that is not economically feasible for the edge storage provider to reallocate all objects in less than δ at the beginning of the audit, from a remote storage location that cannot satisfy the δ threshold on access latency to a nearby storage location that can satisfy δ, i.e., the cost to reallocate all objects every time there is an audit would exceed the costs of maintaining data objects in a location with a access time within δ.

3.4 Design and Implementation

In this section, we describe the design and implementation of our proof of timely-retrievability. As surveyed from the literature, in Section 2.2.5, most proofs of storage are obtained as a response to a single or a set of challenges. Thus, our proof of timely-retrievability is, likewise, obtained as a response to a challenge, that we follow describe.

[1] *Intel SGX* has access to a trusted timer provided by the *Intel Converged Security and Management Engine* (CSME) [34]

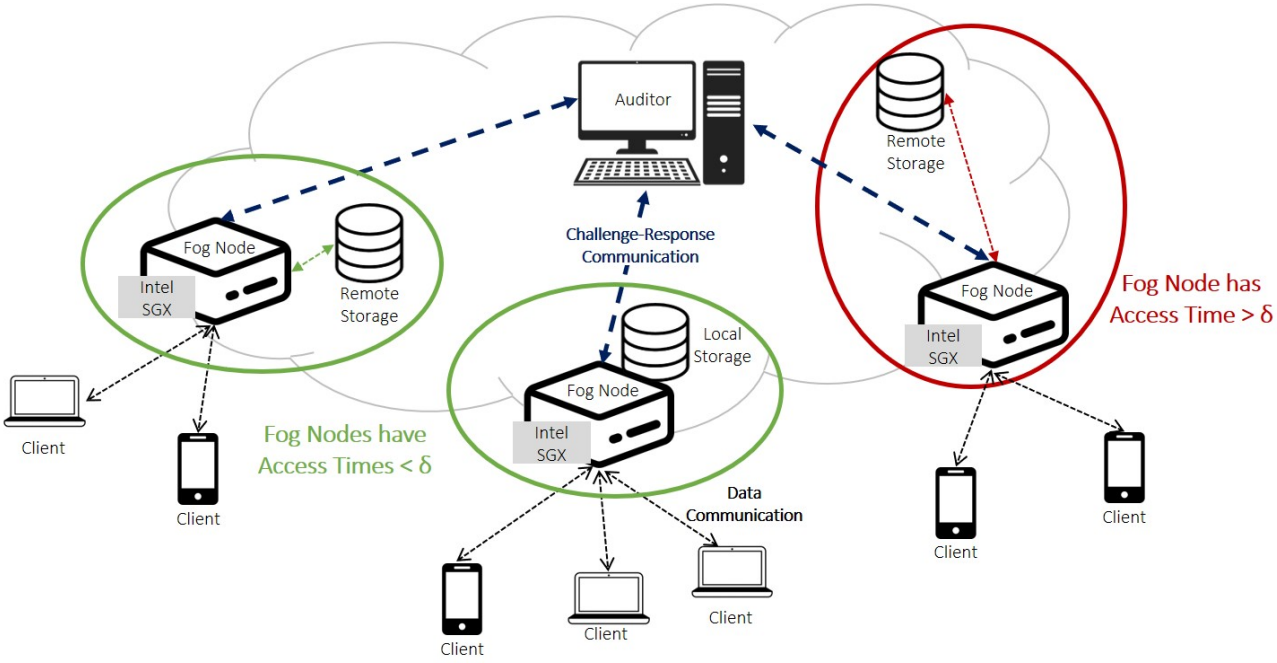

Figure 3.1: System architecture and communication channels.

We begin by describing the system architecture, in Section 3.4.1, followed by the challenge design in Section 3.4.2, the δ estimate in Section 3.4.3, and the challenge configuration in Section 3.4.4. In these sections, we describe in detail on how the challenge is built and configured, i.e., the sequence of data blocks, the cryptographic hashes and the reading delay estimate, in order to meet the design goals from Section 3.1 and overcome the design obstacles introduced in Section 3.2.

3.4.1 System Architecture

Our proof of timely-retrievability is computed by a fog node to prove that it can access data with a latency that is smaller than a given agreed threshold δ. The proof is the response to a challenge, sent by a remote auditor. Thus, the system architecture includes an auditor, a set of fog nodes and a set of remote storage systems, connected to the fog nodes. Simultaneously, multiple and heterogeneous clients are connected to the fog nodes, that may satisfy clients data requests. Figure 3.1 provides an overview of the system architecture, together with the main data flows.

In addition, as already described in Section 3.3, each fog node has an Intel processor with *SGX* extension, that has two components: a trusted execution environment, also known as *enclave*, and an untrusted part. The enclave is trustworthy to the auditor, and, therefore, it is the fog node component that communicates with the auditor. The enclave is trustworthy to the auditor due to the attestation process, where the enclave authenticates towards the auditor [19], and receives its respective symmetric key and the set of the cryptographic functions to be used during an audit. At the same time, due to the

attestation process, the auditor has guarantees that the code running inside the enclave is the expected one, and that this code is protected from the remaining operating system. The untrusted part, in its turn, is responsible to keep files in local storage and communicate with remote nodes, if necessary. As the name describes, the untrusted part is not trustworthy to the auditor. Figure 3.2 describes the *Intel SGX* extension structure and an overview of the messages exchanged between its components for data access.

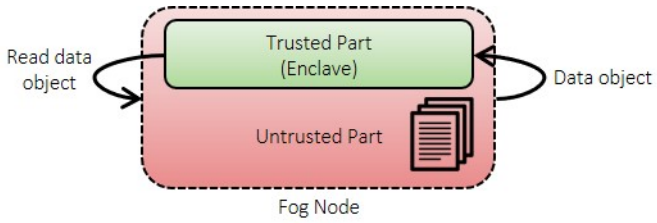

Figure 3.2: Exchanged messages between *Intel SGX* components for data access.

3.4.1.A Fog Node Storage Organization

The edge storage provider is responsible for storing the files at the fog layer. To fulfill the SLA, the provider must ensure that files are stored in such a way that they can be retrieve by the fog node with a latency smaller than δ, the threshold for the access time.

At each fog node with *Intel SGX* extension, local documents are kept at the untrusted part, as the enclave storage capability is limited [8]. Thus, if the node is running on trusted mode (enclave), there must be an exchange between execution environments (from enclave to untrusted part) to read a local data object. For remote documents, there is, also, an exchange of execution environments (once again, from enclave to untrusted part) and it is the untrusted part the one responsible to communicate with the remote machine. The exchange between execution environments occurs via hardware calls: *ECall* and *OCall* [19].

The set of files that needs to be stored by a fog node and that, therefore, can be subject to the auditing procedure, is common knowledge to both the auditor and the audited node. Both parties have a copy of the set of files, i.e., files identifiers, sizes and contents. Thus, both parties can sort the set of files in a deterministic manner (for instance, using the alphabetic order of the file path names) and then use the index of a file in the sorted list as a mutually agreed short unique identifier for that file. We denote this index as the *set index*. As it will be clear in the following discussion, identifying files using their set index simplifies the execution of the auditing procedure.

Note that the above organization is only possible if both the auditor and the audited node agree on the set of files that can be subject to auditing. The mechanisms required to negotiate which set of files can be subject to audit is orthogonal to the contributions of this work.

3.4.2 Design of the Challenge

The challenge requires the fog node untrusted part to access a given number N of data objects, with a given sequence and return, at the end, a value related to the retrieved data objects. *The delay the fog node takes to read the expected data blocks and to compute the final value is used by the auditor to estimate the reading delay δ observed at the audited node.*

Each challenge (implicitly) specifies a sequence of files that needs to be accessed by the audited node. In the challenge, each of these files is uniquely identified by its *set index*, as described in the previous section. For a matter of simplicity and efficiency, instead of retrieving the whole document, the fog node, for each document to be read, only reads a data block with size s_b, a parameter that can be configured when the challenge is defined (usually, a multiple of the block size used by the fog node file system).

In each challenge, the fog node has to read a pseudo-random and unpredictable sequence of N data blocks (each with size s_b), and return a cryptographic hash of all retrieved blocks. The number N of data blocks to be read, as well s_b, are challenge configuration parameters, selected by the auditor during setup phase. It is important to note that higher the value N, more accurate, but less efficient, will be the proof. In Section 3.4.4, we describe on how to choose an accurate number N of data blocks, to guarantee a proof with a given reliability.

The pseudo-random sequence of data blocks is determined by a nonce and the data blocks content. For a given challenge c, at setup phase, the auditor generates a unique nonce η^c and sends it, in a secure manner (i.e., encrypted with the symmetric key sk) to the enclave. Together with the previous nonce, the auditor, sends to the enclave, the number N of data blocks and the block size s_b. With the nonce η^c, the enclave computes its cryptographic hash ($h(\eta^c)$) and forwards the result to the untrusted part. The untrusted part, given the size M of the metadata vector and a module function (mod), computes $h(\eta^c) \bmod M$ that will correspond to a set index and, thus, the first file to be accessed.

Simultaneously, as the challenge requires data blocks with size s_b and files might be larger, the auditor generates a second nonce, η_b^c to determine which data block inside a file is to be accessed. At the same time, this second nonce brings higher randomness to the challenge. As the first nonce η^c, this second one is also generated at setup phase, by the auditor, sent in a secure manner to the enclave and hashed ($h(\eta_b^c)$) before being forward to the untrusted part. Both nonces are hashed by the enclave, to ensure the untrusted part does not know them in plain text. Otherwise, the untrusted part could maliciously pre-determine the N data blocks.

Now, with $h(\eta^c) \bmod M$ the untrusted part knows the first file. With $h(\eta_b^c)$, the block size s_b, the file size s_f and a module function, the untrusted part knows the data block with size s_b, inside the first file to be retrieved. The data block index is given by $h(\eta_b^c) \bmod (s_f/s_b)$. Figure 3.3 provides an overview on how the above hashes identify the first data block. For the set of files at the metadata vector, and

the given block size s_b, each document may be divided in s_f/s_b blocks, with size s_b. For example, in Figure 3.3, the document with set index 0 and size s_{f0} may be subdivided in three blocks of size s_b, i.e. $s_{f0}/s_b = 3$ (each data block is identified with an index from 0 to 2).

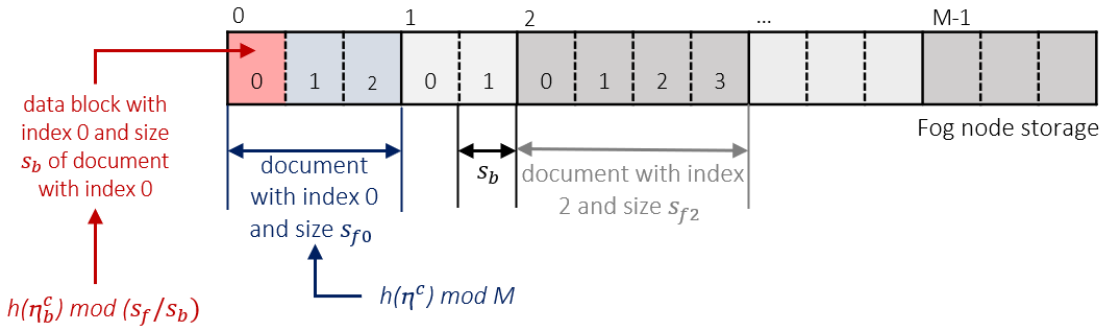

Figure 3.3: Fog node storage organization and hashes conversion to the first data block.

After reading the first data block ($block_1$), the untrusted part computes the cryptographic hash of the content, together with the set index hash ($h(\eta^c)$), and returns the result ($h(h(\eta^c), block_1)$) to the enclave. The enclave, in turn, computes a new hash with the response and the nonce η^c and forwards the result to the untrusted part, as this new result will identify the next file. Only this new hash ($h(h(h(\eta^c), block_1), \eta^c)$), enclave dependent, will determine the next file set index. To increase the data blocks index randomness, the enclave computes a second hash, with the hash of the retrieved data block, i.e., the enclave computes $h(h(h(\eta^c), block_1), \eta_b^c)$.

With the new hashes, the untrusted part repeats the module functions to obtain the next set index and the data block index (inside the identified file), reads the block and computes its cryptographic hash with the file set index, returning to the enclave. The enclave will compute two new hashes and so on, until the N data blocks are read. At the end, the enclave computes a final hash with the nonce η^c and sends the result back to the auditor, as the final proof. The auditor, after receiving the final hash, repeats all the above computations, to build a verifiable version, to check the proof correctness. If both computations match, the issued proof is correct. Figure 3.4 provides an overview of the execution steps of our designed challenge.

Security Guarantees: as each challenge requires two unique nonces and the sequence of N data blocks is pseudo-randomly chosen, the auditor ensures the fog node is unable to retrieve the N data blocks in parallel from neighbor nodes. Simultaneously, with the dependency between data blocks, as the next block index depends on the content of the previous one, the fog node can not reuse outputs from previous challenges. Finally, the constant exchange between execution environments (from enclave to untrusted part and vice versa) ensures the fog node can not resort to a remote machine (at the cloud or in the edge) to compute the whole proof. The proof is effectively computed at the expected machine [25].

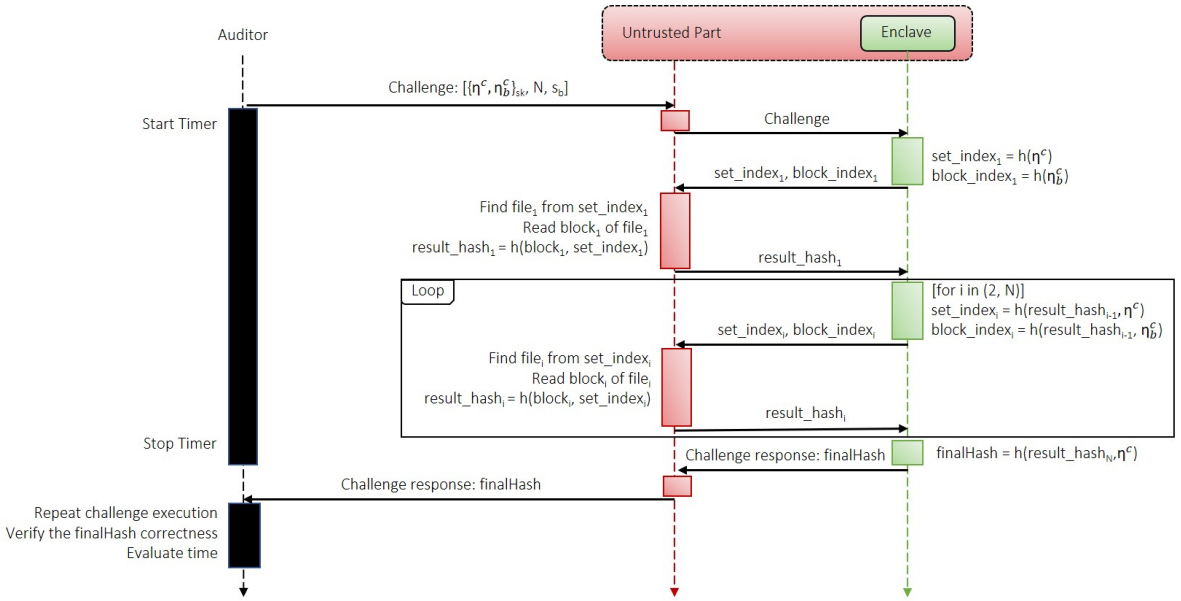

Figure 3.4: Execution of the challenge. set_index_i is the index of a given file in the metadata vector, and $block\_index_i$ is the index of a data block with size s_b inside the $file_i$. $block_i$ is the read data block. sk is the secret key agreed between the auditor and the enclave.

Note that with verification of proof correctness, the auditor assesses the storage documents integrity. Any modified accessed data file will result in an incorrect proof.

3.4.3 Estimating the Reading Delay δ at the Audited Node

Apart from checking the proof correctness, the auditor tests the proof timeliness. The auditor, when sending a challenge to the enclave, starts a timer that is stop when the proof is received. A proof is valid if it is correct and the reading delay estimate, for a single data block, is acceptable to the auditor.

Being T_i the time elapsed between the auditor sending the challenge i and receiving the response from the fog node, T_i may be decomposed in different factors, namely:

$$T_i = rtt_i + \alpha_i^1 + \ldots + \alpha_i^N + \delta_i^1 + \ldots + \delta_i^N \tag{3.1}$$

where rtt_i is the network round-trip time for the challenge-response messages, N the number of data blocks to be accessed, α_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 δ_i^j the delay observed to read a data block j. Note that for sufficiently large values of N, we will have:

$$\frac{\alpha_i^1 + \ldots + \alpha_i^N}{N} = \overline{\alpha}_i \approx \overline{\alpha} \tag{3.2}$$

$$\frac{\delta_i^1 + \ldots + \delta_i^N}{N} = \overline{\delta}_i \approx \overline{\delta} \tag{3.3}$$

However, the auditor is unable to measure accurate values for rtt_i, $\overline{\alpha}_i$ and $\overline{\delta}_i$, once they are different and variable at each challenge i and, at the same time, 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:

$$\overline{\delta}_{\text{estimate}} = \frac{T^i - \overline{rtt} - N\overline{\alpha}}{N} \tag{3.4}$$

Thus, the estimate error, for a given challenge i, will depend on how distant the observed values are to the mean values. Experimentally, we verified that the α values are subject to a negligibly small variance, whereby we assumed $\overline{\alpha}_i = \overline{\alpha}$, i.e, we discard the error introduced by α sampling. Hence, the error estimating $\overline{\delta}_{\text{estimate}}$ depends on two factors: i) the reading error ε^{δ} when estimating $\overline{\delta}$ (that depends on the sample size [2]) 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 3.5 describes:

$$\varepsilon^{\text{rtt}} = \frac{|rtt_i - \overline{rtt}|}{N} = \frac{\Delta^{rtt}}{N} \tag{3.5}$$

3.4.4 Configuring the Challenge

As described above, larger the number N of data blocks, lower the estimate error. However, there will always be an error, even if small, when estimating $\overline{\delta}$. Therefore, when configuring a challenge, the user must provide two parameters: the maximum error $\varepsilon^{\overline{\delta}}_{\text{max}}$ that he is willing to tolerate, when estimating $\overline{\delta}$, and, the challenge reliability ϕ, i.e., the probability to estimate $\overline{\delta}$ with an error lower than $\varepsilon^{\overline{\delta}}_{\text{max}}$.

With the maximum error $\varepsilon^{\overline{\delta}}_{\text{max}}$ and the reliability value ϕ, the auditor is able to determine the number N of data blocks, to ensure a low estimate error. At the same time, the auditor when determining N takes into account the network distribution between the auditor and the target node and the reading delay distribution of the target node. Given the reliability value ϕ and with the inverse cumulative distribution function (ICDF) [35] of the network distribution, we compute the worst case difference between the observed value rtt_i and the expected mean $\overline{rtt}$. With this difference, that we describe as Δ^{rtt}, we are able to determine a value N that places the estimate error of rtt_i above a given value $\varepsilon^{\text{rtt}}_{\text{max}}$.

On the other hand, from the reading delay distribution, at the fog node, and with the a learning curve, we are able to determine the maximum value $\varepsilon^{\delta_i}_{\text{max}}$ of the difference between the mean expected value $\overline{\delta}_i$, that comes from sampling measurements, and the real observed $\overline{\delta}$. Thus, the number N of data blocks may be chosen so that $\varepsilon^{\text{rtt}}_{\text{max}} + \varepsilon^{\delta_i}_{\text{max}} < \varepsilon^{\overline{\delta}}_{\text{max}}$.

[2] The sample size corresponds to the number of accessed data blocks.

Summary

This chapter addresses the design and implementation of our *Proof of Timely-Retrievability*, a proof that tests documents integrity and whether fog nodes are able to satisfy clients data requests within a threshold (usually small) δ. Our proof is issued by a fog node as a response to a challenge, previously sent by an auditor. To prevent the proof from being produced by another machine, the challenge execution depends on the *Intel SGX* extension present in fog nodes. The number of objects that need to be accessed by the auditor, in order to produce a proof, depends on the distribution of the time the audited node observes when accessing local data and on the variability of the network between the auditor and the audited node. In the next chapter, we evaluate the performance of our proposed Proof of Timely-Retrievability.

4

Evaluation

Contents

In this chapter, we describe the techniques and hardware used to evaluate our proof of timely-retrievability. With our experimental setup, that we describe in Section 4.1, we aim to simulate an honest and a dishonest edge storage provider. Remember that an honest provider keeps documents with an access time within δ (a threshold agreed in the SLA), while a dishonest one does not comply with the δ threshold. In Section 4.2 we describe the set of evaluation scenarios and in Section 4.3 the defined number N of data blocks for the required false positive and false negative rates. In Section 4.4 we discuss the obtained results. Finally, in Section 4.5 we discuss on how our challenge may be extended for different δ values.

4.1 Experimental Setup

In our experiments, we use three machines. One of the machines is used to execute the auditor. Another machine is used to execute the audited node. Finally, the remaining machine is used as a remote storage node, that is used by the audited node to access data in configurations where it does not store the files locally. In our experiments, the fog nodes store all files at the same location, i.e., all files are stored locally or all files are stored in the remote storage node.

The three machines are physically placed at our lab, and are connected via a switch, with a mean network delay, between any two machines, of $0.1ms$ (measured via client-server messages). Moreover, we extended our evaluation to emulate deployments where the different machines are placed at distinct geographic locations (i.e., with a network latency between any two machines larger than $0.1ms$). To emulate wide-area links we artificially add delays to the messages exchanged by any pair of nodes. These delays are drawn from a network latency distribution that has been observed experimentally with machines geographically apart. Figure 4.1 provides an overview of our system setup and the applied techniques for evaluation purpose.

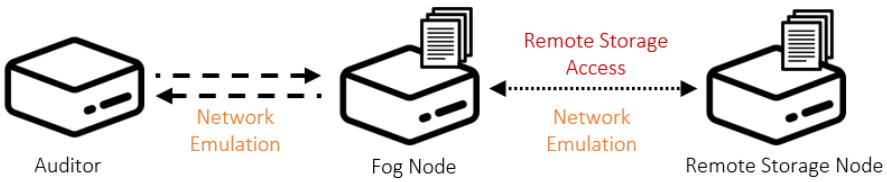

Figure 4.1: Experimental setup.

In the following sections, we describe in detail how the fog node has access to the remote storage, in Section 4.1.1, and which techniques we used for network emulation, in Section 4.1.2.

4.1.1 Access to Remote Storage

The fog node is set up to have access to two file systems: a local and a remote one, so that we can test our proof with both an honest and a dishonest provider. The remote file system is placed at the remote storage node. In our evaluation, we have experimented two different techniques for the fog node to access the remote file system.

4.1.1.A Secure Shell FileSystem

In one configuration, we set up the fog node to access the remote file system via *Secure Shell File System (SSHFS)* protocol. With SSHFS the fog node reads files from remote storage as if they were in local storage [36]. At the same time, we resorted to *symbolic links*, to allow code implementation to be storage location independent. A symbolic link is a pointer to a file and, thus, the content of the symbolic link is the path name of the file to which the link refers to [37]. Given the two file systems, the local and the remote one, the fog node has a set of symbolic links that point to the files in local storage, and has another set of symbolic links that point to the files in remote storage. According to the file system it aims to access, the fog node uses the respective set of symbolic links. Note that both the SSHFS configuration, and the symbolic links generation are executed before any audit.

One advantage of this configuration is that the audited node uses the same file system interface to access the files, regardless of their location. The different protocols used when accessing local or remote files are hidden by the Unix's virtual file system. A disadvantage of this approach is that the SSHFS protocol has a significant overhead. In particular, to access a remote file using the SSHFS protocol the audited node may need to perform several remote calls, which amplifies the effect of the network latency. This makes a dishonest node easier to detect, and may not represent the behavior of an ingenious adversary, that would attempt to hide the fact that it was not storing the files locally.

4.1.1.B Client-Server Plugin

Due to the limitations of the SSHFS-based configuration described above, we decided to implement an alternative technique for supporting remote file access by the audited node. This technique avoids the latency amplification phenomena. It works by placing a server on the remote storage machine that serves as an helper to the audited node to respond to the challenge (the audited node works as a client to this server). When the audited node is requested to compute the hash of a block it does not store locally, instead of attempting to read the block itself, it sends to the server all the information needed to compute the hash. More precisely, during the challenge, and for every iteration with the enclave, the fog node forwards the enclave request (the hashes that correspond to the file and the block indexes) to the remote storage node. The remote node computes the file cryptographic hash, and sends the result back

to the fog node, which forwards back to the enclave. In this approach, the only visible overhead is the network between the nodes, avoiding the extra complexity caused by the SSHFS protocol. This strategy requires a single round-trip in the network to execute a step of the challenge and avoids the transfer of the actual block in the network. As far as we can see, this is one of the most effective strategies for a dishonest node to hide the fact that it is not storing the data locally.

4.1.2 Network Emulation

In our experimental setting all machines are connected to the same local area network. To assess the performance of the system when the auditor is at different locations, and also when the audited node stores the data in remote nodes, we have resorted to network emulation.

For this purpose we have used the *ping* tool to collect the distribution of the observed round-trip time between two endpoints of the network we were trying to emulate. For our experiments we have collected these values for the network between the IST campus in Taguspark (Oeiras) and our lab at INESC-ID in Lisbon and between the Google datacenter in London and also our lab at INESC-ID in Lisbon.

For each of these networks we have collected enough samples to obtain a good approximation of the distribution of the round-trip latency. We collected 600 pings for both endpoints. Then, to emulate a network, we have artificially added a delay in the communication between the two endpoints. We have used two different techniques to introduce this delay, each with its own advantages and disadvantage:

- In the experiments with a *naive adversary*, that uses SSHFS to access remote files, we have used the emulator *NetEm* (*Network Emulator*) [38], available in Ubuntu operating system. This approach as the advantage of allowing to control *all* the communications between two nodes. Therefore, it allows to delay all calls performed by SSHFS without any additional instrumentation. In this case, the *NetEm* was configured to follow a normal distribution and to use the mean and standard deviation values from the samples that we have captured, as explained above. One limitation of using *NetEm* is that it may introduce some bias when approximating the real distribution, for instance the resulting average round-trip time can be larger than the target value [38, 39].

- In the experiments with an *ingenious adversary*, that uses the client-server plugin to request the remote computation of the block hash, we have used the following technique to add delays: every time a remote call was performed, we have instrumented the code to include a sleep time with a duration randomly selected from the samples collected from that network. Albeit simple, we observed that this method was able to offer a better approximation of the real network round-trip time distribution. We have used this technique for the client server plugin and also to the communication between the auditor and the audited node, given that, in these cases, it was easy to identify the points in the code where the instrumentation needed to be inserted.

	μ	σ
INESC-ID–Taguspark	$7.4\ ms$	$12.3\ ms$
INESC-ID–London	$34.5\ ms$	$1.7\ ms$

Table 4.1: Networks mean (μ) and standard deviation (σ) values measured with *ping*.

4.2 Evaluation Scenarios

We now introduce the different evaluation scenarios, where for each scenario, we interchanged the auditor and the remote node between three locations: Lisbon and Oeiras, in Portugal, and London, United Kingdom. We kept the fog node fixed in Lisbon. When two machines were placed in Lisbon, we have used the real network without any change. When two machines were placed in different locations, we emulated the wide-area link using the technique describe above.

Table 4.1 describes the mean (μ) and standard deviation (σ) values, for the two networks. The distribution of the round-trip time is depicted in Figure 4.2. Surprisingly, the network INESC-ID–Taguspark is subject to a larger variation, i.e., more unstable, than the network INESC-ID–London (INESC-ID–Taguspark network has a larger standard deviation value than INESC-ID–London network).

Table 4.2 shows the different scenarios we have used in the evaluation. The case where we do not assign a location to the remote storage node represents a scenario where the fog node keeps the set of documents in local storage and does not resort to a remote node.

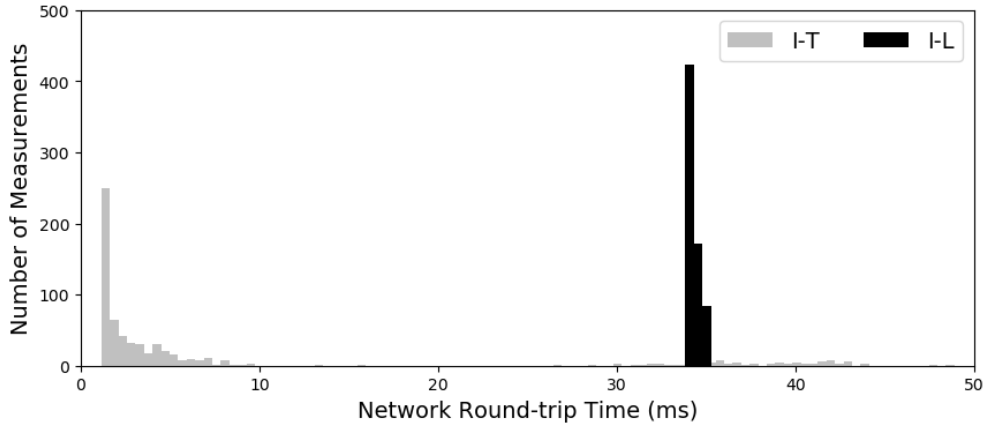

Figure 4.2: Network round-trip time, measured with *ping* tool for the networks INESC-ID–Taguspark (I-T) and INESC-ID–London (I-L)

	Scenarios											
	II	III	IIT	IIL	TI	TII	TIT	TIL	LI	LII	LIT	LIL
Auditor	INESC-ID	INESC-ID	INESC-ID	INESC-ID	Taguspark	Taguspark	Taguspark	Taguspark	London	London	London	London
Fog Node	INESC-ID	INESC-ID	INESC-ID	INESC-ID	INESC-ID	INESC-ID	INESC-ID	INESC-ID	INESC-ID	INESC-ID	INESC-ID	INESC-ID
Remote Node	-	INESC-ID	Taguspark	London	-	INESC-ID	Taguspark	London	-	INESC-ID	Taguspark	London

Table 4.2: Nodes location for evaluation scenarios.

It is important to highlight that it was only possible to simulate machines placement in Oeiras and

London, due to machines physical proximity. Both the *NetEm* and the sleep approach add extra delay on top of the already existing network delay (in this case $0.1ms$). Thus, if the real network delay between any two machines was larger than the values that we wanted to emulate, the used mechanisms for network emulation would be useless. At the same time, to reduce the real network influence at the emulated network, when configuring *NetEm* or the sleep instruction for Oeiras and London locations, we deducted the real delay of $0.1ms$ [39].

Finally, as the ping measurements describe the network round-trip time, in both networks auditor-fog node and fog node-remote node, we only configured the delay at one side of the connection.

4.3 Configuring the Challenge

When setting a challenge, the main factor to be taken into account is the number N of data blocks to be retrieved, as it affects the estimate error $\varepsilon^{\overline{\delta}}_{\text{max}}$ and the challenge reliability ϕ.

Remember from Section 3.4.4 that the number N of data blocks depends on two factors: (1) the network between the auditor and the fog node, and (2) the reading delay distribution. Hence, $\varepsilon^{\overline{\delta}}_{\text{max}}$ and ϕ depend on the above two factors. More precisely, more unstable is the network between the auditor and the fog node, and more variable is the estimated reading delay distribution, greater is the estimate error and lower the challenge reliability. Nevertheless, as the error that comes from the network between the auditor and the fog node is dispersed through the number N of data blocks, as shown in Equation 3.5, the variance of the reading delay distribution is, at the end, the main factor of error.

At the same time, as the auditor is aware of the reading delay distribution at a fog node that complies with δ, the auditor is able to detect an honest provider with higher accuracy than a dishonest provider. Note that a dishonest provider may have an infinite set of misbehaviors [1], which makes it more difficult to the auditor to detect it. As a consequence, our proof is able to guarantee a False Positive Rate (FPR) lower than a False Negative Rate (FNR).

The false positive rate describes the number of challenges that identify a fog node as dishonest, when the node is actually honest, out of the total number of executed challenges. In turn, the false negative rate describes the number of challenges where a node is signed as honest, but it is actually dishonest, out of the total number of challenges. For a matter of efficiency, we select a number N of data blocks that provides acceptable false positive and false negative rates.

In our proof, we want to meet a false positive rate of 0.01% and a false negative rate of 0.05%. Through experiments and error analysis, we noticed that with $N = 1000$, we can meet the desired rates. It is important to note that the above configuration of $N = 1000$ depends on the δ threshold, and on how distant the estimated access times are to δ.

[1] Since having the agreed files with different access times, i.e., some within δ and some beyond δ, to change frequently the files location and, thus, access time.

In Section 4.4, we describe in more detail the observed estimate error $\varepsilon^{\overline{\delta}}_{\text{max}}$ for $N = 1000$, as it is hardware dependent [2] and we used different machines in different moments of our experiments.

4.4 Results

As introduced in Section 4.1.2, we divided our experiments in two adversaries: a naive adversary and an ingenious adversary. In each one we resorted to different techniques for both remote storage access and network emulation. In the naive adversary, we combined the SSHFS protocol, to access the remote file system, with *NetEm*, for network emulation. In the ingenious adversary, we combined the client-server plugin with the sleep time approach.

For each adversary we used machines with different characteristics, due to availability of more advance machines in the middle of the evaluation process. As a consequence, each configuration will have a distinct estimate error $\varepsilon^{\overline{\delta}}_{\text{max}}$, as this error depends mainly on the reading delay distribution, that is hardware dependent.

For a better understanding, we divided this section in two parts, based on the two adversaries. In each part, we describe the used hardware and analyze the estimated reading delay and obtained estimate error. Nevertheless, some challenge configuration parameters are hardware independent or, even if hardware dependent, are the same in both setups and, thus, common between challenges. We now describe the common parameters and configurations:

In the first place, we opted for data blocks with 64K bytes of size, i.e., $s_b = 64KB$, as it is a multiple of $4KB$, the block size used by the file systems in both configurations and provides a low variance [4] ($1.0ms$ and $0.3ms$ variances for the first and second adversaries, respectively). Although, the block size used by the file system is hardware dependent, it is a common characteristic between the used machines.

Second, we filled the file system with a dataset of 520184 images [40], with sizes between $846B$ and $180KB$. Therefore, when the audited node has to read a file with a size smaller than $64KB$, the file data is padded with 0 until the data block has $64KB$. The index of the next file to be access is determined by the cryptographic hash of the $64KB$ data block, together with the hash with $32B$ from the previous iteration (Section 3.4.2). In our code implementation, we used *SHA256* function for the cryptographic hash and *Rijndael AES-GCM 128bits* algorithm to cipher the nonces.

To finish, for each adversary, we ran a set of challenges, each one configured with $N = 1000$, as described in Section 4.3. Simultaneously, we ran the challenges in the scenarios from Section 4.2, more precisely, in the scenarios where the auditor is placed in Taguspark, i.e., TI, TII, TIT and TIL. We opted for the scenarios where the auditor is in Taguspark, as it is the more unstable network and, thus, a more

[2] The estimate error depends on the reading delay distribution that is hardware depend. Two machines with different hardware characteristics will describe different reading delay distributions.

critical and challenging scenario for the auditor. To this end, we have emulated the INESC-ID–Taguspark network between the auditor and the fog node.

4.4.1 Naive Adversary

The naive adversary results as a combination of the SSHFS protocol, to access the remote file system, with *NetEm*, for network emulation. As introduced in Section 4.1.2, we configured *NetEm* to follow a normal distribution based on the network samples captured with *ping*. Thus, for the network INESC-ID–Taguspark, we set up *NetEm* to follow a normal distribution with $\mu = 8.0ms$ and $\sigma = 2.0ms$. Note that the mean and standard deviation values are not equal to the ones captured directly with *ping*, as we did an adjustment to the samples in order to follow a normal distribution.

Hardware Characteristics: For the naive adversary, we used a Intel NUC7i7DNKE machine, with a Intel i7-8650U CPU, that supports *SGX*, 8 GB RAM, 250GB M.2 SSD, and runs Ubuntu 20.04 LTS. Experimentally, we measured the delay this machine takes to compute the cryptographic hash ($\overline{\alpha}$) of a $64KB$ data block with a $32B$ hash, and we obtained $\overline{\alpha} \approx 0.8ms$. Remember from Equation 3.4 that the delay the audited node takes to compute a cryptographic hash is required to estimate $\overline{\delta}$.

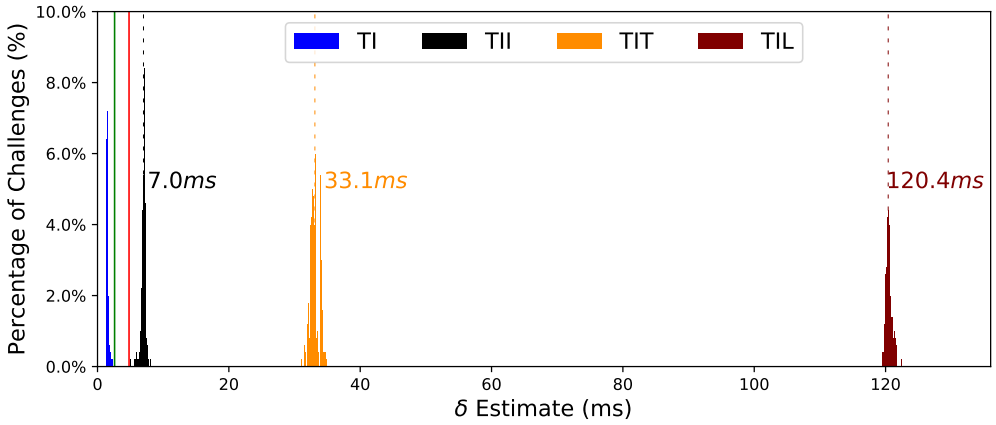

Figure 4.3: Estimate of the reading delay δ, in milliseconds, at the naive adversary, by an auditor emulated in Taguspark and the remote storage node emulated between INESC-ID, Taguspark and London. The green line is the detection threshold (DT), $DT \approx 2.6ms$, and the red line is the unacceptable latency threshold (ULT), $ULT \approx 4.8ms$.

Figure 4.3 shows the estimated reading delay δ verified at a naive adversary, with the auditor emulated to Taguspark and the remote storage node emulated between INESC-ID (TII), Taguspark (TIT) and London (TIL). The first scenario (TI), describes the case where the audited node keeps the whole dataset in local storage and does not resort to a remote storage node. In each scenario, we ran 500 challenges.

Note that for the scenarios where the audited node resorts to a remote storage node for storage and the remote node is emulated to Taguspark (TIT) or, London (TIL), the estimated access times are 4x the

mean of the emulated network. This overhead is due to SSHFS remote calls, as *open*, *lseek*, *read*, etc, that are also subject to the delay added by *NetEm*. The two first scenarios TI and TII are equally subject to the overhead introduced by SSHFS remote calls. However, in the scenario TI, the SSHFS overhead is less clear as the remote calls happen inside the same machine. Nevertheless, we show that with our challenge the auditor is able to estimate access times closer to the access time observed at the audited node.

Scenarios	Observed mean at the Audited Node ($\overline{\delta}_{\text{obs}}$)	Estimated mean by the Auditor ($\overline{\delta}_{\text{est}}$)	$\lvert\overline{\delta}_{\text{obs}} - \overline{\delta}_{\text{est}}\rvert$
TI	$1.6ms$	$1.5ms$	$0.1ms$
TII	$7.1ms$	$7.0ms$	$0.1ms$
TIT	$32.6ms$	$33.1ms$	$0.5ms$
TIL	$120.3ms$	$120.4ms$	$0.1ms$

Table 4.3: Observed mean access time at the naive adversary and the estimated mean access time by the auditor.

Table 4.3 describes the average values observed at the audited node and the average values estimated by the auditor, for the assessed scenarios. Notice that the auditor can estimate the observed reading delays with a maximum deviation of $0.5ms$ for the TIT scenario. The scenario TIT registers the larger difference between the observed and the estimated access times, as it is the scenario with the more unstable network between the audited node and the remote storage node. Remember from Section 4.2 that the network INESC-ID–Taguspark is more unstable than the network INESC-ID–London.

In this experiments, we considered a $\overline{\delta}$ value equivalent to a local storage access, i.e., $\overline{\delta} \approx 1.6ms$, as it is a critical scenario, due to TI and TII estimates proximity. The TI and TII scenarios have the lowest access time difference, between any two scenarios (a difference of $5.5ms$). Nevertheless, with $N = 1000$ and a FPR=0.01%, for the TI estimates, we can ensure an estimate error lower than $1.0ms$, i.e., $\varepsilon^{\overline{\delta}}_{\text{max}} < 1.0ms$.

With this being said, a fog node that complies with δ is correctly assigned as honest, if the estimated access times, by the auditor, are below $2.6ms$ ($\overline{\delta} + \varepsilon^{\overline{\delta}}_{\text{max}} \approx 1.6 + 1 \approx 2.6ms$). We describe this threshold as the Detection Threshold (DT). In other words, the detection threshold corresponds to the maximum access time acceptable to the auditor, that ensures the required false positive rate (in this case FPR=0.01%). Even though the scenarios TI and TII have the lowest estimate difference, the TII estimates are clearly above DT, which means that even with an difference of just $5.5ms$, the auditor is able to accurately distinguish a fog node that keeps files in local storage from a fog node that resorts to a remote storage node in the same local-area network. It is important to highlight that the detection threshold depends on the value δ agreed in the SLA, and the estimate error $\varepsilon^{\overline{\delta}}_{\text{max}}$ at an honest node.

In addition, as we defined the detection threshold, we defined an Unacceptable Latency Threshold (ULT). This threshold lower bounds the access time estimates for a false negative rate of 0.05%. As there are a set of possible scenarios, we defined the ULT based on the TII scenario, as it is the one closer to the above detection threshold. Given the difference between the average estimate and the

value that bounds 0.05% of TII estimates, we obtain the false negative margin. Thus, with $N = 1000$ and a false negatives margin of $2.2ms$, we define an ULT at $4.8ms$. Any fog node with access time estimates above $4.8ms$ is clearly assigned as dishonest (does not comply with the δ).

The scenarios were the audited node resorts to a remote storage node (TII, TIT and TIL), the estimated access times are above $4.8ms$, and, thus, above the unacceptable latency threshold, by what their misbehavior was clearly detected by the auditor. In a matter of fact, for the ran 500 challenges, we did not verify any false positives neither false negatives.

4.4.2 Ingenious Adversary

The ingenious adversary combines the client-server plugin, for remote storage access, with the sleep time approach for network emulation.

Hardware Characteristics: In the experiments for the ingenious adversary, we used a Intel NUC10i7FNB machine, with a Intel i7-10710U CPU, that supports *SGX*, 16GB RAM, 250GB M.2 SSD, and runs Ubuntu 20.04 LTS. The remote storage machine is similar to the adversary machine. Experimentally, once again, we measured the cryptographic hashing delay of a $64KB$ data block with a $32B$ hash, at the audited node, and we obtained $\overline{\alpha} \approx 0.1ms$.

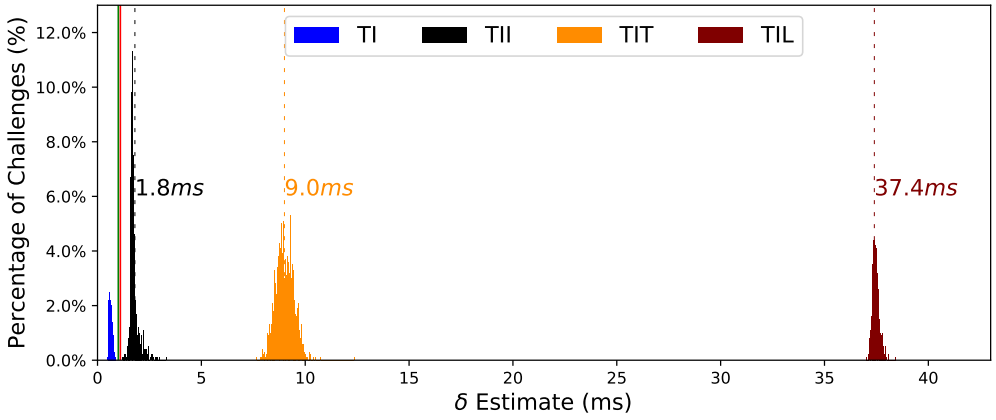

Figure 4.4: Estimate of the reading delay δ, in milliseconds, at the ingenious adversary, by an auditor emulated in Taguspark and the remote storage node emulated between INESC-ID, Taguspark and London. The green line is the detection threshold, $DT \approx 1.0ms$, and the red line is the unacceptable latency threshold, $ULT \approx 1.1ms$.

Figure 4.4 shows the estimated reading delay δ by the auditor, for the ingenious adversary. The auditor is emulated to Taguspark and the remote storage node is emulated between the considered three locations: INESC-ID (TII), Taguspark (TIT) and London (TIL). The auditor also estimated the reading delay for an audited node that does not resort to a remote storage node (scenario TI). For each scenario, we ran 1000 challenges.

Scenarios	Observed mean at the Audited Node ($\overline{\delta}_{\text{obs}}$)	Estimated mean by the Auditor ($\overline{\delta}_{\text{est}}$)	$\lvert\overline{\delta}_{\text{obs}} - \overline{\delta}_{\text{est}}\rvert$
TI	$0.5ms$	$0.6ms$	$0.1ms$
TII	$1.8ms$	$1.7ms$	$0.1ms$
TIT	$9.5ms$	$9.0ms$	$0.5ms$
TIL	$37.5ms$	$37.4ms$	$0.1ms$

Table 4.4: Observed mean access time at the ingenious adversary and the estimated mean access time by the auditor.

Table 4.4 describes the difference between the observe average values at the audited node and the estimated average values by the auditor, for the considered scenarios.

The estimated values in the scenarios with remote storage (TII, TIT, TIL) have no longer an extra overhead introduced by the remote storage access approach. Instead, the estimated access times are closer to the mean of the network between the audited node and the remote storage node. At the same time, the estimated values for the first scenario (TI) are lower than the estimated at the naive adversary, for the same scenario, as we used more advanced hardware.

Simultaneously, note that the variance between the estimated values for the TIT scenario is larger than the verified for the naive adversary. This is due to the configuration of the sleep time approach. As we select randomly a value from the captured samples, the sleep time, for a set of challenges, will follow the network variance, that is in this case $151.3ms$ ($\sigma^2 = 12.3ms$). In the naive adversary the variance is lower, as we set up *NetEm* with a standard deviation of $2.0ms$ and, thus, there is a variance of $4.0ms$.

In this experiments, we considered, $\overline{\delta}$ as the access time to access files in local storage, i.e., $\overline{\delta} \approx 0.5ms$. We select the observed mean value at the TI scenario, as it is once again a critical scenario for the auditor. The difference between the estimated values for the TI scenario and the estimated values for the TII scenario is just $1.3ms$ (lower than the same difference in the naive adversary). As a consequence, the auditor has a smaller gap to place the detection threshold, thus, it becomes more difficult to distinguish the TI and the TII scenarios.

For the local storage scenario (TI), with $N = 1000$ and for FPR=0.01%, there is an estimate error $\varepsilon^{\overline{\delta}}_{\text{max}} < 0.4ms$. Hence, we are able to define the detection threshold at $1.0ms$. Therefore, the auditor is able to detect the audited node misbehavior in scenario TII, as the estimated values are above the DT. In the ran 1000 challenges, we did not verify a false positive.

As for the false negatives, we define the unacceptable latency threshold at the $1.1ms$. Once again, we define this threshold based on the estimated values for the TII scenario, as it is the one closer to the DT. From the difference between the mean and the delay that ensures 0.05% estimates are above it, we obtained a false negative margin of $0.1ms$. Remember that the ULT lower bonds the estimates from misbehavior nodes, for a false negative rate of 0.05%. In the 1000 ingenious adversary experiments, we did not verify any false negative, as all estimates from misbehavior nodes are above $1.1ms$.

4.5 Discussion

In both naive and ingenious adversary, we define δ as the delay to access files in local storage, i.e., in memory space in the audited node machine. Thus, the detection threshold is placed next to the TI estimates, as for the given δ, we aim to distinguish a fog node that keeps files in local storage from one that resorts to a remote node. Moreover, the defined thresholds allow to detect a fog node that resorts to a remote node in the same local-area network (scenario TII). Nevertheless, both δ value and the detection threshold are not fixed values for all evaluation contexts. Given a set of requirements, the client may define a lower, or larger δ value than the one considered in our experiences. One example, would be to test whether the audited node keeps files stored in Portugal or not. For this example and taking into consideration Figure 4.4, the δ value may be defined as $12ms$ and, thus, the detection threshold could be place at about $15ms$. Note that we are able to determine some example values, as it is clear the difference between remote storage in Portugal (TIT scenario), and outside of Portugal (TIL scenario). We may even not observe any false positives, or false negatives, as the estimates are clearly distant to the detection threshold. In other words, even if the network between the fog node and the remote node is unstable and, thus, the estimate error is high, due to estimates distance, the auditor can detect storage in two distant locations with accuracy.

Equally, if any two distinct scenarios have estimates close to each other, it is more difficult for the auditor to place the detection threshold to ensure a given false positive rate. Thus, in this case, the auditor may increase the number N of data blocks to increase the challenge accuracy or, may accept a larger false positive rate. For a scenario where we we accept a latency equivalent to files storage in Portugal border with Spain and there is a remote storage in the border of Spain with Portugal, it may be hard for the auditor to distinguish if a the audited node keeps files in Portugal or resorts to a remote node, outside, but closer to the border. Yet, if the auditor increases the number N of data blocks, it may determine the two close locations with higher accuracy.

Summary

This chapter addresses the experimental setup, together with the techniques used for remote storage access and network emulation. We also described the evaluation scenarios and the considered real networks used for network emulation. We described on how to configure the challenge for the required false positive and false negative rates. We finished this chapter by providing the estimated δ values for the considered scenarios, and analyzing on how our challenge may be extended to other contexts (by changing the value of δ, and relocating the detection threshold).

5

Conclusion

Contents

5.1 Conclusions

The thesis describes a novel auditing mechanism that is able to extract a proof of timely-retrievability, i.e., a proof that a given fog node is able to serve requests without violating some given data access latency constraint δ. Our auditing mechanism is based on a challenge that requires the fog node to access, in sequence, a pseudo-random set of data items, and to respond to the challenge in a timely manner. The challenge is designed in a such a way that, if the fog node does not store locally a significant fraction of the objects, it will be unable to respond in time. To ensure that the challenge is executed by the fog node being audited, and not in some other, we leverage the trusted execution environments supported by modern hardware, namely on the *Intel Software Guard Extensions* (*SGX*). By using *SGX* we can also reduce the communication between the audited node and the auditor, by delegating to the enclave the task of revealing, iteratively, the next block that the audited node has to read.

We have implemented and evaluated experimentally our auditing mechanism. Our results show that our proof can accurately detect a node that is not able to satisfy the target latency constraint δ.

5.2 System Limitations and Future Work

Our work shows that is possible to configure the challenge to bound the probability of returning a false positive, i.e., the probability of erroneously mark a correct node as faulty. This happens because we assume that the auditor can model the operation of correct nodes (in particular, the distribution of access latency) and configure the number of samples accordingly. However, to bound false negatives is impossible, because faulty nodes can have an arbitrary behavior. It would be, however, interesting to identify a set of plausible misbehaviors, and to learn how to configure the challenge to reduce the number of false negatives in face of those attacks. For instance, a faulty node could store a fraction of the files locally, and a fraction of the files remotely. Such attacks may require the challenge to collect a larger set of samples to be detected accurately.

Also, it would be interesting to augment our system with a strategy to locate an enclave in the network, without relying on the attestation procedure to return the exact identity of the machine, where the enclave is running. Such strategy could use triangulation strategies such as the ones used for geo-localization (briefly described in Chapter 2). This would make the solution more general.

Bibliography

[1] J. Benet, “IPFS: Content addressed, versioned, P2P file system,” *arXiv preprint arXiv:1407.3561*, 2014.

[2] Swarm, “SWARM: Storage and communication for a sovereign digital society,” https://ethersphere.github.io/swarm-home/, 2019, [Accessed: 2020-12-12].

[3] 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.

[4] L. Li and L. Lazos, “Proofs of physical reliability for cloud storage systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 5, 2020.

[5] M. Gondree and Z. N. Peterson, “Geolocation of data in the cloud,” in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, San Antonio, Texas, USA, 2013.

[6] K. Benson, R. Dowsley, and H. Shacham, “Do you know where your cloud files are?” in *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, Chicago, Illinois, USA, 2011.

[7] Z. Ning, J. Liao, F. Zhang, and W. Shi, “Preliminary study of trusted execution environments on heterogeneous edge platforms,” in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018.

[8] C. Correia, M. Correia, and L. Rodrigues, “Omega: a secure event ordering service for the edge,” in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020.

[9] L. Ramaswamy and L. Liu, “Free riding: a new challenge to peer-to-peer file sharing systems,” in *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, 2003.

[10] M. Feldman and J. Chuang, “Overcoming free-riding behavior in peer-to-peer systems,” *SIGecom Exch.*, vol. 5, no. 4, 2005.

[11] N. Andrade, F. Brasileiro, W. Cirne, and M. Mowbray, “Discouraging free riding in a peer-to-peer cpu-sharing grid,” in *Proceedings of the 13th IEEE International Symposium on High performance Distributed Computing*, 2004.

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

[13] C. Streiffer, A. Srivastava, V. Orlikowski, Y. Velasco, V. Martin, N. Raval, A. Machanavajjhala, and L. Cox, “ePrivateEye: To the edge and beyond!” in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC)*, 2017.

[14] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, “Cachier: Edge-caching for recognition applications,” in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, Atlanta (GA), USA, 2017.

[15] M. Satyanarayanan, P. B. Gibbons, L. Mummert, P. Pillai, P. Simoens, and R. Sukthankar, “Cloudlet-based just-in-time indexing of IoT video,” in *Proceedings of the Global Internet of Things Summit (GIoTS)*, Geneva, Switzerland, 2017.

[16] G. Ricart, “A city edge cloud with its economic and technical considerations,” in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2017.

[17] N. Afonso, “Mechanisms for providing causal consistency on edge computing,” Master’s thesis, Instituto Superior Técnico, Universidade de Lisboa, 2018.

[18] J. Leitão, P. Costa, M. Gomes, and N. Preguiça, “Towards enabling novel edge-enabled applications,” *arXiv preprint arXiv:1805.06989*, 2018.

[19] C. Correia, “Omega: a secure event ordering service for the edge,” Master’s thesis, Instituto Superior Técnico, Universidade de Lisboa, 2019.

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

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

[22] M. Azraoui, K. Elkhiyaoui, R. Molva, and M. Önen, “StealthGuard: Proofs of retrievability with hidden watchdogs,” in *Computer Security - ESORICS 2014*, M. Kutyłowski and J. Vaidya, Eds., Cham, 2014.

[23] F. Armknecht, L. Barman, J. Bohli, and G. Karame, “Mirror: Enabling proofs of data replication and retrievability in the cloud,” in *Proceedings of the 25th USENIX Security Symposium*, 2016.

[24] C. Tan, M. Hijazi, Y. Lim, and A. Gani, “A survey on proof of retrievability for cloud data integrity and availability: Cloud storage state-of-the-art, issues, solutions and future trends,” *Journal of Network and Computer Applications*, vol. 110, 2018.

[25] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun, “ROTE: Rollback protection for trusted execution,” in *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC, 2017.

[26] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida, “Constraint-based geolocation of internet hosts,” *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, 2006.

[27] 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.

[28] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, “Innovative technology for cpu based attestation and sealing,” in *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, vol. 13, 2013.

[29] M. Barbosa, B. Portela, G. Scerri, and B. Warinschi, “Foundations of hardware-based attested computation and application to SGX,” in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, 2016.

[30] T. Kim, J. Park, J. Woo, S. Jeon, and J. Huh, “Shieldstore: Shielded in-memory key-value storage with SGX,” in *Proceedings of the Fourteenth EuroSys Conference 2019*, Dresden, Germany, 2019.

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

[32] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, Helsinki, Finland, 2012.

[33] D. Meyer, “What GDPR will mean for companies tracking location,” https://iapp.org/news/a/what-the-gdpr-will-mean-for-companies-tracking-location/, 2018, [Accessed: 2021-03-12].

[34] F. Anwar, L. Garcia, X. Han, and M. Srivastava, “Securing time in untrusted operating systems with TimeSeal,” in *2019 IEEE Real-Time Systems Symposium (RTSS)*, 2019.

[35] S. Ross, *Introduction to Probability and Statistics for Engineers and Scientists (Fourth Edition)*. Academic Press, 2009.

[36] L. Heddings, “How to mount a remote folder using SSH on ubuntu,” https://www.howtogeek.com/howto/ubuntu/how-to-mount-a-remote-folder-using-ssh-on-ubuntu/, 2016, [Accessed: 2021-03-11].

[37] M. Kerrisk, “symlink(7) — linux manual page,” https://man7.org/linux/man-pages/man7/symlink.7.html, 2021, [Accessed: 2021-09-30].

[38] S. Hemminger, “Network emulation with netem,” in *Australia’s National Linux Conference*, 2005.

[39] A. Jurgelionis, J. Laulajainen, M. Hirvonen, and A. Wang, “An empirical study of netem network emulation functionalities,” in *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, 2011.

[40] Nari, “tiles_256x49,” https://www.kaggle.com/narimatsu/tiles-256x49?select=0005f7aaab2800f6170c399693a96917_14.png, 2017, [Accessed: 2021-03-12].