Stockholms universitet

Cloud Data Storage - Review and Comparison of Cryptographic Methods

Emilia Dunfelt

Handledare: Hannes Salin
Examinator: Lars Arvestad
Inlämningsdatum: 2021-06-11

Kandidatuppsats i datalogi
Bachelor Thesis in Computer Science

Abstract

Cloud computing is a paradigm of increasing interest, solving many problems related to traditional computing technologies such as scalability, resource constraints, location independence, and disaster recovery. One of the most significant areas of cloud computing is cloud storage, allowing customers to outsource data storage to a public provider. However, the lack of control over outsourced data poses a significant challenge to organizations looking to adopt the new technologies. Using a so-called data integrity scheme, the integrity and correctness of remotely stored data can be ensured. In this survey, we identify the crucial properties required of data integrity schemes used in practical applications and conduct a comparative analysis of some of the recently proposed schemes in this category. Finally, a discussion on the current state of practical implementations offered by large cloud storage providers on the market today is given.

Sammanfattning

Molntjänster ger goda möjligheter att lösa praktiska problem ofta förknippde med traditionella metoder och tjänster, så som resursbegränsningar, platsberoende, datahantering och säkerhetskopiering. Ett av de största områdena inom vilket molnet används är datalagring, vilket möjliggör för användare och organisationer att lagra filer och data på servrar som hanteras av en extern molnleverantör. Detta innebär också minskad kontroll över informationen som lagras då tillgänglighet och datasäkerhet styrs av leverantören, vilket ofta innebär ett stort hinder för organisationer med intresse av molnlagring. Denna uppsats ger en översikt över kryptografiska protokoll för att verifiera dataintegritet i molnet. Vi identifierar de avgörande egenskaper som krävs för att effektivt kontrollera integritet av data, samt ger en jämförande analys av några av de senaste protokollen inom området. Slutligen disskuteras även de möjligehter till implementation av integritetsprotokoll som erbjuds av ledande molnleverantörer på marknaden idag.

Acknowledgements

I want to express many thanks my supervisor Hannes Salin for his patience and guidance in writing this thesis. I also want to thank my family for endless support along the way.

Table of contents

1 Introduction

Cloud computing is rapidly emerging as one of the most prominent paradigms in the area of distributed computing today. Providing clients with near-unlimited computational resources on a pay-per-use basis, cloud computing offers a solution to many of the limitations of traditional computing – location independence, rapid provisioning of processing power, memory, and storage, as well as outsourcing of measure and control services.

One area in which cloud computing proves particularly useful in many organizations is data storage. In contrast to on-premise storage, the cloud storage model offers an on-demand scalable solution that allows clients to access stored data from different locations and devices while also reducing the burden of maintenance and operation. Naturally, as the adoption of cloud technologies increases, so does the interest in the security of these solutions. Outsourcing data storage raises concerns on data privacy and integrity, as a client storing sensitive data generally cannot assume that a storage provider will not operate maliciously. A malicious or compromised server can modify or even delete stored data with a low probability of detection unless the stored data is frequently accessed. Cloud storage for data backups is, for this reason, particularly vulnerable to this kind of integrity attack. Although the client and service provider generally operate under a service level agreement, specifying the service's premises such as availability, recovery conditions, and security, this is often not enough to guarantee the integrity of the stored data.

Data integrity schemes allow a client to perform an audit on the cloud data, thereby verifying that some part of the data has not been altered and is retrievable. The naïve way of verifying data integrity stored in a remote cloud is to download all the data and carry out the verification steps locally using the client's computational resources. However, as one might expect, this soon becomes impractical. One of the main reasons for outsourcing storage to a cloud storage provider is to relieve the client of storing large amounts of data. Often we also assume that the client has limited computational power. In practice, it is often better to choose a data integrity scheme that allows the auditor to instead access only a small, randomly selected subset of the stored data in a way that also provides good integrity guarantees. To this end, there are several solutions presented in the literature. A Proof of Data Possession (PDP) protocol allows a client to audit a small, randomly chosen, subset of the stored data, often using precomputed authenticator values. This probabilistic method can be used to verify, with high probability, that the stored data is intact. Another similar option is to use a Proof of Retrievability (PoR) strategy that, although generally more computationally intensive, also guarantees that the audited file can be restored with great probability.

In this thesis, we concentrate on methods of verifying the integrity of outsourced data and survey some recent solutions in this area. We focus solely on PDP and PoR protocols, which have emerged as an area of interest in research on cloud storage correctness and integrity auditing.

Section 2 provides some of the essential mathematical and cryptographical context needed for analysis of the surveyed data integrity protocols. In Section 3 a background on the cloud computing model is presented, as well as an analysis of the risks involved in cloud storage and a presentation of the relevant properties of data integrity schemes. The primary analysis of state-of-the-art protocols is presented in Section 4; we provide a comparative analysis of the properties, building blocks, and general structure of each scheme. Finally, Section 5 briefly discusses the options available for practical implementation of the examined protocols.

2

Preliminaries

We here present the mathematical and cryptographic preliminaries necessary to analyze and compare data integrity schemes in Section 4. The reader is assumed to be familiar with the foundations of group theory and complexity theory.

2.1. Cryptographic primitives and data structures

This section aims to define the principal cryptographic primitives that are referenced frequently in the examined protocols. Proofs and detailed explanations are available in [13, 7, 20].

We first specify how our notions of security are based on a security parameter in the cryptographic primitives defined in this section and also in Section 3.

Definition 2.1. Let $k \in \mathbb{N}$ be a security parameter. A function $f : \mathbb{N} \to \mathbb{R}$ is said to be *negligible* if, for all positive integers c, we have that

$$f(k) < \frac{1}{k^c},$$

for sufficiently large k.

Furthermore, we say that a problem is hard if there exists no polynomial time algorithm for solving it. Let us now make a few definitions related to the authentication and verification of data, which will be used heavily throughout the rest of the thesis.

Definition 2.2. A cryptographic *hash function* is a function $H : \{0,1\}^* \to \{0,1\}^k$, mapping an arbitrarily sized input document to a bit string of fixed size, generally called the *hash value*, or *hash digest*.

We require the following properties of a cryptographic hash function:

i) Pre-image resistance: Given the value $H(D)$ it is hard to compute D.

ii) Weak pre-image resistance: Given D it is hard to find D' such that $H(D) = H(D')$. In this case, H is also said to be *collision resistant*.

In order to reduce communications overhead in data integrity auditing we are often interested in the ability to combine several hash values into a single value for verification. For this reason we also make the following defintion:

Definition 2.3. A collision resistant hash function $H : \mathbb{F}^m \to G_p$, from a finite field $\mathbb{F}^m$ to a multiplicative group G_p of prime order p, is *homomorphic* if the equality

$$H(\alpha u + \beta v) = H(u)^\alpha \cdot H(v)^\beta$$

holds for every $u, v \in \mathbb{F}^m$ and $\alpha, \beta \in G_p$.

Collision resistant hash functions can be used to verify the integrity of data by checking that a newly computed hash value matches a previously stored hash. Instead of hashing an entire document, we might be interested in hashing blocks of a document to save our computational resources when verifying hash values independently. However, we must then store each file block's hash in memory, thereby consuming a large amount of read-only memory instead. One possible solution to this problem is by using Merkle hash trees.

Definition 2.4. Let H be a collision resistant hash function and $D = \{d_1, d_2, \dots, d_n\}$ a file divided into n blocks. A *Merkle tree* is a perfect binary tree, in which the label of each leaf-node is a data block in D. The intermediary nodes of the tree are labeled with the hash of its child nodes. The hash value of the root is called the *root-digest*.

Using a Merkle hash tree, given a hash $H(D)$ of the data D we can verify that a file block $d_i \in D$ is intact in its correct position by verifying the sibling nodes of the nodes in the path from the leaf d_i to the root. This is exactly the information required to verify a match with the root digest $H(D)$, given d_i. An example of such a proof is shown in Figure 1.

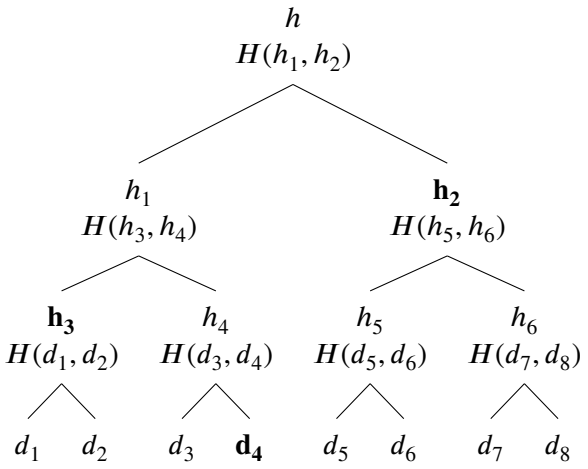

Figure 1: Merkle hash tree with data $\{d_1, d_2, \dots, d_6\}$, to verify authenticity of d_3 we check the values d_4, h_3, and h_2.

Hash functions are also commonly used to sign data in a digital signature scheme, which we define as follows:

Definition 2.5. A *digital signature scheme* consists of a private key K^{Pri}, a public key K^{Pub}, and two algorithms:

`Sign`(D, K^{Pri}): A signing algorithm that takes a document D and the private key as input, and returns a signature D^{Sig} of the document.

`Verify`(D, D^{Sig}, K^{Pub}): A verification algorithm taking a document, its signature, and the public key, and outputs whether the signature is valid or not.

A digital signature scheme has the following properties:

i) The verification algorithm `Verify` always accepts a signature generated by an honest signer.

ii) A signature scheme is secure if an adversary that has observed a polynomial number of signatures generated by `Sign` on chosen documents cannot produce a valid signature on some new document, except with negligible probability.

One frequently used signature scheme in the area of cloud data integrity auditing is the *Boneh-Lynn-Shacham (BLS) signature*. This signature scheme relies on bilinear map computations, defined in Section 2.3.

Definition 2.6 (BLS signature scheme)**.** Let $\mathbb{G}$ and $\mathbb{G}_T$ be groups of prime order p, where g is a generator of $\mathbb{G}$, and $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ a bilinear pairing. A *BLS signature scheme* is a digital signature scheme in which $K^{Pri} = x$, where x is a random element in $\mathbb{Z}_p$, the public key is $K^{Pub} = g^x$, and

`Sign`(D, K^{Pri}): A signing algorithm taking the document D to its signature $D^{Sig} = H(D)^x$, where H is a so-called *full-domain hash function* – an RSA-based hash function provably secure in the random oracle model (Section 2.2) [25].

`Verify`(D, D^{Sig}, K^{Pub}): Given the pair (D, D^{Sig}) the verifier checks the verification equation $e(D^{Sig}, g) = e(H(D), x)$. If this equality holds, the algorithm returns "success", and "failure" otherwise.

Yet another way to verify the integrity and authenticity of data are message authentication codes (MACs), which in contrast to digital signatures are a symmetric construction in which only a private key is used for signing and verifying. We make the following definition:

Definition 2.7. Let K^{Pri} be a randomly chosen private key. A *message authentication scheme (MAC scheme)* consists of two algorithms:

`Sign`(D, K^{Pri}): A signing algorithm taking the randomly chosen private key and the document, generating the *tag* D^{Tag}, often referred to as the message authentication code (MAC).

`Verify`(D, D^{Tag}, K^{Pri}): A verification algorithm taking the original document, its tag, and the private key. Outputs "success" if the tag is valid, and "failure" otherwise.

Valid tags generated by `Sign` should always be accepted by `Verify`.

A MAC scheme is said to be secure if it is secure against *existential forgery* under a *chosen message attack*, except with negligible probability. This means that an adversary who can freely obtain tags for chosen documents (chosen message attack), say from an unknowing signer using their private key, should not be able to forge a message-tag pair, different from the previously seen pairs (existential forgery), that is accepted by `Verify`.

We now define another cryptographic primitive commonly used in combination with hash functions, and MAC schemes:

Definition 2.8. Let $\mathcal{K}$ be a set of encryption keys, $\mathcal{M}$ a space of possible messages, and $\mathcal{C}$ the space of possible ciphertexts. A *pseudorandom function (PRF)* is an efficiently computable deterministic algorithm, F, taking a randomly chosen key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$ to a value $c \in \mathcal{C}$, such that the output of F behaves like a random function from $\mathcal{M}$ to $\mathcal{C}$.

A PRF is said to be secure if it is indistinguishable from a randomly chosen function from the family of functions $f : \mathcal{M} \to \mathcal{C}$, denoted by $Funs[\mathcal{M}, \mathcal{C}]$. Formally, this means that for a challenger choosing either the function F or a random $f \in Funs[\mathcal{M}, \mathcal{C}]$, the probability that an adversary given the output of a sequence of queries to the challenger can distinguish between either choice is negligible.

Lastly, we cover the theory of *zero-knowledge proofs*. A zero-knowledge proof is a cryptographic protocol taking place between two parties: a prover P and a verifier V. The ultimate goal of P is to convince V of the possession of some knowledge through a series of challenges without revealing any information that could also be used to convince an outside party. Let y be a quantity possessing the property $\mathcal{P}$. A zero-knowledge proof between P and V should then satisfy the following properties:

i) **Completeness**: If the quantity y has property $\mathcal{P}$, then V should accept the proof from an honest prover P.

ii) **Soundness**: If y does not have property $\mathcal{P}$, then V should only accept a proof from a cheating prover P with negligible probability.

iii) **Zero-knowledge**: A valid proof from P should not, except with negligible probability, convey any information to V, except the knowledge that y has property $\mathcal{P}$.

The zero-knowledge property of a zero-knowledge protocol is generally proven by the construction of a *simulator algorithm* by which the verifier simulate responses from P such that the simulations are indistinguishable from genuine responses. Thus, V gains no more information through a series of interactions with P than could be obtained using a simulator.

2.2. Cryptographic models and hardness assumptions

Cryptographic schemes are often proven secure under general assumptions about the context in which they operate. Such assumptions fall into two categories - cryptographic models and hardness assumptions. Cryptographic models concern the abilities of the parties involved in the protocol, such as their abilities to perform certain computations or utilize resources. For instance, a model can grant the participants unrestricted access to a cryptographic primitive, after which a theoretical statement such as security is proven. Although not proven secure in the so-called *standard model*, i.e. by making no model assumptions at all, such security proofs provide us with a heuristic indicating which part of the protocol is the weakest link in a practical implementation [12].

One of the more commonly used models is the random oracle model under which the parties of a cryptographic protocol can efficiently evaluate a random hash function by querying an oracle, called the random oracle. We give here the definition as in [5].

Definition 2.9. A *random oracle* is a map $R : \{0,1\}^* \to \{0,1\}^\infty$ chosen by selecting each bit of $R(x)$ uniformly and independently, for every x. In particular, the function $H : \{0,1\}^* \to \{0,1\}^k$ is called a random hash function.

The theory of hardness assumptions instead concerns the computational problems underlying a cryptographic protocol. Under the assumption that a given problem cannot be solved in polynomial time, it is possible to show that an algorithm is secure, assuming that an adversary does not have unlimited computational resources.

We define here the hardness assumptions most frequently relied upon in protocols examined in this thesis. We again rely on [13] in this section.

Definition 2.10. Given a prime p, a generator g of the cyclic group $\mathbb{Z}_p$, and two values $g^a \pmod p$ and $g^b \pmod p$, the *Computational Diffie-Hellman Problem* (CDH), or simply the Diffie-Hellman Problem, is the problem of computing the value of $g^{ab} \pmod p$, without knowledge of the exponents a and b.

Definition 2.11. Given a prime p, a generator g of $\mathbb{Z}_p$, and an element $h \in \mathbb{Z}_p$, the *discrete logarithm problem (DLP)* is to determine the integer x such that

$$g^x \equiv h \pmod p.$$

The final hardness assumption covered here is related to the famous RSA cryptosystem, which we now briefly explain: Bob wishes to send a message to Alice over an insecure channel. Using the RSA cryptosystem, Alice would then choose two large primes p and q, and an encryption exponent e such that $\gcd(e, (p-1)(q-1)) = 1$. The pair $(N = pq, e)$ then constitute Alice's public key, which she openly sends to Bob.

Bob can then choose a plaintext message m, and compute the ciphertext $c \equiv m^e \pmod N$ before sending this value to Alice. Finally, Alice decrypts this message using her secret exponent - the modular inverse $e^{-1} = d$, which she can easily compute using her knowledge of the secret primes p and q. She retrieves the plaintext by computing $c^d \equiv m \pmod N$.

We now describe the so-called RSA assumption, a hardness assumption based on the RSA cryptosystem.

Definition 2.12. Let $N = pq$ be a product of two primes and e an integer such that (N, e) corresponds to the public key in an instance of the RSA cryptosystem. The *RSA problem* is the problem of determining m in the equation $m^e \equiv c \pmod N$ without knowledge of the factors of N.

2.3. Exponentiation, bilinear pairings, and computational cost

In this thesis, we will compare cryptographic schemes based on elliptic curve- and finite field cryptography. To analyze the computational cost of the involved algorithms, we focus on the number of costly operations involved in the process. In this setting, the most expensive operations are exponentiations in a multiplicative group and computations of a bilinear map between two elliptic points. Arithmetic operations such as multiplication and addition are generally less costly than exponentiations and pairing computations, and we will not consider those in our analysis.

We now give a brief overview of the theory of elliptic curves and bilinear pairings, roughly following the presentation given in [7] and Chapter 6 of [13].

Definition 2.13. Let p be a prime. An *elliptic curve* E over the finite field $\mathbb{F}_p$ is an equation of the form

$$E : Y^2 = X^3 + AX + B,$$

such that $A, B \in \mathbb{F}_p$ and $4A^3 + 27B^2 \neq 0$. The set of points, $E(\mathbb{F}_p)$, on E together with the point at infinity, denoted by $\mathcal{O}$, is a finite abelian group under the addition of points on the curve.

Addition of the points P and Q on the elliptic curve E can be interpreted geometrically as in Figure 2. That is, the sum $P + Q = R$ on E is given by drawing the line through P and Q, finding the third point of intersection with E and reflecting it across the x-axis.

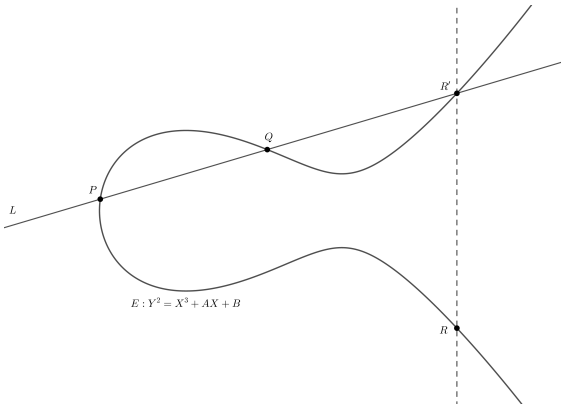

Figure 2: Addition operation on the elliptic curve E.

Operation	Time
Addition in $\mathbb{G}_0$	0.003347ms
Addition in $\mathbb{G}_1$	0.010751ms
Addition in $\mathbb{G}_T$	0.00037318ms
Exponentiation in $\mathbb{G}_T$	2.662ms
Pairing	3.913ms

Table 1: Benchmark comparison using the MCL library on the BN254 curve.

Definition 2.14. Let $\mathbb{G}_0, \mathbb{G}_1$, and $\mathbb{G}_T$ be cyclic groups of prime order q, where g_0 and g_1 are generators of $\mathbb{G}_0$ and $\mathbb{G}_1$ respectively. A *bilinear pairing* is an efficiently computable map $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ with the following properties:

i) bilinearity: for all $u, u' \in \mathbb{G}_0$ and $v, v' \in \mathbb{G}_1$ it holds that

$$e(u \cdot u', v) = e(u, v) \cdot e(u', v) \text{ and } e(u, v \cdot v') = e(u, v) \cdot e(u, v'),$$

and;

ii) non-degeneracy: $e(g_0, g_1)$ is a generator of $\mathbb{G}_T$.

As mentioned previously, we often consider pairings over elliptic curve groups. In this setting we would typically define a pairing over the groups

- $\mathbb{G}_0 \leq E(\mathbb{F}_p)$, a subgroup of order q;
- $\mathbb{G}_1 \leq E(\mathbb{F}_{p^d})$, a subgroup of order q for some $d > 0$, such that $\mathbb{G}_0 \cap \mathbb{G}_1 = \{\mathcal{O}\}$; and
- $\mathbb{G}_T \leq \mathbb{F}_{p^d}$, a multiplicative subgroup.

Explicit constructions of bilinear pairings over elliptic curve groups include the *Weil pairings* and *Tate pairings*, which are computed using Miller's algorithm. The Tate pairing is more commonly used in practice as it is computationally more efficient than the Weil pairing. For further discussion on this deep topic, see [7].

A pairing computation has asymptotic complexity $\mathcal{O}(\log q)$, but is in practical implementations much slower than exponentiation over the same groups. On elliptic curves, the "exponentiation" P^n is really an addition nP since the group operation is addition. The value of such an operation is computed using the double-and-add algorithm, similar to the fast powering algorithm for computing exponents in integer fields. Table 2.3 gives a benchmark comparison for elliptic curve pairings computed using the MCL library on a 1.2 GHz ARM Cortex-A53 on the BN254 curve [18].

3

The cloud storage model

In the following section, we present the fundamental definitions of cloud computing, its characteristics, and service- and deployment models. We then identify the primary security and privacy risks associated with large-scale cloud storage and discuss ways to mitigate and possibly eliminate such risks. To this end, we present several properties of interest in data storage schemes under the assumption that the server is malicious or compromised.

The *National Institute of Standards and Technology* (NIST) gives the following definition of cloud computing:

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. [17]

NIST further identifies the five essential characteristics of cloud computing: on-demand self-service, broad network access, resource pooling, scalability, and measured service. These five properties together capture the advantage of cloud computing in large organizations. Cloud computing offers a solution to the diverse computing needs of the client, such as file and data storage, backup services, application development, and big data.

Providers of cloud computing often offer their services according to the following service models as defined by NIST.

- **Software as a Service (SaaS)**: The provider offers applications running on its cloud infrastructure, accessible via a web- or program interface. Often these services follow a pay-per-use payment model. Some examples include Google G Suite and Microsoft Office 365.
- **Platform as a Service (PaaS)**: In contrast to SaaS, Platform as a Service allows the client to develop and deploy applications onto the cloud infrastructure owned by the provider.
- **Infrastructure as a Service (IaaS)**: This kind of service allows the client to utilize the computational resources such as data storage and processing power of the provider. A few of the most widely used IaaS providers are Amazon Web Services, Microsoft Azure, and Google Cloud Infrastructure.

Following the NIST definition, we also recognize four deployment models of cloud computing:

- **Private cloud**: Cloud infrastructure provisioned exclusively for a single organization. Operated and hosted either on-premise by the organization itself or externally by a provider.
- **Community cloud**: Provisioned for a community of consumers from different organizations with shared interests and concerns such as jurisdiction, security, and policy. The infrastructure is possibly operated by one of the community organizations or by a provider.
- **Public cloud**: A cloud infrastructure offered to the general public over the internet. Operated and hosted by a private provider, a government organization, or an academic institution.
- **Hybrid cloud**: In the hybrid cloud setting, two or more of the other infrastructures are combined but remain separate entities bound together to satisfy the varying computing needs of the client.

Cloud storage falls into the IaaS service model, and in this thesis, we primarily focus on the private deployment model. In the context of cloud storage, we identify the following three primary actors between which security protocols will be taking place: the *client* as the party interested in outsourcing their data storage, the *cloud storage provider (CSP)* to whom the data will be outsourced, and a trusted *third party auditor (TPA)* responsible for auditing the integrity of the data when outsourced to avoid a heavy computational burden on the client. The computational resources of the client is generally assumed to be limited, and the CSP is assumed to be untrustworthy as will be described in Section 3.1.

3.1. Risks and challenges associated with cloud storage

We here discuss the risks involved when outsourcing data storage to the cloud under the assumption that the CSP is malicious. Risks associated with cloud storage are mostly related to the fact that the client no longer is in possession of the outsourced data and often does not have access to the underlying cloud infrastructure and its components. Transparency and trust are therefore of significant importance when evaluating cloud storage solutions by different vendors.

Data availability is one risk in regards to cloud storage. A sudden outage at the CSP level can leave the client unable to access crucial data. For this reason, cloud storage providers and their customers operate under a Service Level Agreement (SLA), agreeing to a specified uptime percentage during a billing cycle and otherwise often providing an agreed-upon amount of financial compensation. For example, Google Cloud Storage provides a level of 99.95% availability [10], and the SLA of Amazon Simple Storage Service (Amazon S3) specifies an uptime of 99.9% with no compensation [1].

The second area of concern is that of data privacy. When outsourcing confidential data, a good option might be to employ client-side encryption to prevent unauthorized access and leakage of data to the CSP. However, this might be computationally expensive, and thus cloud providers can offer server-side encryption and protection against unlawful physical access [11].

Finally, we consider the integrity of outsourced data. When accessing stored data, it is of utmost importance that it has not unexpectedly been modified, corrupted, or even deleted. A malicious provider might attempt to delete outsourced data that is rarely accessed by the client in order to save storage space. Fundamentally, it lies in the interest of the CSP to provide a secure and stable service to customers. On the other hand, the damage to a provider's reputation caused by a catastrophic breach in these areas could be enough to motivate an attempt to evade disclosure of such incidents. Thus, we cannot assume a CSP to strictly act in good faith, and instead we want to guarantee data correctness under the assumption of a malicious cloud provider.

By utilizing so-called data integrity schemes and allowing clients to conduct secure and regular audits, the CSP can guarantee data integrity. In Section 3.2 we cover the critical properties of such schemes before comparing the state-of-the-art protocols in Section 4.

3.2. Properties of data integrity schemes

We identify several properties of probabilistic data integrity schemes that together describe the capabilities and limitations of a proposed scheme. In this thesis, we make the following broader categorization of data integrity schemes:

- **Provable Data Possession (PDP)**: Provides data integrity auditing, often based on precomputed authenticator values. An overview of recently proposed PDP schemes is given in Section 4.2.
- **Proof of Retrievability (PoR)**: Schemes providing data integrity auditing as well as data recovery guarantees, often based on codings (Section 4.3). An analysis of PoR schemes is presented in Section 4.4.

Note that any probabilistic data integrity scheme can trivially be transformed into a deterministic scheme by making a query on all the stored data at once. Therefore, we will not consider any deterministic protocols and will only analyze the efficiency and security of probabilistic protocols for use in the cloud. Following the taxonomy presented in [26], within these categories, we furthermore distinguish the following attributes:

- **Unbounded queries**: Schemes with this property put no restrictions on the number of times an auditor can perform the verification procedure. Typically, an audit would be performed regularly to increase audit quality and avoid data corruption or data loss. Hence, the property of allowing unbounded queries to the CSP is important to consider and also balance against the cost of frequent computation. Indeed, most recently proposed integrity schemes do possess this property [26]. Unless otherwise specified, the data integrity schemes described in our analysis will allow unbounded queries.
- **Soundness**: Often, it is essential that the verification scheme remains secure under the assumption that the CSP is malicious or compromised, as described in Section 3.1. In this scenario, it is essential that a malicious server cannot successfully pass an audit without indeed storing the data intact, unless with negligible probability. This property is referred to as soundness, and we only examine schemes with this property.
- **Statelessness**: If there is no need to store and maintain the state of any party in the cloud storage model between audits, a data integrity scheme is said to be stateless. In such schemes, individual audits are independent, and no resources are used to preserve the results of previous audits.
- **Dynamic data handling**: We consider the data stored by the server to be either static or dynamic. As the name suggests, static data is data that is not expected to be modified, such as backups. Similarly, dynamic data is expected to be modified in some way, for example, via insertion or deletion. Although most schemes only support static data, there are a few examples currently that support dynamic data handling, such as [24] or, more recently, [2].
- **Public auditing**: In many scenarios, it is preferable to outsource not only the storage of data but also the auditing process itself to a third-party auditor (TPA), as described previously. Furthermore, we would like the publicly audited data to remain secure and private against the TPA during an audit. Schemes with this property are called *privacy-preserving*. In contrast to public auditing is private auditing, which describes schemes in which the data owner cannot delegate the auditing to a third party.
- **Batch auditing**: A TPA might receive multiple audit requests from different users simultaneously. In such cases, rather than processing each request separately after one another, it might be possible to increase efficiency by auditing a batch of requests simultaneously by aggregating authenticator values. This is referred to as batch auditing.

4 Analysis of data integrity schemes

In this section, we examine Provable Data Possession (PDP) schemes and Proof of Retrievability (PoR) schemes that have the potential to achieve high guarantees while also being efficient enough for practical implementations.

All examined schemes are probabilistic but can trivially be transformed into their deterministic counterparts by making a query on the entire data set instead of a random subset. We first give a formal definition of PDP and PoR schemes and identify the algorithms involved, after which a comparative analysis of the state-of-the-art PDP schemes is presented, including a comparison of computational cost. Lastly, we examine the cryptographic primitives used to provide redundancy in PoR schemes and give a brief overview of schemes relevant for implementation in the cloud. In Table 2 we list common parameters used throughout this section.

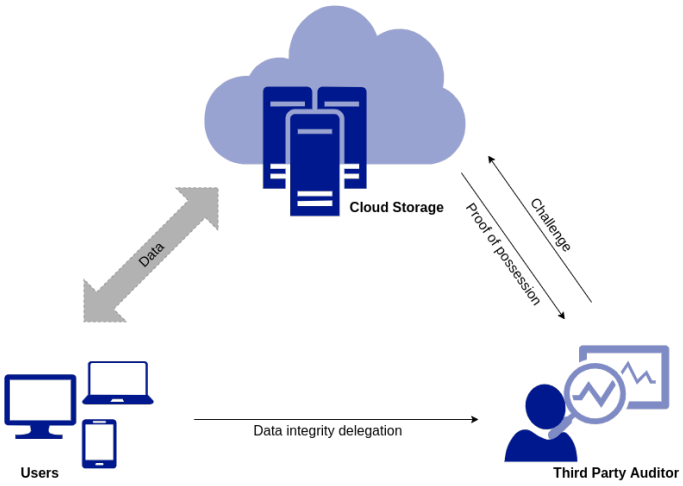

Figure 3: Model of a data integrity scheme.

Parameter	Description
F	The file being stored
n	Number of file blocks
s	Number of sectors in each file block
t	Number of randomly sampled blocks during an audit
k	System security parameter
$[E]$	Computation cost of an exponentiation
$[B]$	Computation cost of a bilinear pairing

Table 2: Parameters used in integrity schemes.

4.1. General structure of integrity schemes

To allow users to remotely verify the integrity of data stored in the cloud, without needing to download and process the stored information, two main methods have been proposed in recent years. Provable data possession schemes (PDP), first introduced in [3], give an effective and flexible alternative to previous deterministic methods. A PDP scheme can only be used to verify data integrity, and is limited by the fact that if a breach in data integrity is discovered, there is no way for the client to be able to restore the lost information. To address this problem, so-called Proofs of Retrievability (PoR) was introduced in [15] around the same time. Similar to PDP, a PoR allows the client to verify data integrity, and it furthermore utilizes more advanced cryptographic primitives to provide retrievability guarantees. The main drawback to these schemes as opposed to PDP is that the additional cryptographic tools used impose a layer of complexity in computation, ultimately making PoR schemes less practical.

Following the original definition of PDP in [3] we now give a formal definition of PDP schemes. The framework of data integrity protocols adhering to this model in depicted in Figure 3.

Definition 4.1. Let k be a security parameter, and $F = (m_1, m_2, \dots, m_n)$ a file divided into n blocks. A *Provable Data Possession* (PDP) scheme consists of four polynomial time algorithms:

$\texttt{KeyGen}(k)$:	a key generation algorithm which, given the system security parameter generates the public-private keypair $\kappa = (pk, sk)$.
$\texttt{TagGen}(\kappa, F)$:	a tag generation algorithm taking as input the keypair generated by `Keygen`, the file F, and returning metadata $M_\eta = (\Sigma, \eta)$ used in the verification, where $\Sigma = \{\sigma_i\}_{1 \leq i \leq n}$ is a set containing the metadata for each file block, encoded using an identifier η of the file F.
$\texttt{ProofGen}(pk, chal, F, \Sigma)$:	takes the public key, the file F and its metadata along with a challenge, $chal$, indicating which file blocks to audit. It returns a proof of data possession $\mathcal{V}$.
$\texttt{ProofCheck}(\kappa, chal, \mathcal{V})$:	validates the proof of possession $\mathcal{V}$ using the public-private keypair and the challenge. Outputs "success" if the file F is intact, and "failure" otherwise.

A PDP scheme together with two additional polynomial time algorithms `Setup`, and `Challenge` constitutes a PDP protocol. During the `Setup` phase, the client generates the keys using `KeyGen`, as well as the verification metadata using `TagGen`. During the `Audit` phase the client then proposes a challenge to the CSP generated by the algorithm `Challenge`. From the challenge, a proof of verification is generated by the CSP using `ProofGen`. Lastly, the validity of the proof is confirmed by the client using `ProofCheck`.

Finally, we give a formal definition of PoR schemes based on [21, 15], which in general follow the same structure as PDP, but also provide retrievability guarantees.

Definition 4.2. Let k be a security parameter and $F = (m_1, m_2, \dots, m_n)$ a file divided into n blocks. A *Proof of Retrievability scheme* is a Proof of Data Possession scheme along with an additional polynomial time algorithm `Extract` taking as input the key sk, a file identifier η, and a series of challenge-response interactions between the verifier and prover in the PoR, and outputs the file F, except with negligible probability.

By encoding the data in the `TagGen` algorithm using the coding techniques described in Section 4.3, such as erasure codes, lost or corrupted data can be restored by the extraction algorithm when interacting with an honest prover.

4.2. Comparative analysis of PDP schemes

This section serves as an overview of some approaches providing provable data possession in outsourced cloud storage. The main results of the analysis are presented in Tables 3 and 4. We here follow the notation described in Table 2.

A summary of the computational complexity of the examined schemes is shown in Table 4. In this analysis, only the most expensive operations are taken into account, that being exponentiation in multiplicative cyclic groups of prime order, denoted by $[E]$, and computations of bilinear pairing function, denoted by $[B]$.

Scheme	SL	DD	PA	PP	BA	Hardness assumption	Security model
[3]	✓	–	–*	–	–	RSA, KEA1-*r*	random oracle
[27]	✓	✓	✓	✓	–	CDH	random oracle
[24]	✓	✓	✓	✓†	✓	CDH	random oracle
[19]	✓	–‡	✓	✓	✓	RSA	random oracle

Table 3: Comparison of properties of PDP schemes. Here SL denote statelessness, DD support for dynamic data operations, PA public auditing support, PP privacy-preserving, and BA support for batch auditing.

4.2.1. Ateniese et al. (2007). Provable data possession was introduced first in [3], and allowed a client storing data at an untrusted CSP to verify the integrity of the data using probabilistic sampling of data blocks. Of the two schemes suggested in the article, the first one, called S-PDP provides the best detection guarantees. The detection guarantee P_X is bounded by

$$1 - \left(\frac{n-m}{n}\right)^t \leq P_X \leq 1 - \left(\frac{n-t+1-m}{n-t+1}\right)^t,$$

where m is the number of blocks modified by the server. In a file consisting of 10,000 blocks, this indicates at least a detection rate of 99% when sampling 4.6% of the total number of file blocks, assuming 1% of file blocks have been disrupted. Their second scheme, E-PDP, is more efficient at the cost of significantly weaker guarantees as it only verifies the possession of the sum of the sampled blocks instead of each individual block. Hence we only consider S-PDP and its modifications. Their RSA-based scheme uses so-called homomorphic verification tags (HVTs) generated for each file block and stored together with the file at the server.

For a file block m_i, its tag (T_{i,m_i}, W_i) consists of: *i)* a random value W_i obtained by concatenating the index i with a random, secret value, and *ii)* a value T_{i,m_i} computed on the block m_i itself and

*With modifications, however data is exposed to TPA under the suggested modification of the S-PDP audit protocol.

†With modification. This also increases the cost for TPA and CSP during the audit phase.

‡Extension by the use of Merkle hash tables, index hash tables, index tables, doubly linked information tables, or index switchers are suggested by the authors but no concrete modifications are further proposed. Authors suggest this is something to be added in their future work.

Scheme	KeyGen	TagGen	ProofGen	ProofCheck
[3]	$[E]$	$2n[E]$	$(t+3)[E]$	$(t+2)[E]$
[27]	$2[E]$	$(2n+s)[E]$	$(t+s+1)[E]+[B]$	$(t+s)[E]+3[B]$
[24]	$[E]$	$2n[E]$	$(t+1)[E]+[B]$	$(t+3)[E]+2[B]$
[24], BA	$[E]$	$2n[E]$	$m((t+1)[E]+[B])$	$m(t+3)[E]+(m+1)[B]$
[19]	$(s+1)[E]$	$(1+(2+s)n)[E]$	$(t+s+2)[E]$	$(2t+s+3)[E]$
[19], BA	$(s+1)[E]$	$(1+(2+s)n)[E]$	$((2t+2)m+s+2)[E]$	$((2t+s+1)m+2)[E]$

Table 4: Complexity analysis of suggested PDP schemes. Schemes marked with BA are modified for batch auditing.

the public key using BLS signatures. The tags allows a user to verify the possession of the corresponding file blocks without having to be in possession of the file itself, and their homomorphic property ensures that multiple tags can be combined into a single value. This makes it possible to combine the HVTs for the t sampled file blocks into a single value that can be used to verify authenticity.

Even so, the HVTs in S-PDP impose a quite significant impairment to the flexibility of the scheme by being usable only for static data, since modification of any block of data requires the recomputation of tags.

S-PDP achieves knowledge soundness under the RSA and KEA1-r assumption in the random oracle model. The KEA1-r is defined in [3] as follows:

Definition 4.3. Let N be a product of two safe primes $p = 2p' + 1$ and $q = 2q' + 1$, g a generator of the cyclic subgroup of $\mathbb{Z}_N^*$ of order $p'q'$, and s an integer. The assumption that for any adversary $\mathcal{A}$ taking as input (N, g, g^s) and returning (C, Y) such that $Y = C^s$, there exists an extractor algorithm which, given the same input as $\mathcal{A}$, returns x such that $C = g^x$, is called the *Knowledge of Exponent Assumption (KEA1-r)*.

A modification to S-PDP is provided that does not rely on the KEA1-r assumption, although it results in a significant increase in network communication costs. This modification can also be further extended to support public auditing. However, the modification exposes a linear combination of sampled file blocks to the TPA, thus potentially exposing user data after running several audits since the auditor can determine file blocks by solving a system of linear equations. So this modification is not privacy-preserving.

S-PDP has been shown to be insecure due to lack of randomness in the `Challenge` algorithm, which allows the CSP to store results of previous audits and cheat in the future [26].

4.2.2. Zhu, Hu, Ahn, and Yau (2012). After the PDP technique was introduced, many improvements and modifications to the original protocol where quickly introduced. In 2012, Zhu et al. [27] proposed an audit scheme based on bilinear pairings supporting dynamic data operations as well as privacy preserving public auditing. The detection rate of their scheme is similar to that of [3]. The scheme builds a hash index table using a BLS-based hash function for which the file tag metadata is computed. By concatenating the sequence number of each block with its version number and a random integer in order to avoid collision, it is possible to construct the hash table

Scheme	Client	TPA	CSP
[3]	$(t + 2n + 3)[E]$	–	$(t + 3)[E]$
[27]	$(2n + s + 2)[E]$	$(t + s)[E] + 3[B]$	$(t + s + 1)[E] + [B]$
[24]	$(2n + 1)[E]$	$(t + 3)[E] + 2[B]$	$(t + 1)[E] + [B]$
[19]	$(2n + s(1 + n) + 2)[E]$	$(2t + s + 3)[E]$	$(t + s + 2)[E]$

Table 5: Comparison of total computations for parties in the cloud storage model.

in a way that supports dynamic data operations. By the zero knowledge property, the scheme remains privacy-preserving during public audits. Furthermore, it has the soundness property under the computational Diffie-Hellman assumption in the random oracle model.

4.2.3. Wang, Chow, Wang, Ren, and Lou (2013). Another privacy preserving PDP scheme was proposed by Wang et al. [24], based on bilinear pairings and similar to [3], homomorphic verification tags that are further integrated with random masking at the server to prevent leakage of information to TPA. Their scheme provide the same detection rate as [3]. By the use of Merkle hash trees their scheme can be extended to support dynamic data operations by additionally sending the tree root to TPA as additional metadata in the setup phase. This does not incur any additional computations of exponentiations or bilinear pairings, and the scheme is privacy-preserving by the random masking.

Their scheme satisfy the soundness property under the computational Diffie-Hellman assumption in bilinear groups in the random oracle model, and is privacy preserving against TPA in the random oracle model.

However, their scheme is not secure against leakage to TPA if the stored data has low entropy, as it is possible to extract information by guessing if a particular message is stored on the server and running the verification using randomly generated coefficients in the verification tags. However, the authors note that for this process to be successful it requires TPA to correctly choose the coefficients from $\mathbb{Z}_p$ where p is assumed to be a very large prime, and it is thus highly improbable. To this end they provide a modification to the proposed scheme which also has the zero-knowledge property. Although the modification provides additional security, it also adds $2[E] + [B]$ in computational cost for CSP in `ProofGen`, as well as an additional bilinear map evaluation to the computational cost of TPA during `ProofCheck`.

A further modification of their scheme provides support for batch auditing allowing the TPA to significantly reduce the computational cost of the verification process requiring only $(m + 1)$ pairing operations instead of $2m$ operations as required by m individual audits. The computational benefits of batch auditing are shown in Table 4.

4.2.4. Ni, Zhang, Yu, and Yang (2020). Recently, Ni et al. [19] proposed an RSA-based scheme, called ID-P^{3}DP, using identity-based public keys, thus providing an alternative to certificate-based public key authentication. In contrast to certificate-based authentication in which user's private keys are issued by a trusted *certificate authority*, the private keys in identity-based authentication are based on personal information of each user, and is valid during a fixed time period. In our analysis we consider only the computations made during one such time interval, and note that a private key exponent would need to be computed for every time block during which the ID is valid.

Currently their scheme does not support dynamic data operations, although several possible extensions are mentioned, for example by the use of Merkle hash trees. The detection probability is also similar to that of [3]. The ID-P^{3}DP scheme is privacy-preserving using a zero-knowledge proof protocol in the random oracle model, and has the soundness property under the RSA assumption. Batch auditing is also supported, the computational benefits of which are presented in Table 4.

4.3. Methods of retrievability

Assuming data has been tampered with by the server, a PoR scheme provides retrievability guarantees based on the metadata generated by the `TagGen` algorithm. While PDP schemes examined in this survey use tags for verification, PoR schemes also use additional methods such as codes and signatures [26]. This section presents a few of the basic techniques used in the encoding step of many proofs of retrievability.

Efficiency demands that our verification schemes are probabilistic, and therefore an audit process might not detect a small number of erasures done by the server. To reduce the risk of losing sensitive data we are interested in being able to fully restore the stored file assuming that `ProofCheck` returns "success" on a random sample of t blocks.

Error-correcting codes allows both error-detection as well as error-correction. Although the theory of error-correcting codes is beyond the scope of this thesis, we here give an example of how such a coding technique might work, before discussing how similar techniques are used in integrity schemes. A further introduction to this theory is given in [14].

Say we wish to transmit a message $M \in \{0, 1\}$ over an unstable channel where messages are at risk of being altered before reaching the destination. The simplest way to ensure that the message can be correctly interpreted by the receiver in this setting is to encode M as a *codeword* of length n obtained from M by repeating the message n times. So, if $M = 0$ and $n = 3$, the message is encoded and transmitted as 000. If at most one error is made during transmission the receiver might obtain any of $\{000, 001, 010, 100\}$, only one of which is valid under this code. It is therefore in this way possible to both detect and correct one error. This code is called the *binary repetition code* of length n, and can correct at most $\left\lfloor \frac{1}{2}(n-1) \right\rfloor$ errors. In general, we can increase the redundancy in the encoding to be able to correct a greater number of errors. However, increasing the redundancy in the error correcting code would also increase the cost of transmission and storage.

In the context of cloud storage, data might not only be altered but also deleted. For this purpose *erasure codes* are used, as described in [21]. Let F be a file divided into n blocks, an erasure code is an error-correcting code that encodes F in into a (n/r)-block file in such a way that any sample of n blocks suffices to restore the original file. The parameter r is called the *rate* of the code, and corresponds to the fraction of encoded blocks that are useful for decoding. Codes in which r is significantly less than n are called *rateless*, meaning that F can be encoded into a codeword of arbitrarily many blocks, of which n still suffices for decoding.

One particularly commonly used code is the *Reed-Solomon code*, which can correct up to $\left\lfloor \frac{1}{2}(n-k) \right\rfloor$ errors, where k is the number of codewords. It can also be used as an erasure code with arbitrary rate, for which encoding and decoding runs in $\mathcal{O}(n^2)$ time [21]. Thus, by encoding the stored file using a Reed-Solomon erasure code of rate t we can be sure to be able to restore the full file if the CSP passes an audit with a random sample of t file blocks.

Aside from erasure codes such as the Reed-Solomon code, there is a wide range of codes and techniques used to guarantee retrievability in PoR. Other suggested codes are for example network

codes, online codes, and locally decodable codes [21, 16], and even algebraic techniques as in [2]. Comparing available codes in terms of efficiency and security is here omitted, but a short analysis in the PoR context is given in [23].

4.4. Comparative analysis of PoR schemes

In this section we analyze some PoR schemes suggested in recent years, starting with the original proposed by Juels and Kaliski in [15]. Table 6 gives an overview over the properties of the proposed schemes. In the analysis, we will primarily compare the properties defined in Section 3.2 and the type of encoding used to ensure retrievability.

For each paper analyzed, we discuss the main scheme presented and only briefly mention any modifications where relevant.

4.4.1. Juels and Kaliski (2007). In their original scheme, Juels and Kaliski [15], proposed a private proof of retrievability for static files, called SENTINEL-PORSYS. Their scheme uses a *sentinel*-based approach to guarantee authenticity; by randomly embedding a fixed set of check-blocks called sentinels among the file blocks in an encoded and encrypted file, data corruption can be discovered with high probability by querying for the sentinel blocks. Encryption ensures that the CSP cannot determine which blocks are among the embedded sentinels, and encoding ensures that the file can be retrieved. Note however that this method does not allow an unlimited number of audit requests as it relies on the sentinel locations being unknown to the CSP, meaning that the data needs to pass through the `Setup` phase after all sentinels have been queried. The basic variant of their scheme uses an error-correcting Reed-Solomon code, with codewords of length l, and which is capable of correcting up to d errors.

The detection rate, P_X, of their scheme is bounded below in the following way

$$1 - \left(1 - \frac{\epsilon}{4}\right)^t \leq P_X,$$

where ϵ is the upper bound on the fraction of file blocks (including sentinels) that have been corrupted by the server. Denote the number of embedded sentinels by q, and let

$$\mu = l\epsilon \cdot \frac{n+q}{n-\epsilon(n+q)}.$$

The probability, P_Y, that a file is retrievable is then bounded below as follows

$$\frac{n}{l} \cdot e^{d-\mu} \left(\frac{d}{\mu}\right)^{-d} \leq P_Y.$$

Say for example we apply SENTINEL-PORSYS to a file with 2^{27} blocks, i.e. a 2GB file, with a block size of 128 bits. Each chunk of 223 file blocks are encoded using a Reed-Solomon code with codewords of length $l = 255$, that is capable of correcting $d = 16$ errors. This results in an expansion of about 14% to the original file size, and the number of blocks under this encoding is $b = 153,477,671$. By embedding $s = 1,000,000$ sentinels, the total expansion of the file then becomes 15%.

Assuming the adversary has corrupted 0.5% of the data, so that $\epsilon = 0.005$, a query on 1,000 sentinels has a detection rate of 71.4% per challenge. Under such circumstances, the probability

Scheme	SL	DD	PA	PP	Retrievability method	Hardness assumption
[15]	–	–	–	–	ECC	–
[21], PRIV	✓	–	–	–	EC	×
[21], PUB[§]	✓	–	✓	–	EC	CDH
[8]	✓	✓	–	–	EC	×
[22]	–	✓	✓[¶]	–	EC	–
[20]	–	✓	✓	–	EC	DLP
[2]	–	✓	✓	✓	Verifiable computing	DLP, LIP

Table 6: Comparison of properties of PoR schemes. Here, SL denote statelessness, DD support for dynamic data operations, PA public auditing support, PP privacy-preserving, and the symbol × indicate that the security relies on properties of the chosen cryptographic primitives.

that the stored file cannot be retrieved is bounded below by approximately $4.68 \cdot 10^{-6}$, which is less than 1 in 200,000.

The scheme in its original form does not allow for public auditing nor dynamic data operations. Several modifications are also presented, mainly focused on bandwidth optimizations, and ways to allow for unbounded audits by exchanging the sentinels for a MAC-based approach.

4.4.2. Shacham and Waters (2008). Shacham and Waters [21] built upon the ideas presented in [15] and gave two static proofs of retrievability, one private and the other one public, called PRIV and PUB respectively. Their work took inspiration from both the PoR of Juels and Kaliski [15], but also of the PDP presented by Ateniese et al. [3]. Similar to [3], their schemes utilize homomorphic authenticators to combine the tags of each queried file block into an aggregated authentication value, thus reducing the response length. Both schemes rely on erasure codes derived from Reed-Solomon codes, providing retrievability guarantees under the assumption of an erasing adversary.

In PRIV, the file is first encoded, then split into blocks on which the authentication values are computed using a combination of a pseudorandom function and a symmetric encryption scheme. A file tag used for authentication is also generated using a MAC scheme. The private keys can then be used to verify authenticity of the response from the CSP. This scheme is secure assuming the MAC scheme is unforgeable, the encryption scheme is semantically secure, and finally assuming that the pseudorandom function is cryptographically secure.

The publicly verifiable scheme instead uses BLS signatures to compute the authenticators for each file block, which can then be verified using only a public key, thus allowing for public auditing via a TPA. This also means that PUB relies on more expensive operations, such as bilinear pairings and exponentiations in symmetric groups. A summary of the cost of computations is as follows:

- `KeyGen`: $[E]$,
- `TagGen`: $n(s+1)[E]$,
- `ProofGen`: $t[E]$, and
- `ProofCheck`: $(t+s)[E] + 2[B]$.

The security of this scheme relies on the CDH assumption, in the random oracle model.

[§]The security of the scheme is proven in the random oracle model.
[¶]With modifications.

4.4.3. Cash, Küpçü, and Wichs (2013). Another approach to the PoR construction is given by Cash et al. in [8], which allows for dynamic data operations while also hiding the client's data and access-patterns from the server.

This is achieved by use of an *oblivious RAM model*. This model was originally introduced in [9], as a continuation of the work on so-called oblivious machines. Computations with the same running time on an oblivious machine accesses the same memory locations. So an oblivious Turing machine is one in which the movement of the tape heads look identical for every computation, and an oblivious RAM is a machine on which the probability distribution of the sequence in which memory addresses are accessed only depends on the length of the input. The oblivious RAM is realized by use of a hierarchical data structure consisting of several layers of hash tables. The higher tables contain the most recently accessed data, and the lower tables contain the less frequently accessed data and has more capacity. The access pattern of a read operation on the stored data is masked by checking each table, starting from the top, until the requested block is found, after which random memory locations in the remaining tables are accessed. The accessed data is then written to the top table. Occasionally, the structure is "rebuilt" to prevent tables from overflowing.

By applying the erasure code only to a small selection of data blocks, instead of the entire file at once, the need to update the entire encoding on every dynamic update is removed. However, this opens up for the possibility of the server to delete or ignore the updated blocks in a given codeword while remaining undetected since an audit only checks a small subset of the file blocks. To hide which blocks has been modified by the client, Cash et al. proposes the use of an oblivious RAM on the server in their scheme called PORAM. The main drawback of this method is that the oblivious RAM needs to be initiated and maintained, and that the use of this data structure generally impose a greater bandwidth overhead compared to other methods [22].

Their scheme also assumes the use of an erasure code such as the Reed-Solomon code, and their oblivious RAM relies on a pseudorandom function and a symmetric encryption scheme. The PORAM is secure under the assumption that the pseudorandom function is secure and that the encryption is secure against chosen-plaintext attacks.

4.4.4. Shi, Stefanov, and Papamanthou (2013). Similar to Cash et al., Shi et al. realized the inherent difficulty in allowing efficient dynamic operations in a PoR scheme [22]. Their dynamic scheme is more efficient than the one proposed in [8], and also allows for public auditing by using a clever construction based on Merkle hash trees. In this setting, public auditing refers to the ability of a public entity to read and verify the authenticity of stored data, while only a trusted entity is allowed to write data to the server.

To solve the issue of dynamic updates, three separate buffers are used on the server-side: the *raw buffer* U, the *erasure-coded buffer* C, and the *hierarchical log* H. The first contains the up-to-date blocks in their un-encoded format on which read and write operations can be performed in the usual way. This is implemented using a Merkle hash tree, thereby allowing audits to be performed on this data structure, as well as public read operations. In C is stored the erasure-coded copy of the data, which only gets rebuilt based on the updated data in U after n write operations. Finally, the H buffer stores a hierarchical log of the recent write operations, so that on every write operation in U the erasure coded written block along with its identifier is added to the top level of H. The log contains $k + 1$ levels, where $k = \lfloor \log n \rfloor$, and each level i contains 2^i erasure coded blocks. When attempting to add a new value to the top of the log the first i levels might be filled already, thus prompting the *rebuilding* of level $i + 1$ at which point the first i levels are emptied, encoded,

and subsequently placed in level $i + 1$. This is similar to the rebuilding of the oblivious RAM , but much faster since it only requires the computation of an erasure code of the new level. An audit is done by querying for random blocks of the encoded buffer and the hierarchical log.

The drawback of this method as compared to the PORAM of [8] that it does not hide the access patterns of the write operations, although this is not a strictly necessary feature of PoR schemes.

Finally, public auditing can be achieved by also building Merkle trees over the hierarchical log H and the encoded buffer C, although it imposes a significant increase in write operations. The details of this modification is here omitted but can be found in Appendix 1 of [22].

4.4.5. Sengupta and Ruj (2020). A recent take on the dynamic PoR is given by Sengupta and Ruj [20]. Their approach is based on that of Shi et al. [22] and uses the three buffers U, C, and H, but also builds two additional data structures $\tilde{H}$ and $\tilde{C}$ which are used to store the homomorphic authentication tags of the blocks stored in H and C respectively. This approach significantly reduces the computational complexity of performing public write operations as compared to [22]. The trade-off between additional storage and less bandwidth overhead is motivated by the fact that the latter is generally more expensive.

The computation of the homomorphic tags require the computation of $n \cdot s$ modular exponentiations, where n denotes the number of file blocks and s the number of block sectors. By again making a trade-off between storage and expensive computations this can be improved by generating a public key which requires s exponentiations in total, after which the tag computation for the file only requires n exponentiations. Assuming this improvement has been made, the algorithm `ProofGen` then requires the computation of t exponentiations, and `ProofCheck` requires $3t$ exponentiations in total.

Furthermore, their scheme uses a digital signature scheme to sign the authentication tags. Assuming the use of a state-of-the-art signature scheme based on elliptical curves, this signing would impose a pairing computation on each tag, and thus an increase of the total cost of computation. Finally, the scheme is secure under the discrete logarithm assumption, and assuming that the signature scheme used is provably secure.

4.4.6. Anthoine et al. (2021). The final scheme considered in this analysis is a recent one by Anthoine et al. [2], that does not rely on any encoding scheme but instead uses a so-called verifiable computing scheme based on matrix algebra. A modification also allows privacy-preserving public auditing. Their scheme is not yet published in a peer-reviewed journal.

In contrast to the other methods examined, their scheme does not require any additional metadata to be generated, thus significantly reducing the persistent storage required. Instead, the file F is organized in a square matrix $M \in \mathbb{R}_q^{s \times t}$, where the number of bits of the file is equal to $st \cdot \log_2 q$, and $\mathbb{R}_q$ is a finite ring of order q. In the privately verifiable scheme, the verification metadata consists of two private control vectors: one randomly chosen vector $u \in \mathbb{R}_q^s$, and one vector $v \in \mathbb{R}_q^t$ computed as

$$v^T = u^T M.$$

An audit request is then performed by choosing a random challenge vector $x \in \mathbb{R}_q^t$ which is sent to the server. In `ProofGen` the server then computes the vector $y \in \mathbb{R}_q^s$ as

$$y = Mx.$$

During `ProofCheck` the client can verify that

$$u^T y = v^T x.$$

Several successful audits ensures that the full matrix M can be restored with great probability by solving for M in the matrix equation $MX = Y$, where X is the matrix of challenge vectors, and Y the matrix of response vectors. Thus, no coding is needed to ensure retrievability. This approach is also independent of computational hardness assumptions. A Merkle hash tree is used to ensure authenticity during read and write operations. The server stores the full tree, while the client is only required to store the root hash.

The publicly verifiable modification is similar to the private scheme, but its security relies on the DLP assumption, and is privacy-preserving under the Linearly Independent Polynomial assumption (LIP). The LIP assumption states that for linearly independent multivariate polynomials $P_1, ..., P_s$ of bounded degree and a secret m in a group G of prime order with generator g, the tuple $(g^{P_1(m)}, ..., g^{P_s(m)})$ is indistinguishable from a random tuple of the same size.

5 Conclusion

Although cloud storage offers a practical solution to many of the problems of traditional storage, such as maintenance and operation costs, location- and device dependence, scaling, and disaster prevention, new threats to customer data are introduced in the form of privacy and security concerns. We have in this thesis presented and analyzed a few data integrity protocols suggested in recent years, which show that although there are many options available, the client often needs to make a tradeoff between the desired properties, storage- and computational costs, as well as time- and space complexity. In this final section, we present a brief overview of the current state of the practical data integrity implementations available, and make a few notes with regards to the future of cloud storage auditing.

5.1. Practical implementations

Examining the offerings of leading cloud providers on the market today, such as Amazon Web Services (AWS), Google Cloud for government, and Microsoft Azure, it does not appear that any dedicated solutions for data integrity auditing are implemented as part of any existing framework today. Hence, a client that requires correctness and integrity guarantees must be able to implement such features independently, which requires a deeper understanding of cloud security and the implementation of protocols such as those described in Section 4. For example, several of the PoR protocols analyzed expect the user to choose which cryptographic primitives provide a suitable level of security and flexibility, thereby raising the bar for the level of cryptographic knowledge necessary to implement them.

However, there is an interest from industry leaders such as AWS to provide secure and trustworthy solutions for customers operating under strict regulations, such as actors in the public sector. In October 2020, AWS announced the launch of the AWS ClearStart program, aimed at organizations in the Swedish public sector [4]. The program consists of two main stages:

- **The AWS ClearStart Delivery Framework**: A framework developed for members of the AWS Partner Network (APN) to follow to deliver and operate services in compliance with Swedish legislation and security regulations.
- **The AWS ClearStart Production Environment**: A production environment tailored for Swedish public sector customers working under strict compliance standards. The production environment uses a wide range of AWS security tools. The APN can choose to customize the environment further to meet special security or performance needs.

The program does not offer an off-the-shelf solution to cloud computing for actors in the public sector but rather a platform for building a solution suitable for meeting the specific demands of a customer. By collaboration between the customer and an APN, a tailored solution that could include data integrity auditing protocols can be obtained. Although AWS does not currently provide any particular service that implements a data integrity protocol such as those examined in Section 4, it is possible to build such functionality using a combination of other AWS services such as AWS Encryption SDK [6].

By working together with experts providing knowledge of today's encryption standards and the tools offered by cloud computing platforms, it is possible to translate theoretical results in data

storage auditing to a more reliable cloud environment. The introduction of such solutions is a step towards bridging the gap between the strict regulatory conditions of actors in the public sector and valuable computing paradigms such as cloud computing offered by providers in the private sector.

5.2. A look into the future

In this thesis, we have identified the most significant properties of data integrity schemes and analyzed several PDP and PoR solutions based on this characterization. From this analysis, we can conclude that it is often difficult to determine which protocol is most suitable for implementation in a given computing environment, as desirable features often require greater computational resources. Several different interpretations of the PDP and PoR frameworks exist and make the area difficult to overlook. Beyond this thesis, there is a further need for standardization and comprehensive guides to the different options available for data integrity auditing. Such efforts will possibly also increase the number of practical implementations offered by cloud computing providers.

As many protocols propose different choices of cryptographic primitives such as encryption schemes, signature schemes, and codings, another possible option for further study is a comparative analysis of practical implementations based on the theoretical results presented here.

References

[1] Amazon Simple Storage Service. *Amazon S3 Service Level Agreement*. Ed. by Inc Amazon Web Services. https://aws.amazon.com/s3/sla. July 2019. (Visited on 05/01/2021).

[2] Gaspard Anthoine et al. *Dynamic proofs of retrievability with low server storage*. 2021. arXiv: 2007.12556 [cs.CR].

[3] Giuseppe Ateniese et al. "Provable data possession at untrusted stores". In: CCS '07. Alexandria, Virginia, USA: Association for Computing Machinery, 2007, pp. 598–609.

[4] AWS Public Sector Blog Team. *Announcing AWS ClearStart for Swedish public sector to accelerate security and regulatory compliance*. file:///home/edun/Zotero/storage/GWV894EG/announcing-aws-clearstart-swedish-public-sector-accelerate-security-regulatory-compliance.html. Oct. 2020. (Visited on 05/03/2021).

[5] Mihir Bellare and Phillip Rogaway. "Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. CCS '93. New York, NY, USA: Association for Computing Machinery, 1993, pp. 62–73. DOI: 10.1145/168588.168596.

[6] Roger Bille, Omegapoint. Private communication. 2021.

[7] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography, version 0.5*. Jan. 2020. (Visited on 05/02/2021).

[8] David Cash, Alptekin Küpçü, and Daniel Wichs. "Dynamic Proofs of Retrievability Via Oblivious RAM". In: *Journal of Cryptology* 30.1 (2017), pp. 22–57.

[9] Oded Goldreich and Rafail Ostrovsky. "Software Protection and Simulation on Oblivious RAMs". In: *Journal of the ACM* 43 (Jan. 1996).

[10] Google Cloud Storage. *Cloud Storage Service Level Agreement (SLA)*. Ed. by Google LLC. https://cloud.google.com/storage/sla. Jan. 2020. (Visited on 05/01/2021).

[11] Google Cloud Storage. *Google Cloud Platform Terms of Service*. Ed. by Google LLC. https://cloud.google.com/terms. Apr. 2021. (Visited on 05/01/2021).

[12] Matthew Green. *What is the random oracle model and why should you care? (Part 5)*. https://blog.cryptographyengineering.com/2020/01/05/what-is-the-random-oracle-model-and-why-should-you-care-part-5/. Jan. 2020. (Visited on 05/02/2021).

[13] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *An Introduction to Mathematical Cryptography*. Springer, 2014.

[14] W. Cary Huffman and Vera Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003.

[15] Ari Juels and Burt Kaliski. “POR:s Proofs of retrievability for large files”. In: *IACR Cryptology ePrint Archive* 2007 (2007), p. 243.

[16] Julien Lavauzelle and Françoise Levy-Dit-Vehel. “New proofs of retrievability using locally decodable codes”. In: *2016 IEEE International Symposium on Information Theory (ISIT)*. IEEE, July 2016.

[17] Peter Mell and Timothy Grance. “The NIST Definition of Cloud Computing”. In: *NIST Special Publication* 145 (2011), p. 7.

[18] Shigeo Mitsunari. *mcl*. `https://github.com/herumi/mcl/blob/master/bench.txt`. 2018. (Visited on 05/05/2021).

[19] Jianbing Ni et al. “Identity-based Provable Data Possession from RSA Assumption for Secure Cloud Storage”. In: *IEEE Transactions on Dependable and Secure Computing* (2020), pp. 1–1.

[20] Binanda Sengupta and Sushmita Ruj. “Efficient Proofs of Retrievability with Public Verifiability for Dynamic Cloud Storage”. In: *IEEE Transactions on Cloud Computing* 8 (2020), pp. 138–151.

[21] Hovav Shacham and Brent Waters. “Compact proofs of retrievability”. In: *Advances in Cryptology - ASIACRYPT 2008* (2008), pp. 90–107.

[22] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. “Practical Dynamic Proofs of Retrievability”. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. CCS ’13. Berlin, Germany: Association for Computing Machinery, 2013, pp. 325–336.

[23] Choon Ben Tan et al. “A survey on Proof of Retrievability for cloud data integrity and availability: Cloud storage state-of-the-art, issues, solutions and future trends”. In: *Journal of Network and Computer Applications* 110 (2018), pp. 75–86.

[24] Cong Wang et al. “Privacy-Preserving Public Auditing for Secure Cloud Storage”. In: *IEEE Transactions on Computers* 62.2 (2013), pp. 362–375.

[25] Wikipedia contributors. *Full Domain Hash — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Full_Domain_Hash&oldid=816794922`. 2017. (Visited on 05/25/2021).

[26] Faheem Zafar et al. “A survey of cloud computing data integrity schemes: Design challenges, taxonomy and future trends”. In: *Computers & Security* 65 (2017), pp. 29–49. ISSN: 0167-4048.

[27] Yan Zhu et al. “Efficient audit service outsourcing for data integrity in clouds”. In: *Journal of Systems and Software* 85.5 (2012), pp. 1083–1095.

Kandidatuppsats 2021
Datalogi
Juni 2021

www.math.su.se

Beräkningsmatematik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm