Delft University of Technology
TNO

Literature Report

Cryptographic Solutions for Security and Privacy Issues in the Cloud

Author:
Daniël Mast

Supervisors:
Zekeriya Erkin
Thijs Veugen

A literature report written in fulfilment of the requirements for the degree of Computer Science

in the

Department of Intelligent Systems *of*
Delft University of Technology

October 2015

Abstract

Faculty Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Computer Science

Cryptographic Solutions for Security and Privacy Issues in the Cloud

by Daniël Mast

This report discusses the cryptographic solutions for security and privacy issues in cloud computing. An overview is given on cloud computing, describing its definition, architecture, and benefits and challenges. Solutions concerning outsourced computation integrity in the cloud are discussed, as well as models for data outsourcing integrity.

Contents

Chapter 1

Introduction

Cloud computing is an emerging field of computing today [10]. In traditional computing, a user would often store its data and perform computations locally. Along with the increasing popularity of the Internet, it has become much easier and faster to share data and send it over the Internet to remote places, offering many new purposes. This is nowadays popularly referred to as using and storing data 'in the cloud'. First is the ability to store data remotely as an extra backup. This means that the local system can crash or get stolen, but the original data can still be retrieved. Second, the size of the data may be so large that it requires a massive storage device, which is inconvenient for storing locally. Third, storing data in the cloud enables multiple users to share data, access it from their own location, and edit it in a collaborative way [26].

Another phenomenon that has become very popular together with this development is performing computations remotely. Local personal computers or smart phones are often much less powerful compared to remote supercomputers. A user can now design a computation, which has to perform a specific task on a (very large) dataset, and assign this computation to the remote supercomputer. Users themselves do not have to worry anymore about buying too many or too few local machines, because they only pay for the resources used at that time. The supercomputers are owned and maintained by the so-called cloud provider, which can efficiently distribute the computations from a large number of cloud users over his machines [10]. An additional advantage of this is that users do not have to maintain their own machines anymore, saving large costs. Apart from outsourcing the user's data, it has become very convenient for consumers to use services from other parties that are not installed as an application on their local machine, but rather run elsewhere [26]. An example of this is webmail. The emails are not stored locally, allowing the user to access them and send emails to others from anywhere he wants. This service can be instantly used, and often does not require installing software

locally. Outsourcing computation and storage, and using remote services make cloud computing a very powerful computing paradigm.

Apart from the benefits, the shift from local to remote storage and computation also introduces new challenges. The user has less insight into what is happening with his data. The cloud provider may use the data for other purposes than intended, or sell the data to other parties. While outsourcing computations, it is hard to check for a user whether the remote party performs the computation in the right way, and whether the returned result is correct. The cloud provider may choose to terminate the computation before finishing, saving energy costs, and returning an incorrect or incomplete result to the user. Even when the cloud provider itself has no malicious intentions, the online available data still has to be protected against other malicious parties [33].

This report focuses on security- and privacy-related challenges that occur in cloud computing. Together with these challenges, a range of solutions, e.g., Verifiable Computation, Provable Data Possession and Third Party Auditing, are discussed. A special focus will be put on cryptography-based solutions. Cryptography provides a measurable certainty of security, and is able to be deployed without the need of special hardware or policies. Solutions including policies, hardware tokens, network adaptations or other facilities are not in the scope of this study.

The rest of the report is organized as follows: Chapter 2 explains in more depth what cloud computing is, its architecture and characteristics, the underlying technique, and why it is of such great importance in the world of computing today. Chapter 3 focuses on the problems in securing outsourced computation, and how to tackle them. Chapter 4 discusses the security and privacy issues of storing user data in the Cloud, also referred to as data outsourcing, and investigates how these issues can be overcome. Finally, Chapter 5 discusses open issues and draws a conclusion.

Chapter 2

Cloud Computing

In Section 2.1, the evolution from grid computing and utility computing to cloud computing will be explained. Section 2.2 shortly introduces the different actors that are involved in the general cloud computing model. Section 2.3 gives a formal definition of cloud computing, according to NIST [24], and lists the characterizing elements. It also presents the service and deployment models. Section 2.5 shows how the architecture looks like. It describes the service layers of which it consists, and the applications that occur in these different layers. Section 2.6 takes a look at the existing implementations that brought cloud computing from theory to practice, and shows the state-of-the-art methods used. The chapter concludes with the benefits and challenges of cloud computing in Section 2.7.

2.1 From Grid Computing to Cloud Computing

Grid Computing, which has been around for about two decades [17], got its name from the comparable electric power grid, in which each house is connected to the energy network and able to get energy on demand. In Grid Computing, the network consists of computers that can use facilities from other computers. Facilities include computing power, hardware, software and data [17], which the user itself does not possess. With Grid Computing, each user is able to use the power of supercomputers on demand, with a wide range of applications, and is not limited to his own equipment. The difference with traditional Distributed Computing is that Grid Computing not only connects computers that solve a problem together, but instead connects computers that work in different administrative domains. This enables using and processing a large variety of information resources.

The property of using software on demand, which Grid Computing enables, lead to the model of Software as a Service (SaaS). In this model, software is hosted by a service provider, and can be used by a user whenever he wants. The user accesses the service over the Internet. Installation is not required. Also, the user does not have to worry about updates. This is the concern of the service provider. This model offers a much more flexible way of using software [15]. Utility Computing is the term that is used when these services are being sold [9]. This monetized way of using services on demand is again very similar to the electric power grid. In [9], cloud computing is defined as the combination of SaaS and Utility Computing, meaning that the provided services are paid for by users. However, cloud computing is more than that, because it not only defines the use case, but also the underlying infrastructure, as in Grid Computing [17]. Cloud computing can, therefore, be defined as the combination of Grid Computing, Utility Computing and SaaS.

2.2 Involved Actors

In cloud computing, three main actors are involved: 1) The cloud provider: owns and maintains the actual data centers and hardware, also referred to as the Cloud. 2) The cloud user/SaaS provider: uses the cloud to build applications, and at the same time, provides these applications as services. 3) The SaaS user, the end user of the provided services. Combined roles are possible, for example when a Cloud Provider also provides services to end (SaaS) users [10]. From here, the titles of these actors will be often used to explain the fundamentals of cloud computing.

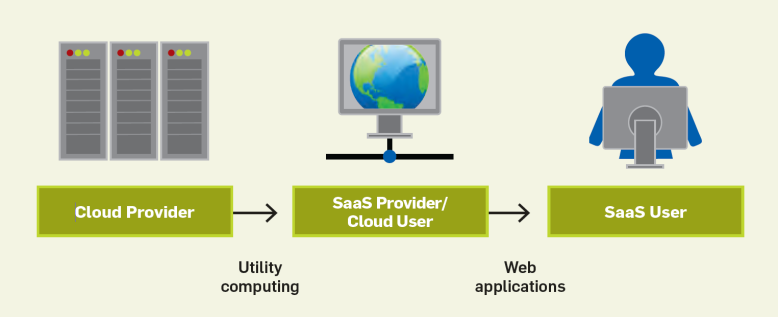


FIGURE 2.1: Actors in cloud computing [10]

2.3 NIST Definition

The National Institute of Standards and Technology (NIST) defines cloud computing as "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." [24]. The NIST further decomposes cloud computing into five essential characteristics, three service models, and four deployment models [24].

2.3.1 Characteristics

Some of the characteristics will follow logically out of the definition, and are discussed in the previous sections, but not all. Below, each characteristic will be shortly explained.

- On-demand self-service. Comparable to the electric power grid, a user can at any time ask for computing power or other resources, without the need of human intervention.
- Broad network access. The network provides the ability to access all resources. Standard protocols allow the resources to be used by all types of different user devices.
- Resource pooling. The whole set of resources is pooled to the users in a multi-tenant model. The physical and virtual resources are dynamically assigned to users on their demand. The absolute location of the resources is irrelevant and often invisible to users.
- Rapid elasticity. A quick change of scale of the resources a user needs should be no problem. Both high and low demands are provisioned. To the user, it appears as if the available resources are unlimited. The user does not have to predict beforehand how many resources are going to be needed. Section 2.4 will discuss this in more detail.
- Measured service. The use of resources is metered by the provider. This enables optimization of resource usage, and also the monitoring of usage per user. With this, it is possible for provider and user to follow a pay-per-use agreement.

2.3.2 Service models

The service models distinguish the different layers from which the architecture of cloud computing is built. This architecture will be fully explained in 2.5. First, the layers will be discussed.

- Software as a Service (SaaS). SaaS enables consumers to use the software of the service provider on demand. The consumer does not have to manage any server- or network-related issues, except possibly certain configuration settings.
- Platform as a Service (PaaS). PaaS gives cloud users the tools to develop applications on the underlying cloud infrastructure from the cloud provider. The cloud user does not have to manage the infrastructure itself.

- Infrastructure as a Service (IaaS). The deepest layer, in which the user is able to manage the operating system, storage and sometimes the network. The applications that are deployed determine how these resources are used. Figure 2.3 shows examples of these applications [24].

2.3.3 Deployment models

A deployment model defines the group that can use the cloud services. In a *private cloud*, a single organization can access the cloud. The organization itself may manage the cloud, or an external party. A *community cloud* is closed for the general public, but shared by a number of organizations, managed by one of them, or an external party. In a *public cloud*, the general public can use the services. The cloud can be owned by an academic government organization or business. A *hybrid cloud* is a combination of two of the previous models. Because a public cloud is often less expensive and more scalable, this may be used for the general services, and a private cloud may be used for more sensitive, internal services [24].

2.4 Resource Provisioning

One of the greatest benefits of cloud computing is the rapid elasticity of resources. We explain this by first explaining the problems of the traditional situation, in which a user cannot use the provider's resource pool, but instead has to manage his own servers. This is shown in Figure 2.2.

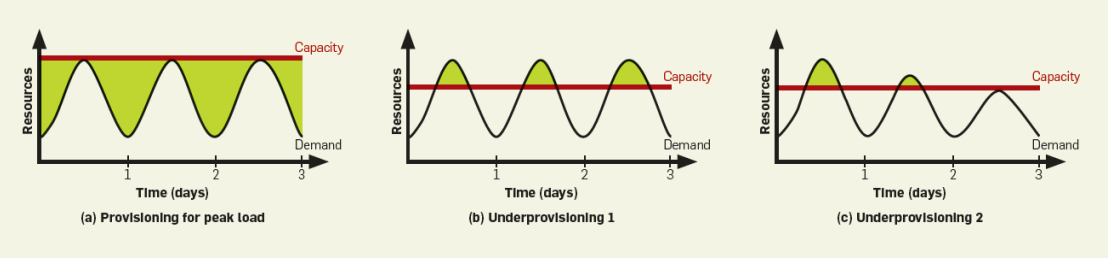


FIGURE 2.2: Provisioning [10]

The diagrams show the number of resources demanded over time. In (a), the owner has successfully predicted the capacity needed, even at peak load. However, at nonpeak times, the biggest part of the capacity is not used. This is called overprovisioning, and leads to a waste of resources. In (b), the owner wastes less resources at nonpeak times, but for the cost of not being able to deal with peak times. This is called underprovisioning. Part (c) shows the result that underprovisioning can have in the longterm. Bad

service can convince a part of the consumers to not use the service anymore, resulting into less revenue. As can be seen, in a traditional setting, the owner of a service that runs on his own server, will always suffer from either over- or underprovisioning [10].

The rapid elasticity characteristic of cloud computing tackles this problem. The cloud user does not have to predict the demands anymore. The cloud provider provides the resources needed, in any form and any quantity. One might argue that this only shifts the problem from user to provider. However, since the resources are pooled over a large number of users, it becomes much easier to predict the total demand. This total demand is much more stable than the demand of a single web server. This enables the provider to prevent underprovisioning and limit overprovisioning. This different architecture drastically increases the efficient use of resources.

2.5 Architecture

Figure 2.3 clearly shows how the three deployment models are stacked as layers. Below is the hardware, with IaaS on top of it, where different types of services manage the resources. This happens in the lowest layer, either physically (e.g., Emulab [4]) or virtually (Amazon EC2). Computationally, services like Hadoop MapReduce are used. For storage, GoogleFS (File System) is a big player. In the PaaS layer, services occur that assist developers. Google App Engine provides tools to build applications that can use the cloud capabilities. Django [2] is a Python-based programming environment that provides a framework for building web applications. The SaaS layer offers services for consumers like Google Docs and other application services. Administration and business support are found throughout the three layers. This includes the metering and billing for used resources, and the additional configurations [33].

2.6 Implementations

Multiple products contribute to the way cloud computing functions today. Some are not specific for cloud computing, and were created for distributed computing purposes (e.g., GoogleFS, Hadoop MapReduce). Others were designed specifically for cloud computing environments (e.g., Amazon EC2, Microsoft Windows Azure, Google App Engine). Both will be discussed.

GoogleFS and HDFS [6]. Located in the IaaS layer, the Google File System was designed by Google with the aim of reaching a high reliability and availability of data.

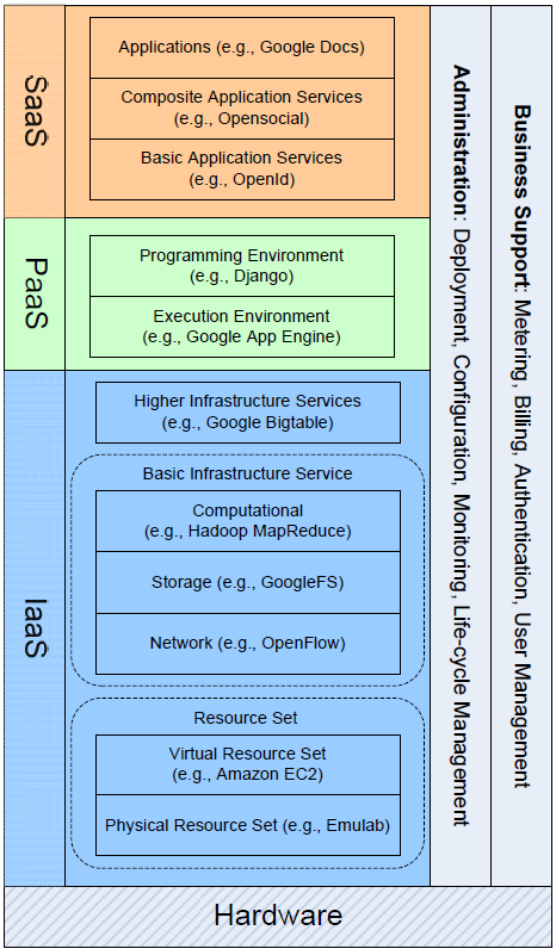


FIGURE 2.3: Cloud Computing Architecture [33]

Files consist of separate pieces of 64 megabytes, replicated over multiple machines, geographically distributed, ensuring that single machine failure does not lead to inaccessible data. The design allows high throughput and speed. The Hadoop Distributed File System (HDFS) [8] is inspired by GoogleFS, and divides files in a similar way. HTTP allows the data to be accessed by a web browser. A protocol enables the different machines to redistribute their data for replication, to keep the availability high [35].

Hadoop MapReduce [7]. A framework that enables a large number of machines to perform a computation on a large data set together. The framework, originally designed by Google, uses a Master machine that assigns so-called jobs to the other machines. The implementation of a job defines the task that one machine should do with the data, and the way in which the results should be combined [35].

Amazon EC2 [3]. With Amazon Elastic Compute Cloud (Amazon EC2), cloud users can manage their own virtual machine, which runs in the cloud. Comparable to a traditional operating system, users can install software, and set up services that are accessible to SaaS users. To increase availability and capacity, an image of the virtual

machine can be obtained, to launch a new identical copy. A downside of the low level control that cloud users have, is that automatic scaling becomes more complex [35].

Microsoft Windows Azure [1]. Comparable to Amazon EC2, Microsoft Windows Azure delivers an operating system-like environment. Comparable to the look and feel of Microsoft Windows, the user can install software. SQL Azure handles the data storage and services. ".NET Services" allows using the cloud-based infrastructure. Windows Azure focuses on languages that are also supported in traditional Windows operating systems, like C++ and Visual Basic [35].

Google App Engine [5]. This platform operates in the PaaS layer, and therefore gives the user less control over the underlying infrastructure, but supports better scaling and quicker deployment of applications. Programming languages like Python and Java are supported, including frameworks that are implemented in these languages. APIs are available that deal with storage in Google BigTable. App Engine liberates the user from server outage and monitoring, by dealing with this automatically [35].

2.7 Benefits and Challenges

The discussed characteristics of cloud computing have clearly shown the power of this paradigm today. To summarize, cloud computing enables a rapid deployment of services, and does not require purchase of local machines. This also liberates the SaaS provider from having to predict the scale of demand, and does not have to change anything when the scale changes over time. The resource pool is designed to take care of this, and the cloud provider simply charges per use. The geographical distribution of machines ensures a high availability of the data and services.

However, the design of cloud computing also incurs new challenges. In [9], a list of top 10 obstacles is composed and divided into three types: concerning either adoption, growth, or policy and business. Adoption-related issues include availability of service, data lock-in, and data confidentiality and auditability. Data lock-in refers to the issue for a user to remove his data from the cloud storage, and being certain that the cloud provider does not retain a copy elsewhere. Growth-related obstacles include data transfer bottlenecks, performance unpredictability, scalable storage, bugs in large distributed systems, and quick scaling. Regarding policy and business, cloud computing has to deal with reputation-fate sharing and software licensing.

In [33], three main challenges of cloud computing are stated, focusing on security and privacy. These are outsourcing, multi-tenancy, and massive data and intense computation. Outsourcing incurs loss of physical control of data and tasks. Multi-tenancy

introduces threats, because applications from multiple users run on the same machines. Security leaks can enable adversaries to access other users' data and applications. Massive data and computation can form a problem, because traditional ways of verifying integrity of data can cause massive overhead. The next two chapters discuss the issues of outsourcing, and massive data and intense computation. Multi-tenancy issues are not discussed, as they are not typically solved by a cryptographic approach.

Chapter 3

Outsourced Computation Integrity

A great advantage and characteristic of cloud computing is that computations can be outsourced. A relatively weak local computing device can transfer its task to a remote supercomputer [18]. This has become extra relevant in the last decade, in which the usage of smartphones has increased drastically. The benefit of portability of such a device is that it can be used anywhere, gathering information at any location [27]. The disadvantage is that its computational capacity for processing this data is limited. Moreover, most consumer machines are incapable of dealing with the extreme amounts of data in a reasonable time. This problem has automatically amplified the need for outsourcing computations to remote supercomputers in the cloud.

The loss of control over the computation raises new challenges. First, how can a user be sure that the cloud provider performs the task correctly, and returns the correct result? It may be cheaper to terminate the computation before finishing, saving processor usage, and providing an incorrect result to the user. The naive way of preventing this is to run the computation locally, and check if the results match. However, this nullifies the advantage of outsourcing. The requirement is that the check is less computationally intensive than performing the task itself. This chapter discusses techniques that ensure the integrity and correctness of outsourced computation. Most methods discussed also ensure the confidentiality of the computation, disabling the untrusted party to use the input or output, or the computation itself for malicious purposes.

In traditional computation, parties that wanted to perform heavy computations often possessed their own supercomputers. There was no other party that they had to trust or audit. With the emergence of cloud computing, the usage of untrusted remote supercomputers increased the interest of being able to verify outsourced computation integrity

[30], called Verifiable Computation (VC). Verifiable Computation enables a client to verify correct execution of his computation that is outsourced to the server. The criterium is that the verification procedure should be less expensive than performing the computation itself. This is also the biggest challenge, as current approaches often rely on computationally costly methods, like Fully Homomorphic Encryption.

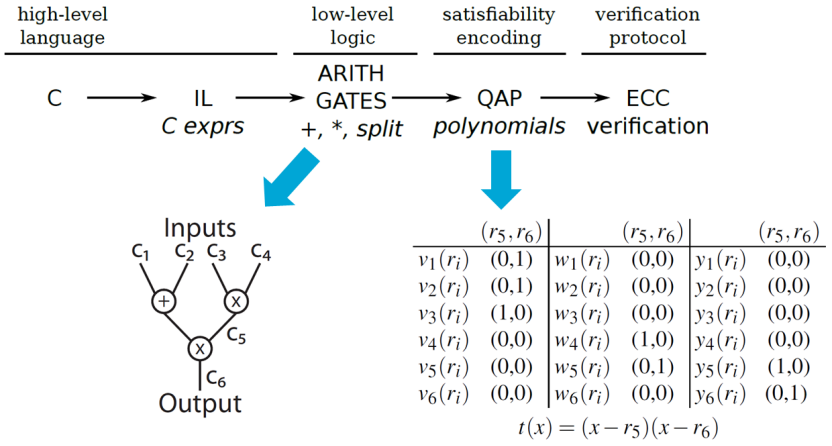
3.1 Interactive Proofs

The work in [21] introduces a modified version of Interactive Proofs. An interactive proof system consists of two parties, the prover and the verifier. The prover, possessing unlimited computational power has the task to convince the verifier of a certain proposition. The verifier can ask questions, and run tests on the replies of the prover, whatever necessary to become convinced or reject the proposition. The verifier is however computationally limited [20]. The research in [21] modifies this definition slightly. They assume that the prover does not have unlimited computation power. Instead, the prover should run in polynomial time as well, which aligns more to reality. Next, it is explained how this system can be used for delegating computations and verifying correctness of the execution. It is argued that this system is particularly efficient for languages computable by log-space uniform NC, which are circuits of polylog(n) depth.

The way in which this efficiency is reached, is based on the log-space uniformity of the circuit. The verifier does not have to traverse the entire circuit, but instead only one path from top to bottom. The complexity is therefore linear to the depth of the circuit, rather than the size. The circuit here is a layered arithmetic circuit. The idea is that the distinct input leaves will together contribute to the output root. The circuit in between matches the performed computation. The prover is challenged to provide a path from the root to a single leaf that contains correct computations. As in every step, the direction of traversal down the tree is randomly selected, the prover can only with low probability guess the path and hide incorrect computations.

3.2 Pinocchio

Pinocchio [28] is a system that enables nearly practical VC. This system makes it possible to construct a general computation, write its code in (a subset of) C, and compile it to a program that is suitable for running the VC protocol between a verifier and a prover. This happens in the so-called Pinocchio's Toolchain, shown in Figure 3.1.



	(r_5, r_6)		(r_5, r_6)		(r_5, r_6)
$v_1(r_i)$	(0,1)	$w_1(r_i)$	(0,0)	$y_1(r_i)$	(0,0)
$v_2(r_i)$	(0,1)	$w_2(r_i)$	(0,0)	$y_2(r_i)$	(0,0)
$v_3(r_i)$	(1,0)	$w_3(r_i)$	(0,0)	$y_3(r_i)$	(0,0)
$v_4(r_i)$	(0,0)	$w_4(r_i)$	(1,0)	$y_4(r_i)$	(0,0)
$v_5(r_i)$	(0,0)	$w_5(r_i)$	(0,1)	$y_5(r_i)$	(1,0)
$v_6(r_i)$	(0,0)	$w_6(r_i)$	(0,0)	$y_6(r_i)$	(0,1)

FIGURE 3.1: Pinocchio's Toolchain [28]

First, the C code is compiled to an arithmetic or boolean circuit representation, consisting of arithmetic gates (+, *) or boolean gates (AND, OR). This circuit can be transformed to a Quadratic Arithmetic Program (QAP) or Quadratic Span Program (QSP). This QAP or QSP consists of a set of polynomials, which are used in the verification protocol. The polynomials allow the verification computations to be much more efficient compared to executing the program itself. In the protocol, bilinear maps are used for encoding the polynomials, before they are sent. With this, the prover can still use the polynomials to prove that he is performing the computation correctly, but at the same time does not know the characteristics of the polynomials. This method is safe, and more efficient compared to Homomorphic Encryption.

The table in Figure 3.1 represents a QAP. The QAP consists of three sets of $m+1$ polynomials (V, W and Y), where m equals the sum of the number of inputs and the number of multiplication gates. V encodes the left input of the gates, W the right input, and Y the output. For example, $v_1(r_6) = 1$, because input 1 is a left input for gate g_6. r_i represents the (arbitrary) root value for gate g_i. The QAP also contains a target polynomial $t(x) = \prod_g (x - r_g)$. This polynomial equals 0 whenever x equals a gate's root value.

When F is a function with n inputs and n' outputs ($N = n + n'$ IO elements), we can say that a QAP Q *computes* F if $(c_1, ..., c_N) \in \mathbb{F}^N$ is a valid assignment of F's inputs and outputs, which is equivalent to the existence of coefficients $(c_{N+1}, ..., c_m)$, such that $t(x)$ *divides* $p(x)$, where $p(x)$ is formulated in Equation 3.1.

$$p(x) = \left(v_0(x) + \sum_{k+1}^{m} c_k * v_k(x)\right) * \left(w_0(x) + \sum_{k=1}^{m} c_k * w_k(x)\right) - \left(y_0(x) + \sum_{k=1}^{m} c_k * y_k(x)\right) \tag{3.1}$$

Equation 3.1 [28] uses the fact that the output of a multiplication gate equals the product of its inputs. Q only computes F if there exists a polynomial $h(x)$ for which $h(x) * t(x) = p(x)$ holds. If $h(x)$ is not constructible, then $t(x)$ does not divide $p(x)$, and Q does not compute F.

During the verification phase, the verifier randomly selects a secret key $s \in \mathbb{F}$, and lets the prover compute $v(s)$, $w(s)$ and $y(s)$ "in the exponent", using the bilinear group, without being able to retrieve the decrypted values itself. From this, the prover can compute $p(s)$ and $h(s)$, and send this as a proof to the verifier. The verifier can verify correctness of the computation by checking whether the obtained $h(s)$ divides $p(s)$.

Pinocchio is faster compared to previous VC systems. However, the circuit representation does not allow mutable state and iteration, making it less useful for applying on many existing C programs. Moreover, the setup of the evaluation and verification key is only efficient when the pre-processing phase is amortized over multiple inputs.

3.3 Non-interactive VC

Research in [18] is the first to use the term Verifiable Computation. Its method introduces interesting new features. First, it argues to be non-interactive, meaning that the client sends a single message to the prover and vice versa. No extra communication is required for proving correctness of the computation. As this still involves two messages, it is more correctly referred to as minimally interactive. To fulfill the rules of Verifiable Computation, the proving protocol must take less time for the client than performing the computation itself. For a single computation, this is not the case. The algorithm is only efficient in an amortized way, meaning that the algorithm only becomes efficient when the pre-processing results can be reused for proving the correctness of a large number of executions of the same algorithm. Second, the method is privacy-preserving, and therefore does not allow the untrusted worker (prover) to learn the original input or output. Next will be explained how this is achieved.

The protocol consists of three phases:

- Preprocessing. The client computes the public and private information about the function F. The computational cost of preprocessing is linear to one execution of the algorithm, but only has to be performed once, and is therefore amortized over multiple executions.
- Input Preparation. When the input x is known, the client computes the public and private information about x and sends the public part of F and x to the worker.
- Output Computation and Verification. The worker computes π_x, which is the encoded value of $F(x)$ and sends it to the client. The client can retrieve $F(x)$ from this value and verify correctness.

The protocol combines Fully Homomorphic Encryption (FHE) [19] with Yao's Garbled Circuits [34] to achieve computational privacy. The function F is first converted to a boolean circuit C, and garbled with Yao's protocol. In Yao's original protocol, the input and output are labeled as long random strings, as well as every internal wire, resulting into a truth table for each gate in the circuit. This, however, disables reuse of the circuit. The worker now possesses a valid output string. For a next computation, the worker can choose to return this string again. It is impossible to verify whether this output is correct. To solve this problem, the protocol is enriched with FHE, which enables reusing the garbled circuit for multiple inputs. The labels that are associated to the input bits are encrypted with a public key. A new public key is generated for each execution, so that the garbled circuit can be reused, and the worker can still perform the homomorphic computation.

Non-interactive VC argues that any future improvement of the FHE scheme will improve their protocol as well, as FHE is used in a black box fashion. This argument, however, is not very convenient today, as FHE is not efficient yet. Therefore, the protocol remains theoretical. Moreover, a more complex algorithm is translated to a relatively large boolean circuit, with a large number of input and output bits, internal wires and gates. Creating random labels for all these wires and truth tables for each gate can also become very impractical.

3.4 VC from Attribute-Based Encryption

The research in [27] defines VC in an even more distributed way. Public delegation is introduced, which means that arbitrary parties are allowed to submit inputs for the delegated function. The benefit of this feature is explained with an example of a clinic, in which a doctor defines the structure of the function F, and the lab assistants provide the actual inputs. The lab assistants can perform the delegation protocol with the cloud, without involvement of the doctor. Public verifiability means that these lab assistants

(the arbitrary parties) are also able to autonomously verify the correctness of the results of the worker.

The method is supported by the use of Key-Policy Attribute-Based Encryption (KP-ABE). Here, a function F is associated with a user's key, and a set of attributes is associated with the ciphertext. The ciphertext can only be decrypted by the key, when the function is true for the attributes in the set. This means that the party that decrypts the ciphertext is convinced that the attributes are true, because otherwise decryption would not lead to the expected plaintext. When using KP-ABE in VC, the attributes correspond with the input values of the function, so that the worker can not cheat by using incorrect inputs.

Chapter 4

Data Outsourcing Integrity

In cloud computing, user data is not stored locally, but instead at a storage facility owned by the cloud provider. This approach, called data outsourcing, has several advantages. First, large local storage devices or servers become unnecessary, saving the user costs of purchase and maintenance. Second, the user himself does not have to worry about physical protection of his data. This means that hard disk failure or theft does not result into permanent data loss. Third, the geographical location of a user becomes irrelevant. His data is stored online, and is therefore universally accessible. The user does not have to use the same device to access his data, or move his data with an external hard drive. As long as the user is connected with the Internet, he can access the data [31].

Unfortunately, this approach of working with outsourced data also incurs serious threats, concerning security of the data and privacy of the user. The threats are all, directly or indirectly, caused by the fact that the user does not physically possess his own data anymore. This chapter discusses the problems and solutions in security and privacy of outsourced data integrity. In the context of integrity in data outsourcing, the user does no longer know whether his data is correctly stored over time. How can the user verify integrity of his data? Moreover, in the context of collaboration between multiple users, and sharing of data, how can one user know that the data originates from a known other user, and that this data is still intact? Besides integrity, some of the methods discussed also provide ways to improve data confidentiality and privacy-preservability. Confidentiality ensures the user that only he accesses his data, and no other unauthorized parties.

Data outsourcing can lead to numerous other issues as well, including availability problems by network failure and accountability problems caused by the complex mixture of involved parties. This report however, mainly discusses the literature that focuses on issues that can be tackled by cryptographic solutions. Cryptography is more present

in integrity- and confidentiality-related issues, and less in availability and accountability. This chapter discusses different methods that improve data integrity in a cloud environment.

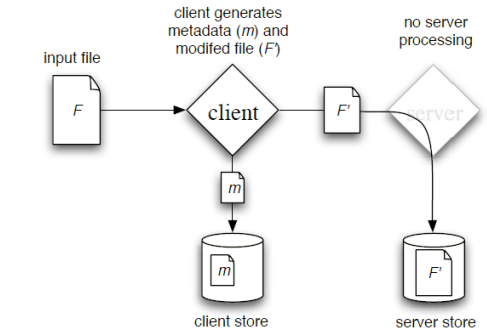
4.1 Provable Data Possession

One approach to ensure data integrity is Provable Data Possession (PDP) [11]. This method enables a client to verify that the server possesses his data, without having to retrieve the entire data. The model implements a challenge/response protocol between client and server, in which the server has to deliver a so-called Proof of Possession. Instead of a deterministic proof, this proof is probabilistic, as in this protocol, it is not possible to verify every bit of data. This would require having to download the entire data set, which is undesirable for large amounts of data. Instead, random samples of blocks are verified. The goal of PDP is therefore to detect misbehavior of a server when it does not possess the complete file anymore. In contrast to some other methods, PDP requires the client to store a small and constant amount of metadata locally, to minimize network communication. This enables verifying large data sets, distributed over many storage devices. The authors propose two schemes that are supposedly more secure than previous solutions, have a constant server overhead, independent of the data size, and have a performance bounded by disk I/O and not by cryptographic computation. The size of the challenge and response are both 1 Kilobit. The method works with homomorphically verifiable tags, that enable possession verification without having to possess the files locally. For each file block, such a tag is computed. The homomorphic property enables multiple tags to be combined into one value. The file blocks, together with their tags, are stored.

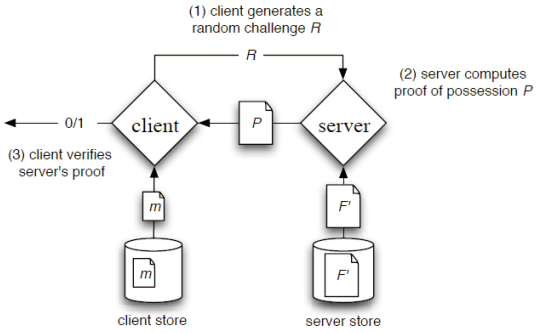
4.1.1 Challenge/response protocol

Whenever the user wants to check whether the server is performing the right behavior, he can decide to send a challenge to the server. The challenge consists of the references to a random subset of file blocks. The server has to construct a proof of possession out of the blocks that are queried, and their tags, to convince the client that the server possesses the blocks, without having to send them. Note here that the server should not be able to construct a proof without possessing the queried blocks. Figure 4.1 clearly shows the steps.

Step (a) shows the pre-processing phase, in which the client uses the file to create the metadata, and stores this metadata locally. The client may modify the file, by appending



(a) Pre-process and store



(b) Verify server possession

FIGURE 4.1: PDP Protocol [11]

other metadata, or encrypting the file. Next, the modified file is sent to the server, which stores it without further processing. In (b), the verification step is shown. The client sends a challenge to the server. The server uses the stored file to create a proof, and sends it back to the client. The client uses the stored metadata to verify this proof.

4.1.2 Requirements

PDP states three types of performance parameters:

- Computation complexity: The costs of pre-processing a client's file, and generating and verifying the proof.
- Block access complexity: The number of blocks needed to generate the proof.
- Communication complexity: Amount of data sent and received during the protocol.

The computational complexity at the server side should be minimized, as the server may interact with many clients concurrently. Therefore, only a subset of the blocks are queried, instead of all. Communication complexity should be minimized as well, to minimize the bandwidth. The authors claim that server misbehavior can be detected with a challenge that contains a constant number of blocks, independent of the total number of blocks stored. This also makes it possible to store a constant amount of local metadata, independent of the size of the data stored at the server.

4.1.3 Scheme

A client C stores a file F on a server S. File F consists of n ordered blocks: $F = (m_1, ..., m_n)$, where m stands for a message, which in this scenario equals a block. T_m is the homomorphically verifiable tag (HVT) for m, which is stored together with the message. HVTs have three important properties:

- Blockless verification: the client does not have to possess the file blocks himself to prove that the server possesses them.
- Homomorphic tags: two tags T_{m_i} and T_{m_j} can be combined into a value $T_{m_i+m_j}$, which is equal to the tag of the sum of the messages $m_i + m_j$.
- HVTs and their proofs have a fixed size, which is much smaller than the corresponding file block.

The scheme consists of four polynomial-time algorithms:

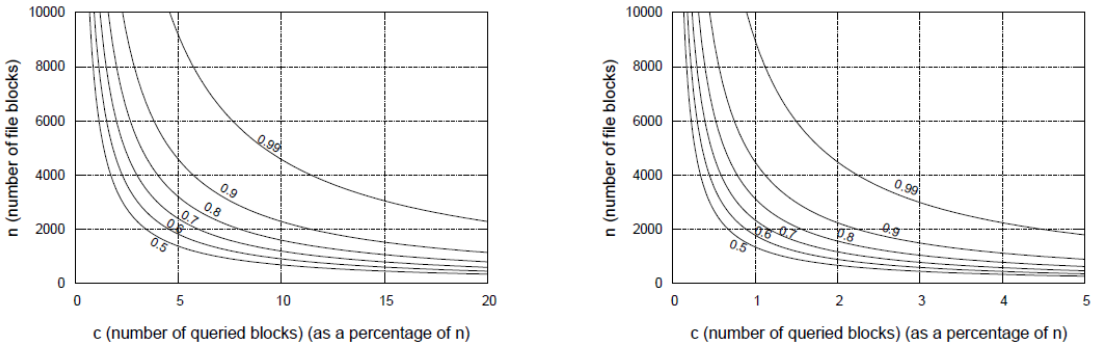
- KeyGen$(1^k) \rightarrow (pk, sk)$ is run by the client, takes a security parameter k, and returns a public and private key pair.
- TagBlock(pk, sk, m) $\rightarrow T_m$ is run by the client, uses the key pair together with a message, and returns the corresponding HVT.
- GenProof(pk, F, chal, $\sum$) $\rightarrow V$ is run by the server, takes the public key, an ordered collection F of blocks, a challenge *chal* and an ordered collection $\sum$ of the corresponding metadata of F. It returns the proof of possession V for the subset of blocks of F that are selected in the challenge.
- CheckProof(pk, sk, chal, V) $\rightarrow$ {"*success*", "$failure$"} is run by the client, and uses the key pair, the challenge and the proof to verify whether the proof of possession is correct.

The client runs KeyGen, and sends the public key to the server. It then runs TagBlock for each file block, and sends F and $\sum$ to the server. The client locally stores the key pair, but nothing else.

4.1.4 Discussion

PDP proposes an efficient protocol that enables a client to detect misbehavior of a server concerning data storage integrity. The client only has to store a limited amount of local data, and the protocol requires minimal communication costs. The homomorphic tags enable possession verification without having to possess the files locally. A downside is that the PDP promotes its method by arguing that previous methods require storage of data at least the size of the data itself. This implies that this method only stores

the data itself and nothing more. However, this method also requires the file blocks to be extended with their tags. The size of the additional is however less proportionally. Second, the proposed scheme only works for storage of static files. After a file has been uploaded to the server, editing the file will cause the protocol to fail [33]. Third, Figure 4.2 shows the relation between the number of file blocks n that are stored, and the number of queried blocks c, as a percentage of n. It turns out that, for a given percentage of deleted blocks t, the number of blocks that should be queried to detect misbehavior with a certain probability, is constant. Therefore the claim stated in 4.1.2 is correct. However, one might question for a real scenario, whether the number of deleted files is a percentage of the whole. If the number of deleted files is instead static, then misbehavior cannot be detected with a static number of queried blocks.



(a) $t = 1\%$ of n (b) $t = 5\%$ of n

FIGURE 4.2: Probability of misbehavior detection [11]

4.2 Proof of Retrievability

Similar to PDP, a Proof of Retrievability (POR) scheme allows a client to execute a protocol with the server, to prove whether the server can still deliver the data to the client, without the client having to store his own data locally, or to download the entire file. However, the difference is that POR does not require the server to possess the original data. It only requires that the server is able to retrieve the data when necessary. Therefore, possession is a stronger form of retrievability. Developments like Error Correcting Codes allow retrievability of the original file when it has become corrupted and possession is lost. Another example is that the server may not possess the entire file, because the client keeps a piece of it locally, or that the server possesses an encrypted version of the file. The authors of [22] focus on proofs of retrievability of large files, requiring minimization of computation and communications costs.

4.2.1 Protocol

In Figure 4.3, the scheme for the POR protocol is displayed. In contrast to PDP, the file is encoded by the user before sending to the server. The key for this encoding is generated at the user side, and stored by the user. Similarly to PDP, the user and archive conduct a challenge/response protocol, in which the server has to prove file retrievability to the verifying user.

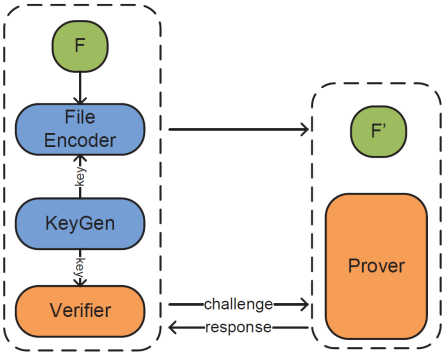


FIGURE 4.3: POR Schematic [33]

For the protocol, the user is required to store a single cryptographic key, independent of the size of the files stored, together with a small amount of dynamic state for each file. During the protocol, only a selected portion of the file is checked, instead of the entire content, making it a probabilistic check. In addition to the encryption, a file is enriched with so-called sentinels, which are blocks with random content that are going to be used in the verification step. Encryption of the file is done so that sentinels are indistinguishable from other file blocks. During verification, the user challenges the archive by sending a number of positions of sentinels to the archive, from which the archive has to return the sentinel values. The underlying idea is that if the archive has changed or deleted a substantial number of files, then the sentinels will probably be changed as well, avoiding the archive to respond with the correct values. A relatively small number of changes may however not change the sentinels, disabling the user to detect malicious behavior. The protocol solves this problem by also implementing error correcting codes, enabling correction of corrupted files [22].

4.2.2 Discussion

POR presents an elegant way to probabilistically prove that an archive can still retrieve correct files to the user. Compared to PDP, a benefit is that POR not only strengthens integrity of the file, but also confidentiality, because of encryption. The downside of this is that encryption involves extra computational complexity. Furthermore, POR

is unclear about the way in which the sentinels are verified. It is stated that only a cryptographic key and dynamic state values are stored. This raises the question how verification can be performed with only this little information. An external source [13] explains the protocol in more detail. It describes a precomputation phase in which the user computes M and Q values. The M values are a summation of sentinel subsets. The Q values are the corresponding encrypted values of M. In the challenge/response phase, the user asks the archive to compute these values as well and return it. The user verifies whether M equals the decrypted Q, and returns true or false accordingly. However, it is not clear why this equality forms a proof of retrievability. This may be possible by the use of a secret decryption key, but lacks explanation. Next, POR distinguishes two scenarios of data loss. In the first one, a small number of bits has changed, leading to a corrupted file, which can be corrected with error correcting codes. In the second, too many bits have changed for correction, but malicious behavior is detected with high probability because sentinels are probably changed too. POR, however, does not clearly describe how many bits can be changed before error correcting codes are unable to correct them, or how many bits it takes to successfully detect malicious behavior. It is not clear whether there exists a number of changes that cannot be corrected nor detected. If this number exists, it incurs a serious security issue. Lastly, it is explained that sentinels are one-time verifiable, because after usage, the archive knows that this is a sentinel, and not a regular file block. The question is however, why sentinels have to be implemented for verification, and why it is not possible to use the regular files itself for this. This would probably also tackle the one-time verifiability issue, because even though this block may have been used for verification before, the archive is still not able to delete it.

4.3 Third Party Auditing

The previous approaches used a direct protocol between client and server. A downside of this is that the client is still burdened with the verification task. This involves local storage of data needed for verification, local verification computation, as well as the communication overhead of the challenge/response protocol [33]. A way to relieve the client from this work is to introduce a third party, which becomes an auditor that takes over the work. However, the outsourcing of auditing from a client to a third party should not introduce new privacy issues. In other words, it is desirable that the third party auditor (TPA) can successfully audit a server without learning the content of the client's data.

4.3.1 Architecture

The architecture of third party auditing in a Cloud environment is shown in Figure 4.4. The three attending parties are the Client (or User), the Server (or Cloud Service Provider) and the Third Party Auditor (TPA). The client and server only communicate when the client wants to read or edit his data. The TPA does the auditing work, executing a protocol with the server, and sending the verification results to the client. In this architecture, it is key that the TPA is trusted by both client and server. The client must trust that the TPA performs the verification checks correctly, reporting misbehavior when it is detected (false negative). The server must trust that the TPA does not unfairly report misbehavior when this is not the case (false positive). The TPA usually possesses more computational resources than the client, so that it can successfully perform the auditing computations, which would be too heavy for the client.

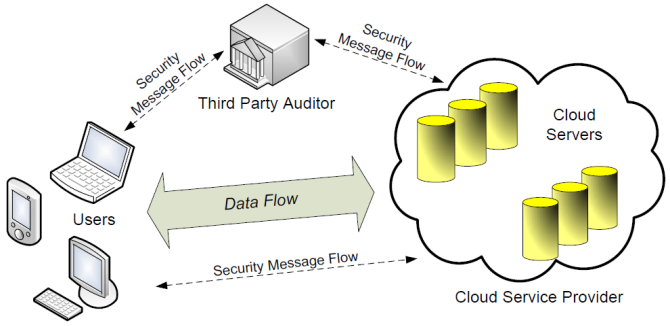


FIGURE 4.4: TPA Architecture [31]

4.3.2 Techniques

The research in [31] comes up with a scheme that enables third party auditing. The method uses the following techniques:

- Public key based homomorphic authenticators. They work in a similar way as the HVTs explained in 4.1.3. A homomorphic authenticator is an encoded version of the original file block, that can be used to check correctness of the block, without possessing or accessing the file block itself. This is a very convenient property for third party auditing, because the third party can use these homomorphic authenticators to do verification without obtaining knowledge about the client's files. Homomorphic authenticators are implemented using bilinear maps.
- Random masking. This prevents that a TPA can build a linear group of linear equations from which the file content of the client can be derived. This however does not affect the ability to audit.

4.3.3 Protocol

The protocol consists of four algorithms. *KeyGen* is run by the client and sets up the scheme. *SigGen* is run by the client and generates verification metadata used for auditing. *GenProof* is run by the server and generates the proof of correctness of data. *VerifyProof* is run by the TPA and verifies the server's proof.

The client first runs KeyGen and SigGen. He sends the file to the server and the metadata to the TPA. Then he deletes his local data. In the auditing phase, the TPA creates a challenge and sends it to the server. The server runs GenProof, sends the proof to the TPA, and the TPA runs VerifyProof to verify.

4.4 Dynamic Provable Data Possession

Both client/server auditing as third party auditing until now supported only the storage of static data. Insertion and modification of files would disable the client to detect malicious behavior of the server. As the services in cloud computing demand more than just archival storage, it is of critical importance that dynamic data too can be tested for integrity. Dynamic Provable Data Possession (DPDP) [16] is the first technique that enables this, stating the four dynamic operations that have to be supported: append, insert, modify, and delete. It uses a variant of authenticated dictionaries based on rank information, involving computational overhead for dynamic updates. However, this overhead is low in practice. The research also introduces three operations: PrepareUpdate, PerformUpdate and VerifyUpdate. The actual update is performed by the server, but the preparation and verification are done by the client [33].

Research in [32] is the first to combine dynamic data auditing with a third party. It uses an improved version of the POR method, combined with bilinear maps and a modified version of the Merkle Hash Tree (MHT) [25] construction for block tag authentication. The aim is to give the client a periodical integrity verification check of his remotely stored data, without requiring local copies at the client side or computational overhead for the client. First is explained why the original POR method is not suitable for dynamic data operations. Every file block is assigned an index i for the order. Insertion of a new block requires all following blocks to increase their index by 1. This also requires updating all their signatures, which incurs a high computational overhead. The new method removes the index information in the signature, and instead creates a tag that is only constructed by the content of the block, and not his position in the file.

4.4.1 Merkle hash tree

An MHT is an authentication structure in the form of a binary tree, that proves that a set of elements is undamaged and unaltered. The leaves are hash values of the data blocks. A parent is a hash value of the sum of hashes of its children. The root node can therefore be used to verify whether the entire data set is authentic.

Figure 4.5 shows how using an MHT architecture enables dynamic data. Block n_2 is modified, becoming n_2', and its hash value in the tree is updated. Now, all parents have to be updated as well, all the way up to the root. This requires the hashes of the siblings, which the server provides to the verifier.

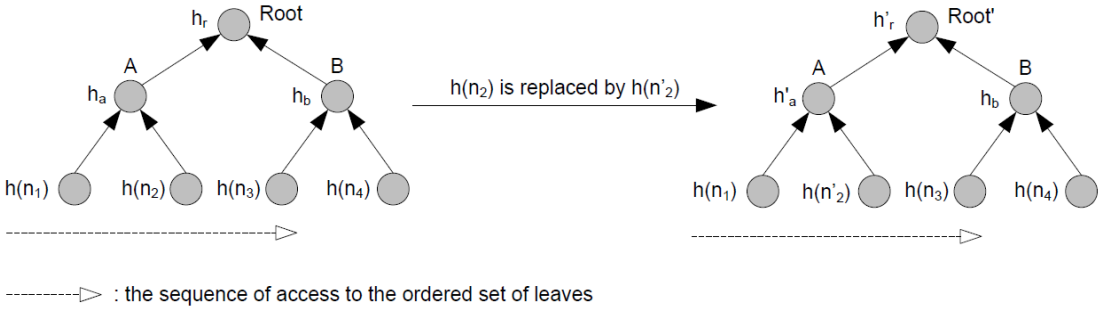


FIGURE 4.5: MHT modification operation [32]

4.4.2 Protocol

The protocol consists of the following algorithms:

- $KeyGen(1^k) \rightarrow (pk, sk)$. Run by client. Creates a public and private key pair.
- $SigGen(sk, F) \rightarrow (\Phi, sig_{sk}(H(R)))$. Run by client. Creates a set of signatures $\Phi = \{\sigma_i\}$ for each block m_i in file F, and a signature of the root R (signed with the private key) of the Merkle hash tree.
- $GenProof(F, \Phi, chal) \rightarrow (P)$. Run by server. Constructs a proof out of file F, the signature set, and the challenge *chal* (created by the TPA).
- $VerifyProof(pk, chal, P) \rightarrow \{TRUE, FALSE\}$. Run by TPA. Verifies the proof, using the public key and the corresponding challenge. Returns true or false.
- $ExecUpdate(F, \Phi, update) \rightarrow (F', \Phi', P_{update})$. Run by server. Performs the update on the file, and returns the new file and signatures set, as well as a proof for the update.
- $VerifyUpdate(pk, update, P_{update}) \rightarrow \{(TRUE, sig_{sk}(H(R'))), FALSE\}$. Run by client. Verifies whether the update proof is correct, meaning that the update has been performed correctly. If correct, returns a new root signature.

4.4.3 Discussion

The need for the verifier to receive the siblings' hashes from the prover, can form a security risk, as the prover can provide certain values that confirm his proof, even when it is not correct. Unfortunately, this risk is not clearly elaborated on. Another indistinctness is the distribution of work over client and TPA. The research claims that the TPA relieves the client by taking over most of the work, but as the protocol algorithms show, most algorithms are still conducted by the client. Also, the client still needs to store the private key for signing.

4.5 Distributed Storage

Previous methods strongly contribute to the detection of misbehavior of a server, deleting or wrongly modifying files of the client. However, after detection, there is no way in which the files can be retrieved again (ECCs will work only for small changes). HAIL (High-Availability and Integrity Layer) [12] is a distributed setting in which the client's files are spread across multiple servers with redundancy. It reuses the POR method for integrity checking, and supplements it with a file recovery mechanism. When one server has lost data, it communicates with the other servers to retrieve the correct data. Future research has to enable dynamic operations on data, as of now, only static data is supported.

4.6 Authenticated Data Structures

So far, this chapter has discussed methods for a client to ensure that his data is stored correctly. Another scenario is possible, in which other users except the data owner want to be certain that the data they query is equal to the data that the owner stored in the first place. This is where authenticated data structures (ADS) become useful. In [29], ADSs are defined as "a model of computation where untrusted responders answer queries on a data structure on behalf of a trusted source and provide a proof of the validity of the answer to the user" [29]. The supporting example introduces a stock exchange, in which stock quotes from the main stock exchange are distributed over brokerages. The main stock exchange is trusted, but the brokerages are not. An investor wants to be sure that the stock quotes he retrieves from a brokerage are identical to the ones at the stock exchange.

For different reasons it may be very convenient to introduce a middle party between original source and user. Data can be replicated geographically, bringing it closer to the

user, reducing latency. Replicated servers can remedy denial-of-service attacks. Also, scalability can be improved [29].

The models is formalized as a structured collection S of objects and three parties: the source, the responder, and the user. When the source performs an update, it sends the new data to the responder, along with *structure authentication information*, consisting of a time-stamped sign. When a user performs a query to the responder, the responder enriches the data with *answer authentication information*, with which the user can proof that the data is valid [29]. A similar definition, but with different party names, is given in [23]. Figure 4.6 shows the corresponding scheme. The data owner sends his data to the publisher, which owns the database. The clients send queries to the publisher to get the data, who returns the result, and a verification-object. In contrast to the original scheme, the owner also communicates with the clients, in the form of sending a small digest that helps the clients to verify the results.

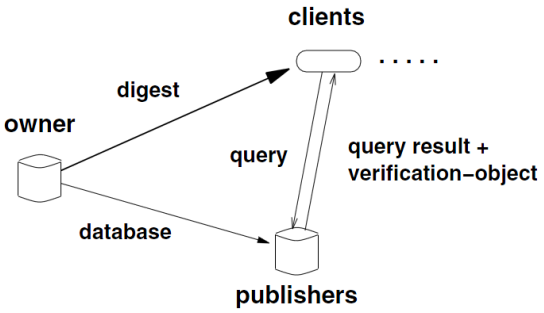


FIGURE 4.6: ADS scheme [23]

A comparable model can be used very effectively in a cloud environment. In [14], the owner is referred to as the writer, the clients are readers, and the publisher is an untrusted cloud server. The digest that the writer sends to the readers is a short authenticator value.

Chapter 5

Conclusion & Discussion

This report has given a broad overview on integrity-related issues in Cloud Computing. A clear view of the definition of Cloud Computing was given, along with its architecture, its benefits and challenges. A distinction was made between integrity of outsourced computation, and data outsourcing. For both fields, many methods were discussed that improve integrity in the cloud.

Future work should continue improving the efficiency of verification techniques, in order to achieve practical usage. Here, it is important that both the requirements of users as well as the cloud provider have to be taken into account. Full confidentiality of a user's data might discourage cloud providers to store their data, as it cannot be used anymore for analytics.

In the context of computation integrity, most algorithms still require computationally expensive pre-computation steps, requiring one function to be executed many times before it becomes efficient. This is not always a realistic scenario. Often, a client may only want to execute a function once by the server. Future work should make verifiable computation for these types of functions more efficient.

Regarding data integrity, future work should combine misbehavior detection of the server with methods to retrieve the data, because detection only is not very useful when retrieval is impossible, causing permanent loss. When a large amount of data is lost or damaged, error correcting codes will be insufficient, so this will have to be supported by additional methods. When a third party is active, the client should not be still imposed with computations for key and signature generation, and update verification. Instead, the client should be genuinely relieved from these tasks, to obtain full efficiency of a TPA.

In my own work, I will continue focusing on authenticated data structures. I will implement different existing techniques, compare storage complexity, and performance of query and update operations, and look for possible improvements of weaknesses of existing schemes, aiming to bring deployment of these structures in a cloud environment closer to reality.

Bibliography

[1] Microsoft Windows Azure. URL azure.microsoft.com.

[2] Django. URL www.djangoproject.com.

[3] Amazon Elastic Compute Cloud. URL aws.amazon.com/ec2.

[4] Emulab. URL www.emulab.net.

[5] Google App Engine, . URL cloud.google.com/appengine.

[6] Google File System, . URL research.google.com/archive/gfs.html.

[7] Hadoop MapReduce. URL wiki.apache.org/hadoop/MapReduce.

[8] Hadoop Distributed File System. URL hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[9] M. Armbrust, O. Fox, R. Griffith, A. D. Joseph, Y. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. M.: Above the clouds: a Berkeley view of cloud computing. 2009.

[10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[11] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609. Acm, 2007.

[12] K. D. Bowers, A. Juels, and A. Oprea. Hail: a high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 187–198. ACM, 2009.

[13] K. D. Bowers, A. Juels, and A. Oprea. Proofs of retrievability: Theory and implementation. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 43–54. ACM, 2009.

[14] C. Cachin. Integrity, consistency, and verification of remote computation, 2014. URL www.zurich.ibm.com/~cca/talks/veriftut.pdf.

[15] A. Dubey and D. Wagle. Delivering software as a service. *The McKinsey Quarterly*, 6(2007):2007, 2007.

[16] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia. Dynamic provable data possession. *ACM Transactions on Information and System Security (TISSEC)*, 17 (4):15, 2015.

[17] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[18] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO 2010*, pages 465–482. Springer, 2010.

[19] C. Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[20] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 59–68. ACM, 1986.

[21] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 113–122. ACM, 2008.

[22] A. Juels and B. S. Kaliski Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597. Acm, 2007.

[23] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, 2004.

[24] P. Mell and T. Grance. The NIST definition of cloud computing. *Special Publication 800-145, National Institute of Standards and Technology Gaithersburg*, 2011.

[25] R. C. Merkle. Protocols for public key cryptosystems. In *Proceedings of the 1980 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 14-16, 1980*, pages 122–134, 1980.

[26] M. Miller. *Cloud computing: Web-based applications that change the way you work and collaborate online*. Que publishing, 2008.

[27] B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Theory of Cryptography*, pages 422–439. Springer, 2012.

[28] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 238–252. IEEE, 2013.

[29] R. Tamassia. Authenticated data structures. In *Algorithms-ESA 2003*, pages 2–5. Springer, 2003.

[30] M. Walfish and A. J. Blumberg. Verifying computations without reexecuting them: from theoretical possibility to near-practicality. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 20, page 165, 2013.

[31] C. Wang, Q. Wang, K. Ren, and W. Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9, 2010.

[32] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *Computer Security–ESORICS 2009*, pages 355–370. Springer, 2009.

[33] Z. Xiao and Y. Xiao. Security and privacy in cloud computing. *Communications Surveys & Tutorials, IEEE*, 15(2):843–859, 2013.

[34] A. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[35] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.