Addressing the Data Location Assurance Problem of Cloud Storage Environments

Ali Noman

A thesis submitted in partial fulfillment of the requirements for the
Doctorate in Philosophy degree in Computer Science

School of Electrical Engineering & Computer Science
Faculty of Engineering
University of Ottawa

Abstract

In a cloud storage environment, providing geo-location assurance of data to a cloud user is very challenging as the cloud storage provider physically controls the data and it would be challenging for the user to detect if the data is stored in different datacenters/storage servers other than the one where it is supposed to be. We name this problem as the "Data Location Assurance Problem" of a Cloud Storage Environment. Aside from the privacy and security concerns, the lack of geo-location assurance of cloud data involved in the cloud storage has been identified as one of the main reasons why organizations that deal with sensitive data (e.g., financial data, health-related data, and data related to Personally Identifiable Information, PII) cannot adopt a cloud storage solution even if they might wish to. It might seem that cryptographic techniques such as Proof of Data Possession (PDP) can be a solution for this problem; however, we show that those cryptographic techniques alone cannot solve that. In this thesis, we address the data location assurance (DLA) problem of the cloud storage environment which includes but is not limited to investigating the necessity for a good data location assurance solution as well as challenges involved in providing this kind of solution; we then come up with efficient solutions for the DLA problem. Note that, for the totally dishonest cloud storage server attack model, it may be impossible to offer a solution for the DLA problem. So the main objective of this thesis is to come up with solutions for the DLA problem for different system and attack models (from less adversarial system and attack models to more adversarial ones) available in existing cloud storage environments so that it can meet the need for cloud storage applications that exist today.

Acknowledgment

It is my great pleasure to thank the many people who made this thesis possible.

I am out of words to show my gratitude to my supervisor, Professor Carlisle Adams. His distinguished expertise and deep understanding of the subject matter helped me a lot throughout my Ph.D. study; I always knew who to contact, when I encountered an issue during my research work. My supervisor was always there when I needed him. I always got a prompt response from him whenever I needed any feedback during devising a solution, writing a paper for publication, and writing my thesis. Aside from the academic work, I received continuous support from my supervisor in my tough time; especially in the last 2 years period when I was passing very difficult time (due to some family matters, having a newborn, and my new job). I am deeply grateful to him for his great guidance, continuous support, and considerable patience. Thank you, Professor!

I would like to also thank Professor Anil Somayaji and Professor Guy-Vincent Jourdan for their valuable feedback on my thesis proposal; it helped me a lot.

I would like to show my deepest gratitude to my mother, Late Nurun Nahar; whatever I have achieved so far in my life, it is because of you. Being the youngest son of a big family, I have been continuously receiving moral support from my brothers, sisters, and brother-in-law; thank you, everyone, for your continuous support!

Last but not least, I would like to thank my beloved wife, Fatma Zafreen, who has been very considerate throughout my Ph.D. study; especially for the last one year or so, she has been managing almost all family matters alone, with our two kids (8.5-year-old daughter, Laurin Noman and 13 months old baby boy, Ayaan Noman) in order to provide me space for finishing my thesis work. The unconditional love of my little treasures, Laurin and Ayaan, has always been a source of inspiration for me.

Dedicated to

My mother, Late Nurun Nahar,

My two kids: Laurin Noman and Ayaan Noman, and

My wife, Fatma Zafreen

Table of Contents

Abstract ……..ii

Acknowledgment iii

Table of Contents v

List of Figures ix

List of Tables x

List of Acronyms xi

1 Introduction 1

1.1 Background *1*
1.1.1 Cloud Computing 1
1.1.2 Data Storage in Cloud 2

1.2 Data Location Assurance Problem- Our Research Problem *2*

1.3 Motivation *4*
1.3.1 A solution is required to attract more Organizations in adopting cloud service 4
1.3.2 This is a challenging and unsolved problem which needs addressing 4

1.4 Objectives *6*

1.5 Research Methodology *6*

1.6 Publications *7*

1.7 Contributions *7*

1.8 Outline *9*

2 Foundations 10

2.1 Cloud Computing *10*
2.1.1 Cloud Architecture 10
2.1.2 Overview of Amazon S3 (Simple Storage Service) 11
2.1.3 Overview of Microsoft Azure Storage 14
2.1.4 Overview of Google Cloud Storage 17

2.2 Trusted Platform Module (TPM) *19*
2.2.1 TPM 1.2 vs TPM 2.0 20

2.3 Internet Host Geolocation *21*

2.4 Relevant Cryptographic Primitives *22*
2.4.1 Zero Knowledge Sets 22

2.4.2 CP-ABE 24
2.4.3 Proof of Data Possession 25
2.4.4 Proof of Retrievability 25
2.4.5 Pairing-based Cryptography: Bilinear Pairing 26
2.4.6 Authenticated Encryption (AE) 26
2.4.7 Well known Cryptographic primitives 28
2.5 Related Work 28
2.5.1 Significance of a solution for the Data Location problem & its associated challenges 28
2.5.2 Significant Attempts 29
2.6 Solution Requirement & Strategy 35
2.6.1 Solution Requirement 35
2.6.2 Strategy 36
2.7 Conclusion 37
3 Proposed Solution 1: DLAS 39
3.1 Objective of this solution 39
3.2 System and Attack Model 39
3.3 Cryptographic Primitives Used 41
3.4 The Proposed Solution: DLAS 42
3.4.1 DLAS Scheme-Formal definition 44
3.4.2 DLAS Construction 47
3.4.3 Discussion & Analysis 52
3.5 Conclusion 58
4 Proposed Solution 2: HDLAS: A 2-entity based solution 60
4.1 Motivation & Objective of this solution 60
4.2 System and Attack Model 61
4.3 Cryptographic Primitives Used 62
4.4 Overview of the Proposed Solution: HDLAS 62
4.5 Hardware-based DLAS Scheme-Formal definition 64
4.6 How to Construct Hardware-based DLAS 67
4.6.1 Adopting Hardware-based DLAS in Microsoft Azure: 68
4.7 Discussion & Analysis 69
4.7.1 Security Analysis of Hardware-based DLAS 69
4.7.2 Implementation and Performance Issues 70
4.8 Conclusion 71
5 Proposed Solution 3: A PDP Solution for Cloud Storage Environments 73
5.1 Motivation & Objective of this Solution 73
5.2 System and Attack Model 74
5.2.1 System Model 74

5.2.2 Attack Model 76

5.3 Limitations of Existing PDP schemes: Why do we need a new PDP solution for the cloud? 77

5.4 Defining new requirements of a PDP Solution for cloud storage environments 78

5.5 Proposed PDP Solution 79
5.5.1 Notation & Preliminaries 79
5.5.2 General Idea 80
5.5.3 Definition of PDP solution for cloud storage environments 81
5.5.4 Details of the Proposed PDP Solution 84
5.5.5 An example of the PDP scheme 89

5.6 Security Analysis 90
5.6.1 Offering 128 bits of security 90
5.6.2 Offered Security Guarantees 90

5.7 Performance Analysis 93
5.7.1 Performance Numbers 93
5.7.2 Performance Numbers Comparison 96

5.8 Construction of DLAS using our proposed PDP 98

5.9 Conclusion 100

6 Proposed Solution 4: Further Enhancement of DLAS Solutions 101

6.1 Enabling Public Verifiability in our proposed solution 101
6.1.1 Enabling Public Verifiability- General Idea 102
6.1.2 Enabling Public Verifiability Using PKI 103
6.1.3 Enabling Public Verifiability Using CP-ABE 106

6.2 Supporting Dynamic Operation on Outsourced Data 108
6.2.1 General Idea 108
6.2.2 Block Update 110
6.2.3 Block Deletion 111
6.2.4 Batching Updates and Deletions 112
6.2.5 Block Append 112
6.2.6 Discussion 114

6.3 DLASPDPΔ*: A DLAS solution with the possible optimal security guarantee. 115*
6.3.1 System & Attack Model 115
6.3.2 Key Properties of Proposed Solution 117
6.3.3 Proposed Solution Description 117
6.3.4 Discussion 119

6.4 Conclusion 120

7 Conclusions & Future Work 121

7.1 Conclusion 121

7.2 Future Work 125

References 126

Appendix 1 Cloud Computing History 134

Appendix 2 NIST Definition of Cloud Computing 135

Appendix 3 Further Explanation on The Detection Probability of PDP Scheme 138

List of Figures

Figure 1 A simple example of access tree which allows the access if the user is userID1 and the data is either dataID2 or dataID5 24
Figure 2 Definition of Bilinear Pairing 27
Figure 3 System Model considered for DLAS solution 40
Figure 4 The Bootstrapping Phase 48
Figure 5 The User Data Registration Phase 50
Figure 6 The Offline Phase 51
Figure 7 The Data Location Verification Phase 53
Figure 8 The original PDP scheme proposed by Ateniese et al. 63
Figure 9 System Model for Proposed PDP Solution 76
Figure 10 Definition of Our Proposed PDP Solution. It includes all 4 algorithms 83
Figure 11 Execution flow of our proposed PDP scheme 85
Figure 12 Pseudocode for Txor computation of our previously proposed PDP solution 109
Figure 13 Block Update Algorithm 111
Figure 14 Logical representation of outsourced data for supporting append operation 113

List of Tables

Table 1 Cryptographic Support Comparison between TPM 1.2 and TPM 2.0 21
Table 2 Classification of existing PDP schemes 32
Table 3 Performance Comparison with some existing PDP schemes 96
Table 4 Performance comparisons with an concrete example 98
Table 5 A summary of all proposed solutions 123

List of Acronyms

Acronym	Definition
ABE	Attribute-Based Encryption
AE	Authenticated Encryption
AEAD	Authenticated Encryption with Additional Data
AES	Advanced Encryption Standard
Amazon S3	Amazon Simple Storage Service
AWS	Amazon Web Service
CA	Certificate Authority
CP-ABE	Ciphertext Policy- Attribute-Based Encryption
CSP	Cloud Storage Provider
DLA	Data Location Assurance
DLAS	Data Location Assurance Service
ECIES	Elliptic Curve Integrated Encryption Scheme
MAC	Message Authentication Code
MAS	Microsoft Azure Storage
IaaS	Infrastructure-as-a-Service
IoT	Internet of Things
PaaS	Platform-as-a-Service
PBC	Pairing-Based Cryptography
PCR	Platform Configuration Register
PDP	Proof of Data Possession
PII	Personally Identifiable Information
POR	Proof of Retrievability
RBAC	Role-Based Access Control
REST	REpresentational State Transfer
SaaS	Software-as-a-Service
SOAP	Simple Object Access Protocol
SLA	Service Level Agreement
TPM	Trusted Platform Module
VHD	Virtual Hard Disk
ZKS	Zero Knowledge Sets

1 Introduction

1.1 Background

1.1.1 Cloud Computing

In 1961, John McCarthy, the computer scientist and inventor of the term "Artificial Intelligence", made a prediction that "Computing may someday be organized as a public utility, just as the telephone system is a public utility [1]." The introduction of cloud computing takes us very close to that prediction. The main reason behind the quick popularity of cloud computing is that cloud computing facilitates a cloud user (an organization or an individual) to use computation and storage resources on demand as a "pay as you go" approach rather than owning (and maintaining) them. The authors of [2] provide an excellent discussion on the evolution of cloud computing under the term "cloud computing history"; we include it in Appendix 1.

Definition of cloud computing: There are many definitions of cloud computing [3] [4] [5]. However, the US National Institute of Standards and Technology (NIST) has published a working definition [6] that seems to be able to capture the commonly agreed aspects of cloud computing. This definition describes cloud computing using:

- Five characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service.
- Four deployment models: private clouds, community clouds, public clouds, and hybrid clouds.
- Three service models: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

Note that [6] also includes definitions of all deployment models and all service models.

1.1.2 Data Storage in Cloud

Storage service is one of the most popular services, if not the most popular one, offered by a cloud. This service facilitates any organization to shift their burden of managing data towards the cloud storage provider at the expense of losing physical control of their data. Depending on the client's security requirements for their data, now the CSP is not only responsible for the confidentiality, integrity, and availability of the stored data but also it has to make sure that there is no misuse of the data by any unauthorized external/internal entity. There is no doubt that if the CSPs are totally dishonest, there will be no security guarantees for client's data. But in order to stay up in the competitive cloud storage service market, we can assume that CSPs will not be totally dishonest and in fact all CSPs have been trying their best to meet their client's data storage need so that they can increase their business by attracting more clients. For example many CSPs such as Amazon Simple Storage Service (Amazon S3) [7] ,Microsoft Azure (previously known as Windows Azure) [8] and Google Cloud Storage [9], now not only offer data encryption services to their clients, but also store client's data across multiple geographically distant data centers to ensure higher availability and better protection in the event of any natural disasters such as earthquake, flood and so on. This standard practice of copying a piece of data in multiple geographically distant data centers is known as Geo-replication of data. Most of the renowned CSPs now do the geo-replication of data for ensuring higher availability.

1.2 Data Location Assurance Problem- Our Research Problem

"In a cloud environment, a user does not have the physical control of their data and thus does not know for sure where his data is physically located at a given instance of time"- this is one of the main security issues in cloud environments which need addressing since it creates a number of major security concerns for cloud users [10] [11] [12] [13] [14]. Like other works ([11] [12])**,** we agree with this concern and in one of our previous works [15], we name this security issue as the "**Data Location Assurance (DLA) problem**" of cloud storage environments.

It is true that at present, some cloud service providers (e.g., Amazon S3) allow their users to specify a preference for the geographic location of data in the Service Level Agreement (SLA). However, in this case, you as a user have no means to continuously verify that cloud service providers are meeting their contractual geographic obligations; by the time you become aware of a violation (which may be unintentional/accidental) of this contract, your data has already been affected [16]. And it seems that, even if not on a regular basis, such (accidental) incidents can happen in the cloud: recently there have been a number of instances wherein a cloud outage rendered business services inaccessible to cloud users for a significant period of time. For instance on February 28, 2017, due to Amazon Web Service S3 outage, many websites and apps were fully or partially broken [17]; some of the affected websites and services include Quora, newsletter provider Sailthru, Business Insider, Giphy, image hosting at a number of publisher websites, file sharing in Slack, and many more. Connected lightbulbs, thermostats, and other IoT (Internet of Things) hardware were also being impacted, with many unable to control these devices as a result of this outage [17]. A similar kind of outage was also observed with Microsoft Azure on February 19, 2017 that lasted for 5 hours, but it did not get the same level of attention as the above-mentioned incident [18]. This kind of outage was also observed in the past. For instance, on Dec 24, 2012, Netflix services, hosted on Amazon Web Services (AWS), were unavailable to customers for more than 12 hours [19]. Similar data outages were reported by Dropbox users [19] [20] and Xbox live users [19].

The standard practice of geo-replication of data (i.e., keeping multiple copies of a data in different datacenters to increase service availability) by some Cloud Storage Providers (CSPs) such as Microsoft Azure, Amazon S3, raises the following question:

How can we be sure that our geographic location preference for cloud data mentioned in the SLA, has never been violated, particularly if the CSP might move a piece of the data to a location we did not choose in order to protect against service unavailability and data outage?

Addressing the above questions is synonymous to addressing the data location assurance problem. Data Location Assurance (DLA) problem is a very challenging problem which needs to be solved in order to maintain the continuous growth of the cloud storage business.

We consider this challenging data location assurance problem of the cloud storage environments for the geo-replication of data setting as our research problem.

In order to solve this data location assurance problem in a geo-replication of data setting, we have to also find answers for the following research questions: *How can we locate all the storage servers having the data at the moment? Do those storage servers really have the data (and it is intact) at the moment? How can we be sure about that? Do those servers' locations satisfy the clients' geo-location preference?*

1.3 Motivation

Here we discuss our main motivations for choosing the Data Location Assurance Problem in geo-replication of data setting as our research problem.

1.3.1 A solution is required to attract more Organizations in adopting cloud service

Apart from security and privacy risks involved in adopting any cloud service, the inability to get verifiable/provable geo-location assurance of cloud data from the cloud can be considered one of the main reasons why organizations that deal with sensitive data (such as financial data, health-related data, and data related to Personally Identifiable Information, PII) cannot adopt a cloud storage solution even if they really want to. The geographic location of the data has significant effects on its confidentiality and privacy; the reason is that the data stored on the cloud will be subject to the laws and regulations of the country where it is physically stored [21] [22]. More precisely, health and personal privacy laws of a country might enforce restrictions on the physical location of the data. For example, the Canadian Personal Information Protection and Electronic Documents Act [23] requires that any PII related data has to be stored within Canada and so any outsourcing of such data storage must be with a guarantee about the location of the stored data. There is no doubt that a good solution for the data location assurance problem will make a huge impact on the rapid growth of cloud storage business as it will motivate companies dealing with sensitive data to adopt cloud storage solutions.

1.3.2 This is a challenging and unsolved problem which needs addressing

The Data Location Assurance (DLA) Problem in geo-replication of data setting is a challenging problem. Coming up with a solution for this data location problem is more difficult

than that of a data privacy solution. For many system models, Encrypt-then-store (i.e., the user encrypts the data before storing it into the cloud) is a very well-known technique where a user can still guarantee the privacy protection of its data even if the CSP is considered to be malicious. However, in most of the cases, the encrypt-then-store solution is not applicable and does not help to solve the data location assurance problem since here we are concerned about the geo-location of the data, not the privacy guarantee of the data. Most likely, the laws and acts mentioned in the previous section (in regards to the geo restrictions on the sensitive data) might not permit to keep the sensitive data outside the geographic boundaries of a particular country even if it is encrypted. In addition, some countries' jurisdictions do not allow encrypted data to be kept. Furthermore, the geo-replication of data practice among the CSP makes it more difficult to come up with a solution for the DLA problem as now a piece of data can be stored in more than one geographic location. So any attempts based on estimating the time required for getting the response from the storage server will **only** help to predict the location of **one** particular storage server that is sending the response [see section 2.3 and 2.5].

Proof of Data Possession (where a user is able to get a verifiable data possession proof from the un-trusted storage server) scheme seems to be a good candidate for devising a solution for the DLA problem. Note that, the Proof of Data Possession (PDP) scheme was originally developed for use with a single un-trusted storage server, whereas in a cloud storage environment there would be more than one storage server and they can communicate with each other. As a result, when using **only** a PDP scheme, it is not possible to build a solution for the DLA problem because for the DLA problem, we want a data possession guarantee from the storage servers (or datacenters) rather than from the cloud storage environment as a whole. Aside from the related work section (see section 2.5), we provide more details about the limitation of the existing PDP schemes for the cloud storage environment in section 5.3.

The DLA problem is identified as an important problem by many researchers [10] [11] [12] [13] but has not yet been solved. There are a few scattered attempts for solving the DLA problem but all have limitations (see section 2.5). Most importantly, no existing work addresses the standard geo-replication practice of data among CSPs (Cloud Storage Providers) and thus the DLA problem in case of geo-replication of data setting is an unsolved challenging problem. Furthermore, many of those existing solutions are based on geo-location techniques

(e.g., ping-based solutions) which are not very accurate. Last but not least, through those existing solutions, a client will not receive verifiable location guarantees from the CSP and might not be convinced to adopt a cloud storage solution. We will review the existing solutions in section 2.5.

1.4 Objectives

The main objective of our research is to solve the very challenging and unsolved Data Location Assurance (DLA) problem of the cloud storage environments in geo-replication of data setting. We want to come up with efficient solutions for the data location assurance problem of cloud storage environments. These solutions must not only work in geo-replication of data setting but also allow users to receive verifiable location guarantees about their data from the CSP. We name the solution for the DLA problem as DLAS (Data Location Assurance Service) solution.

1.5 Research Methodology

For CSPs that have total physical control over the location of data, it is very challenging to come up with a DLAS solution for cloud storage environments. In fact, it is almost impossible to come up with a DLAS solution if the CSP is totally dishonest. In case of a totally dishonest model, CSP does not need to correctly follow the protocol execution. So it can always lie about the location of a piece of data when data is supposed to be stored in a datacenter located in Region X but is actually kept in a datacenter located in Region Y. In this case, cryptographic primitives deployed on the Cloud, would not be of much help either. However for the sake of their business reputation, existing CSPs do not follow the totally dishonest server model (i.e., totally malicious) on a permanent basis. Rather in most cases they will follow the protocol correctly. In the worst case, they might become malicious for a small amount of time to generate a false data possession proof. Most of the existing security solutions for cloud storage environments are based on the honest-but-curious attack model [24] [25] [26] [27] [28].

To solve the DLA problem, we first research existing cloud storage scenarios (especially those requiring the data location guarantees) as well as existing CSPs and then identify

and define all possible system and attack models which should essentially cover most of the existing cloud storage applications requiring the geo-location guarantees for their data. Finally, we propose DLAS solutions for each identified system and attack model. Our strategy is to first propose a DLAS solution that works for a comparatively restrictive system and attack model (i.e., less adversarial system and attack model) and then progressively look for DLAS solutions for more adversarial ones.

Note that, during the process of devising DLAS solutions for different system and attack models we simplify the problem by making a number of assumptions. In real life, these would need to be extended in various ways.

1.6 Publications

So far from this thesis work, three publications have been already contributed to the literature. They are the following:

1. Ali Noman, Carlisle Adams, "Providing A Data Location Assurance Service for Cloud Storage Environments", pp. 265-286, Vol.8 No.4 June 20, 2013. Journal of Mobile Multimedia (special issue on cloud computing).

2. Ali Noman, Carlisle Adams, "Hardware-Based DLAS: Achieving Geo-location Guarantees for Cloud Data using TPM and Provable Data Possession", pp. 280-285,17th Int'l Conf. on Computer and Information Technology, 22-23 December 2014(IEEE).

3. Ali Noman, Carlisle Adams, "DLAS: Data Location Assurance Service for Cloud Computing Environments". In proc 10th international conference on Privacy Security and Trust (PST 2012) in France, 2012(IEEE).

These works will be discussed in the following chapters where we discuss our proposed solutions.

1.7 Contributions

Our contributions will focus mainly on finding solutions for the data location assurance problem in the geo-replication of data setting. More precisely, they include the following:

- Determining necessary system and attack models for the DLAS solutions by studying the existing cloud storage applications (especially those requiring data location guarantees; e.g. organizations that deal with heath related data, financial data, and so on) and the existing CSPs.
- Proposing DLAS solutions for various system and attack models- progressively going from a comparatively less adversarial system and attack model to more adversarial ones. In this thesis work, we define 4 system models and 3 attack models and propose 4 DLAS solutions for those system models and attack models.
- Identifying the limitation of existing Proof of Data Possession (PDP) solutions in building any DLAS solution for the cloud storage environment and proposing a PDP scheme specifically for the cloud storage environment. This PDP scheme, independently, can be considered as one of the main contributions of this Ph.D. research.
- Use the above PDP scheme as the main building block for building the DLAS solution for a more adversarial attack model. This DLAS solution has the ability to meet the need for all cloud storage applications requiring the data location guarantees.
- Further improvement on our proposed PDP and DLAS solutions by incorporating *public verifiability* and *dynamic operation on data* capabilities. Whereas public verifiability is able to facilitate an entity (other than the data owner) with proper credentials, to execute our proposed DLAS solution (also true for PDP solution) and verify the data location guarantee, dynamic operation on data, enables our DLAS solution to deal with dynamic data (where a data owner can make some changes (e.g., update, modify, delete, and so on) on their stored data). Public verifiability is particularly necessary for the health-related data storage scenario where many people have to share the same data.
- We also provide a rough sketch of an ideal DLAS solution that can offer optimal security guarantees. The main purpose of this solution is to identify the key properties/characteristics of a DLAS solution with optimal security guarantees. This solution has the potential to be used as the basis for any future research attempt of devising a more efficient DLAS solution with optimal security guarantees.

1.8 Outline

The rest of this thesis is organized as follows.

In chapter 2, we provide a background necessary to understand this thesis work which includes discussion on cloud storage environments, important cryptographic primitives related to this thesis, and related work. We conclude chapter 2 by pointing out the solution requirement and strategy we set for our proposed solution.

Chapter 3 includes a complete description of our first proposed solution for the DLA problem, which we name as DLAS.

In chapter 4, we discuss in detail our second proposed solution, HDLAS where we considered a comparatively more adversarial system and attack model than the DLAS solution.

Chapter 5 includes one of the most significant contributions of this thesis where we propose the first Proof of Data Possession solution for the cloud storage environments. Here we also point out how a solution for the DLA problem can easily be built using this PDP solution.

In chapter 6, we propose some further enhancements of our proposed PDP and DLAS solutions which include but are not limited to adding capabilities such as public verifiability and dynamic operation on data. In order to facilitate future research in this domain, here we also provide a rough sketch of an ideal PDP solution to identify the key requirements necessary for achieving the optimal security guarantees.

We conclude our thesis work in chapter 7 with an indication of the future work.

2 Foundations

In this chapter, we try to cover the background required by a reader to follow our research work. We first talk in detail about cloud computing, keeping cloud storage scenarios in mind; here we also discuss how existing cloud storage providers such as Amazon S3, Microsoft Azure and Google Cloud Storage work. Second, we provide an overview of some important cryptographic primitives which are an integral part of this thesis work. Third, we discuss existing significant research attempts related to our thesis work. We finally conclude this chapter by discussing the solution requirement and the strategy of our proposed solutions.

2.1 Cloud Computing

2.1.1 Cloud Architecture

Whereas NIST's definition of cloud computing, available in Appendix 2, summarizes all key aspects of cloud computing, here we reiterate some aspects of it to explain the cloud architecture. Deployment models and service delivery models are the two key aspects of cloud architecture.

The **deployment models** are: (1) Private cloud: a cloud platform is dedicated to a specific organization, (2) Community Cloud: similar to private cloud; but is dedicated to a community sharing the same interest, (3) Public cloud: a cloud platform available to public users to register and use the available infrastructure, and (4) Hybrid cloud: a combination of two or more cloud infrastructures from the above three. Public clouds are the most vulnerable deployment model because they are available for public users to host their services and these users may be malicious.

The cloud service **delivery models** include:

Infrastructure-as-a-service (IaaS): where cloud providers deliver computation resources, storage, and network as internet-based services. It is based on a virtualization technology. Amazon EC2 is the most familiar IaaS provider.

Platform-as-a-service (PaaS): where cloud providers deliver platforms, tools and other business services that enable customers to develop, deploy, and manage their own applications, without installing any of these platforms or support tools on their local machines. The PaaS model may be hosted on top of an IaaS model or on top of the cloud infrastructures directly. Google Apps is an example of PaaS.

Software-as-a-service (SaaS): where cloud providers deliver applications hosted on the cloud infrastructure as an internet-based service for end users, without requiring installing the applications on the customers' computers. This model may be hosted on top of PaaS, IaaS or directly hosted on the cloud infrastructure. SalesForce CRM (Customer Relationship Management) is an example of a SaaS provider. Moreover, these service delivery models may coexist in one cloud platform, leading to further complication of the security management process.

Cloud Federation: It is relatively a new term, which defines the practice of interconnecting public and private cloud services from two or more providers to extend the scalability of existing cloud systems, increase elasticity in resource management, and accommodate spikes in demand [29]. Since workloads are outsourced to cost-effective regions, cloud federation can also potentially result in significant cost savings (electricity, hardware provisioning, and so on). This outsourcing also facilitates federated clouds to overcome emerging challenges, such as big data handling/disaster recovery through colocation and geographic distribution of computing and storage resources, together with other methods. Therefore, it isn't surprising that the popularity of cloud federation is increasing day by day. A Gartner study, for example, forecasts that through 2020, hybrid clouds will overtake both public and private cloud services to be the dominant cloud service model [29].

2.1.2 Overview of Amazon S3 (Simple Storage Service)

As an instance of existing Cloud Storage Providers, here we talk about Amazon S3. Amazon S3 offers a simple web services interface that you can use to store and retrieve any amount of data, at any time, from anywhere on the web. It is intentionally built with a minimal feature set that focuses on simplicity and robustness. Here we provide a summary of Amazon S3 taken from [7].

2.1.2.1 How data storage works in Amazon S3

Storing data in Amazon S3 starts with creating and naming a "**bucket**" that stores data. **Buckets** are the fundamental container in Amazon S3 for data storage. You can store an unlimited amount of data in a bucket. Buckets serve several purposes: they organize the Amazon S3 namespace at the highest level, they identify the account responsible for storage and data transfer charges, they play a role in access control, and they serve as the unit of aggregation for usage reporting.

Data stored in a bucket is denoted as "**object**". In other words, a bucket is a container for **objects** stored in Amazon S3 and every object is contained in a bucket. According to Amazon S3, you can store an infinite number of objects in a bucket; each object can contain up to 5 TB of data. An object is comprised of data and metadata where the data portion is opaque to Amazon S3. The metadata is a set of name-value pairs that describe the object. You can choose metadata from the default metadata list such as the date last modified, and standard HTTP metadata, such as Content-Type; in addition you can also specify your own custom metadata where you are storing the object.

A **key** is a unique identifier for an object within a bucket. Every object in a bucket has exactly one key. Because the combination of a bucket, key, and version ID uniquely identify each object, Amazon S3 can be thought of as a basic data map between "bucket + key + version" and the object itself. Every object in Amazon S3 can be uniquely addressed through the combination of the web service endpoint, bucket name, key, and optionally, a version. For example, in the URL http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl, "doc" is the name of the bucket and "2006-03-01/AmazonS3.wsdl" is the key.

For **data access**, Amazon S3 offers both REST and SOAP interfaces which are designed to work with any internet-development toolkit. It facilitates downloading your data any time you like or allows others to do the same. "Permissions" is one of the key feature sets of Amazon S3, which facilities granting or denying access to others who want to upload or download data into your bucket.

2.1.2.2 Regions/ Data Center's Location

Amazon S3 facilitates you to choose the geographical region where Amazon S3 will store the buckets you create. As a client, you might choose a region in order to optimize latency, minimize costs, or address regulatory requirements. Amazon S3 currently supports 14 regions: 5 located in North America (among them, 1 is in Canada), 1 in South America, 5 in Asia Pacific, and 3 in EU [7]:

2.1.2.3 Storage Classes

Amazon S3 defines 4 storage classes for the object you want to store in your bucket. They are: STANDARD, STANDARD_IA, REDUCED_REDUNDANCY, and GLACIER [7]**.** One has to choose from one of those 4 storage classes depending on the use case scenario and performance access requirements. Note that all of these storage classes offer high durability.

STANDARD is the default storage class which is ideal for performance-sensitive use cases and frequently accessed data. STANDARD_IA (where IA stands for infrequent access) is optimized for long-lived but less frequently accessed data, for example backups and older data where the frequency of access has reduced, but the use case still demands high performance. Note that the STANDARD_IA objects are available for real-time access. REDUCED_REDUNDANCY storage (RRS) class is designed for noncritical, reproducible data stored at lower levels of redundancy than the STANDARD storage class, which reduces storage costs. The GLACIER storage class is suitable for archiving data where data access is infrequent. Archived objects are not available for real-time access. In contrast to STANDARD, RRS, and STANDARD_IA objects, Restoration of the GLACIER object is required before it can be accessed. Note that it is not possible to create an object as a GLACIER object; it can be created by first uploading objects using STANDARD, RRS, or STANDARD_IA as the storage class and then converting it into a GLACIER object.

2.1.2.4 Geo-replication of data

By default, Amazon S3 stores clients' data across multiple geographically distant availability zones [7]. However, Amazon S3 also offers another service called **cross-region replication**. Cross-region replication is a bucket-level configuration that enables automatic, asynchronous copying of objects across buckets in different AWS Regions. You, as a client, can configure which objects will be replicated and in which AWS regions. If needed, it is also possible for you to instruct Amazon S3 to change replica ownership to the AWS account that owns the destination bucket regardless of who owns the source bucket or the source object. Note that currently, a client can replicate objects from a source bucket to only one destination bucket.

2.1.3 Overview of Microsoft Azure Storage

Microsoft Azure Storage (MAS), previously known as Windows Azure Storage [30], is a Microsoft-managed cloud storage service which is highly available, secure, durable, scalable, and redundant. Similar to Amazon S3, MAS customers have access to their data from anywhere at any time and only pay for what they use and store.

Whereas all the details about Microsoft Azure storage are available in [8], here we provide a brief summary of MAS:

2.1.3.1 How Microsoft Azure Storage Works

Microsoft Azure Storage consists of three data services and two types of storage accounts. *Blob storage, File storage, and Queue storage* are the three data services offered by Azure Storage; *standard* and *premium* storage are the available storage account types. In order to use any of the data services provided by Azure storage (i.e., Blob storage, File storage, and Queue storage), you need to first create a storage account, and then you can transfer data to/from a specific service in that storage account. Similar to Amazon S3, after storing the files in Azure storage, you can access them from anywhere in the world using URLs, the REST interface and one of the Azure SDK storage client libraries.

Azure Data Services: Blobs are basically files (e.g., pictures, Microsoft Excel files, HTML files, virtual hard disks, big data such as logs, database backups) that you typically store on your computer (or tablet, mobile device, and so on). Blobs are stored in containers, which are similar to folders. There are three types of blobs: block blobs, page blobs (used for virtual hard disks files), and append blobs. Whereas block blobs are used to hold ordinary files up to about 4.7 TB, page blobs are used to hold random access files up to 8 TB in size. Note that append blobs are made up of the block blobs, but are optimized for append operations.

Azure File Storage is used for setting up highly available network file shares that can be accessed by using the standard Server Message Block (SMB) protocol. It facilitates multiple Virtual Machines (VMs) to share the same files with both read and write access. In addition, it is also possible to access those files using the REST interface or the storage client libraries. One thing that distinguishes Azure Files from files on a corporate file share is that you can access the files from anywhere in the world using a URL that points to the file and includes a shared access signature (SAS) token. Further details on Azure File Storage is available in [8].

Finally, **the Azure Queue service** is used to store and retrieve messages. Queue messages can be up to 64 KB in size, and a queue can contain millions of messages. Queues are generally used to store lists of messages to be processed asynchronously.

Types of Storage Accounts: The most widely used storage accounts are **standard storage accounts**, which can be used for all types of data. It uses magnetic media to store data. On the other hand, **premium storage account** provides high-performance storage for page blobs, which are primarily used for VHD files and it uses SSD to store data.

Apart from this, there is a specialized storage account called "**Blob Storage account**" which is used for storing block blobs and append blobs. You can't store page blobs in these accounts, therefore you can't store VHD files. These accounts allow you to set an access tier to *Hot* or *Cool* and this access tier can be changed at any time. The *hot access tier* is used for files that are accessed frequently where storage cost is higher but the cost of accessing the blobs is much lower. In contrast, for blobs stored in the *cool access tier*, you pay a higher cost for accessing the blobs, but the cost of storage is much lower.

2.1.3.2 Region

At present. Microsoft Azure is available in 36 regions around the world, with plans announced for 6 additional regions. [8] includes the complete list of all regions. Among those 36 regions, only US East2 and West Europe support "*availability zones*". The main purpose of "availability zones" is to help to protect its customers from datacenter-level failures. They are located inside an Azure region, and each one has its own independent power source, network, and cooling. To ensure resiliency, there's a minimum of three separate zones in all enabled regions. The physical and logical separation of availability zones within a region protects applications and data from zone-level failures.

Microsoft Azure puts a high priority on geographic expansion to enable higher performance, and to support its customers' requirements and preferences regarding data location.

2.1.3.3 Replication of Data

In Microsoft Azure Storage, data is stored using both local and geographic replication to facilitate disaster recovery. In order to ensure that your data is durable, Azure Storage has the ability to keep (and manage) multiple copies of your data. This is called replication, or sometimes redundancy. A client selects a replication type while setting up their storage account. In most cases, this setting can be modified after the storage account is set up. Here we summarize all available replication types.

All storage accounts have **locally redundant storage (LRS)**. In the past, it was defined as "intra-stamp replication" [30]. This means three copies of your data are managed by Azure Storage in the data center specified when the storage account was set up. When changes are committed to one copy, the other two copies are updated before returning success. This means the three replicas are always in sync. Note that these three copies reside in separate fault domains and upgrade domains, which means your data is available even if a storage node holding your data fails or is taken offline to be updated.

Zone-redundant storage (ZRS) is the 2nd replication type which maintains the three local copies of your data (i.e., LRS) as well as another set of three copies of your data. The second set of three copies is replicated asynchronously across datacenters within one or two regions. Note that ZRS is only available for block blobs in general-purpose storage accounts. Also, once you have created your storage account and selected ZRS, you cannot convert it to use any other type of replication, or vice versa. ZRS accounts provide higher durability than LRS, but ZRS accounts do not have metrics or logging capability.

Geo-redundant storage (GRS) is another replication type available in Microsoft Azure Storage which maintains the three local copies of your data (i.e., LRS) in a primary region plus another set of three copies of your data in a secondary region hundreds of miles away from the primary region. In the event of a failure at the primary region, Azure Storage will fail over to the secondary region.

Read-access geo-redundant storage is a variant of GRS. It is similar to GRS except that you get read access to the data in the secondary location. If the primary data center becomes unavailable temporarily, you can continue to read the data from the secondary location. This can be very helpful. For example, you could have a web application that changes into read-only mode and points to the secondary copy, allowing some access even though updates are not available.

2.1.4 Overview of Google Cloud Storage

In general, Google Cloud Storage can provide all the storage services / features available in Amazon S3 and Microsoft Azure.; however, their underlying architecture might be different.

Whereas all the details about Google Cloud Storage is available in [9], here we provide a brief summary.

2.1.4.1 How Google Cloud Storage Works

Google Cloud Storage works like this. All data in a Google Cloud Storage belongs to a **project**. A project consists of a set of users, a set of APIs, and billing, authentication, and monitoring settings for those APIs. A cloud user can have one project or multiple projects.

Similar to Amazon S3**,** here **Buckets** are also the basic containers that hold data. Everything that a cloud user wants to store in a Cloud Storage, must be kept in a bucket. A cloud user can use buckets to organize data and control access to data, but unlike directories and folders, the creation of nested buckets is not possible.

In order to create a bucket, a cloud user has to specify a globally-unique name, a *geographic location* where the bucket and its contents are stored, and a *default storage class*. The default storage class you choose applies to objects added to the bucket that don't have a storage class specified explicitly. After a bucket is created, it is still possible to change its default storage class, to any class supported in the bucket's location; however, the only way of changing the bucket name and location is through deleting and re-creating the bucket.

Google Cloud Storage offers four storage classes: *Multi-Regional Storage*, *Regional Storage*, *Nearline Storage*, and *Coldline Storage*. All storage classes offer the same throughput, low latency (time to first byte typically tens of milliseconds), and high durability. These classes differ by their availability, minimum storage durations, and pricing for storage and access. *Multi-Regional Storage* is suitable for storing data that is frequently accessed ("hot" objects) around the world, such as serving website content, streaming videos, or gaming and mobile applications. On the other hand, *Regional Storage* is also suitable for storing frequently accessed data where data would be mainly accessed from the same region. In this case, data is redundant across availability zones, not across different regions. Nearline Storage and Coldline Storage are suitable for less frequently accessed data. Nearline Storage is ideal for back-up and serving long-tail multimedia content which can be accessed for instance no more than once in a month. Coldline Storage is typically intended for disaster recovery, or data that is archived and may or may not be needed at some future time.

Objects are the individual pieces of data that is stored in Cloud Storage. There is no limit on the number of objects that can be created in a single bucket. An object has two components: *object data* and *object metadata*. **Object data** is typically a file that a cloud user wants to store in Cloud Storage. **Object metadata** is a collection of name-value pairs that describe various object qualities. An object's name is treated as a piece of object metadata in Cloud

Storage. Object names can contain any combination of Unicode characters (UTF-8 encoded) and must be less than 1024 bytes in length.

2.1.4.2 Geo-replication of Data

Similar to Amazon S3 and Microsoft Azure, Google Cloud Storage also facilitates geo replication of data. In this case, this service is provided under the name of "multi-regional storage class".

As we mentioned earlier, a cloud user has to specify a location for storing data during the creation of a bucket. There are 2 types of locations: *regional location and multi-regional location*. A *regional location* is a specific geographic place, such as London. On the other hand, a *multi-regional location* is a large geographic area, such as the United States, that contains at least two geographic places. These places are a combination of regional locations and additional Google data centers

It is important to note that certain storage classes can only be used in a certain type of location. For example, regional storage data must be stored in a regional location such as us-east1 and multi-regional storage data must be stored in a multi-regional location such as EU. Google Cloud Storage stores individual Multi-Regional Storage objects in at least two separate geographic places within the selected location.

At present, there are 15 regions and 3 multi-regions in Google Cloud Storage [9].

2.2 Trusted Platform Module (TPM)

The TPM is a security specification defined by the Trusted Computing Group [31]. Its implementation is available as a chip that is physically attached to a platform's motherboard and controlled by software running on the system using well-defined commands [32]. The TPM is manufactured with a public/private key pair built into the hardware, called the endorsement key (EK). The EK is unique to a particular TPM and is signed by a trusted Certification

Authority (CA). It also provides secure storage for small amounts of information such as cryptographic keys. Because the TPM is implemented in hardware and presents a carefully designed interface, it is believed to be resistant to software attacks [32]. TPM provides cryptographic operations such as asymmetric key generation, decryption, encryption, hashing, Hmac computation, signing and migration of keys between TPMs, as well as random number generation and hashing. Most importantly, the reason behind using TPM in our proposed solution is that TPM has the capability to do integrity measurement and remote attestation.

Integrity Measurement: It is a process by which information about the software, hardware, and configuration of a system is collected and digested. At load-time, the TPM uses a hash function to fingerprint an executable, an executable plus its input data, or a sequence of such files. These hash values are used in attestation to reliably establish code identity to remote or local verifiers. The hash values can also be used in conjunction with the sealed storage feature. A secret can be sealed along with a list of hash values of programs that are allowed to unseal the secret. This allows the creation of data files that can only be opened by specific applications.

Attestation: It is a mechanism that a software can use to prove its identity. The goal of attestation is to prove to a remote party that your operating system and application software are intact and trustworthy. The verifier trusts that attestation data is accurate because it is signed by a TPM whose key is certified by the Certificate Authority (CA).

2.2.1 TPM 1.2 vs TPM 2.0

Whereas the complete details about TPM 1.2 and TPM 2.0 specification can be found in [31] [33], here we briefly discuss the new features introduced into TPM 2.0. We mainly focus on their cryptographic capabilities. In terms of the cryptographic support, Elliptic Curve Cryptography and SHA-256 are the significant additions into TPM 2.0 which are not present in TPM 1.2.

The following table includes a comparison between TPM 1.2 and TPM 2.0 in regards to their cryptographic support.

Table 1 Cryptographic Support Comparison between TPM 1.2 and TPM 2.0

Algorithm Type	Algorithm Name	TPM 1.2	TPM 2.0
Asymmetric	RSA 1024	Yes	Optional
	RSA 2048	Yes	Yes
	ECC P256	No	Yes
	ECC BN256	No	Yes
Symmetric	AES 128	Optional	Yes
	AES 256	Optional	Optional
Hash	SHA-1	Yes	Yes
	SHA-2 256	No	Yes
HMAC	SHA-1	Yes	Yes
	SHA-2 256	No	Yes

It is important to note that TPM 2.0 is not backward compatible with TPM 1.2. There are significate behavioral and architectural differences (e.g., how the authorization works?) between these 2 TPM specifications. An interested reader can look into the official website of the Trusted Computing Group [31] for any further information on TPM 1.2 and TPM 2.0.

2.3 Internet Host Geolocation

Geolocation of Internet hosts facilitates delivery of a diverse and interesting new class of location-aware applications. Examples of such location-aware applications include but are not limited to targeted advertising on web pages, automatic selection of a language to display content, restricted content delivery following regional policies, and authorization of transactions only when performed from pre-established locations. Inferring the location of Internet hosts from their IP addresses is a challenging problem because there is no direct relationship between

the IP address of a host and its geographic location. In general, the measurement-based geolocation of Internet hosts [34] uses the positions of landmarks, reference hosts with well-known geographic locations, as the possible location estimates for the target host. This leads to a discrete space of answers, i.e. the number of answers is equal to the number of reference hosts; this can be inaccurate because the closest reference host may still be far from the target. The Constraint-Based Geolocation (CBG) approach is a different approach which infers the geographic location of Internet hosts using multilateration; it tries to overcome the problem with the measurement based technique for geolocation of internet hosts mentioned above. Multilateration refers to the process of estimating a position using a sufficient number of distances to some known fixed points. As a result, multilateration establishes a continuous space of answers instead of a discrete one. It uses a set of landmarks to estimate the location of other Internet hosts. The fundamental idea is that given geographic distances to a given target host from the landmarks, an estimation of the location of the target host would be feasible using multilateration, just like the Global Positioning System (GPS). The details of CBG are included in [34]. [35], [19], [36], and [37] are some of the recent proposals for geo-location techniques.

2.4 Relevant Cryptographic Primitives

In order to solve the Data Location Assurance (DLA) problem, we considered a wide range of cryptographic primitives including some advanced cryptographic schemes such as searchable encryption, Pairing-based cryptography, multi-party computation, different forms of Attribute-Based Encryption (ABE) and commitment schemes, proxy re-encryption and so on. However, here we discuss only those cryptographic primitives that we use in devising our proposed solutions.

2.4.1 Zero Knowledge Sets

Zero Knowledge Sets (ZKS) is a cryptographic primitive which facilitates a polynomial time Prover to commit to an arbitrary finite set S of strings so that, later on, for any string x, he can prove whether $x \in S \; or \; x \notin S$, without providing any further information about S; not even the size of S [38]. Whereas [38] is the first proposal (and well-known as well) for ZKS, there have been other ZKS schemes (e.g., [39], [40]) which try to do further improvement (e.g., [39] reducing the size of the proof) on the original. In devising one of our proposed

solutions, we use the original ZKS scheme proposed in [38]. Since zero knowledge set is a special case of an elementary database (EDB), it has been explained in terms of an elementary database.

Definition of Elementary Database (EDB): An EDB, DB, is a subset of {0, 1}* × {0, 1}* such that if (x, v) ∈ DB and (x, v') ∈ DB then v = v'. By [DB] we denote the support of DB, i.e., the set of $x \in \{0,1\}^*$ for which ∃v such that$(x, v) \in DB$. We denote such unique v as DB(x); if x ∉ [DB] then we also write DB(x) = ⊥ [38].

A ZKS protocol is a non-interactive protocol which relies on a public random string σ, the reference string. This reference string is polynomially long in K, the security parameter controlling the probability of error or successful cheating.

Definition of ZKS: A ZKS protocol has three algorithms: Commit_ZKS (DB, σ, K), Proof_ZKS (DB, K, σ, PK, SK) and Verify_ZKS (x, π_x, K, σ, PK) and two parties- a Prover (also known as the committer) and a Verifier. Whereas the Prover is responsible to execute the Commit_ZKS and Proof_ZKS algorithm, the job of Verifier is to perform the Verify_ZKS algorithm. Here $PK, SK \ and \ \pi_x$ represent public key (i.e., the commitment of DB) and private key, and DB's proof about the value, x, respectively [38].

In our proposed solution, DLAS (see section 3.2), the CSP and the Enterprise (E) play the role of the Prover and the Verifier respectively.

A ZKS protocol must satisfy completeness, soundness and zero-knowledge. Informally, completeness means that if DB(x) = y, the Prover should always be able to convince the Verifier of this fact. Soundness means that no efficient Prover should be able to produce a commitment to DB, a value x and two proofs π_x , π^1_x that convince the Verifier of two distinct values for DB(x). Finally, zero-knowledge means that an efficient verifier learns only the values DB(x) from his interaction with the Prover, and nothing else (not even the size of DB). In particular, the Verifier does not learn the values DB(x') for an x' not queried to the Prover.

Note that many cryptographic primitives such as Pedersen commitment scheme and collision resistant hash functions have been used in the zero knowledge set protocol. [38] provides a full description of the original ZKS protocol.

2.4.2 CP-ABE

A Ciphertext Policy-Attribute-Based Encryption (CP-ABE) scheme [41] is one kind of Attribute-Based Encryption where a user's private key is associated with an arbitrary number of attributes expressed as strings; when an encryptor encrypts a message, he specifies an associated access structure (also known as an access tree) over attributes. A user will only be able to decrypt a ciphertext if that user's attributes pass through the ciphertext's access structure. At a mathematical level, access structures used in this scheme are described by a monotonic "access tree", where nodes of the access structure are either threshold gates or leaves which represent attributes. Note that AND gates can be constructed as n-of-n threshold gates and OR gates as 1-of-n threshold gates. Figure 1 shows an example of a simple access tree.

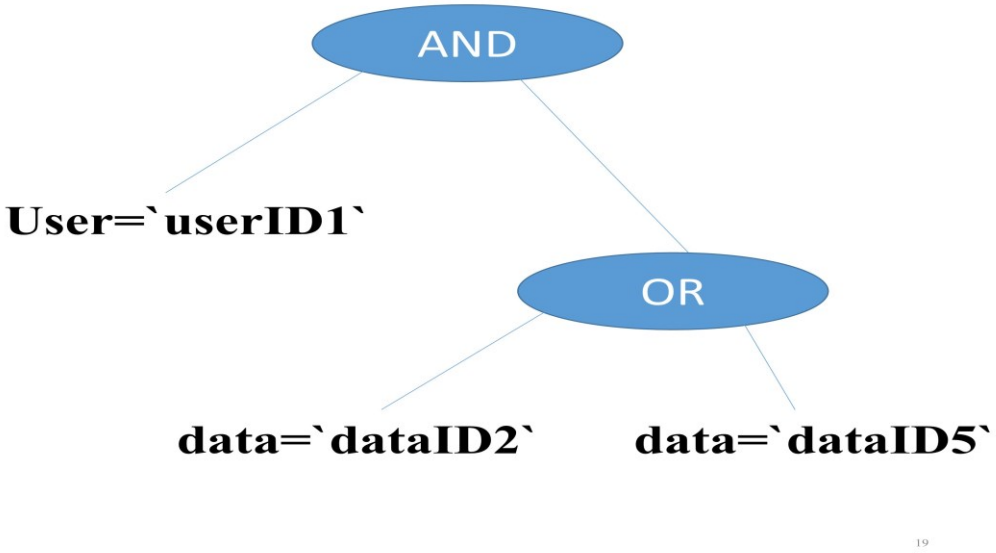

Figure 1 A simple example of access tree which allows the access if the user is userID1 and the data is either dataID2 or dataID5

Definition of CP-ABE : A CP-ABE scheme consists of four fundamental algorithms: Setup_CPABE(Δ), Encrypt_CPABE(PK, M, A) , KeyGen_CPABE (MK, S) and Decrypt_CPABE (PK,CT, Credential) where Δ, PK, M, MK, A, CT, S, and Credential represent an implicit security parameter, public key, plain text, master key, an access structure, the corresponding ciphertext of M, a set of attributes (used in KeyGen_CPABE() algorithm to generate the private key, Credential) and private key.

Note that the CP-ABE encryption scheme is based on the bilinear map. The complete description of the CP-ABE scheme can be found in [41]. At present, an implementation of the CP-ABE scheme is also publicly available in [42].

2.4.3 Proof of Data Possession

A proof of data possession (PDP) is a cryptographic scheme executed between a client and a server where the server can prove to the client that it did not tamper with its data. There have been a number of PDP schemes proposed recently [see Table 2]. However, the objective of all the PDP schemes as well as the working pattern is almost same. The PDP scheme in general works as follows. Generally, in a PDP scheme, data (often represented as a file F) are preprocessed by the client, and verification meta-data are also produced in advance for future verification purposes. The file along with the verification metadata is then sent to an untrusted server for storage, and the client may delete the local copy of the file. The client might need to keep some (possibly secret) information to check server's responses later. The server proves the data have not been tampered with by responding to challenges sent by the client. This PDP scheme provides probabilistic guarantees of data possession, where the client checks a random subset of stored blocks with each challenge. Further details on PDP schemes and solutions of this kind can be also found in section 2.5 and section 5.3.

2.4.4 Proof of Retrievability

POR is complementary to PDP. A proof of storage (PDP) is a protocol executed between a client and a server where the server can prove to the client that it did not tamper with its data. On the other hand, a proof of retrievability (POR) is a cryptographic primitive, which in addition with the data integrity guarantee (offered by the server in a PDP protocol), can also guarantee that the client can retrieve the data. Various POR schemes can be found in the literature, for example [43] [44] [45] [46] [47] [48]. For discussion, we choose the POR scheme, PORSYS, proposed in [43] as this is probably the most well-known one and it includes a formal definition of the POR scheme. PORSYS has six algorithms (keygen, encode, extract, challenge, respond and verify) among which the respond algorithm will be executed by the prover

and the other five algorithms will be executed by the verifier. The formal definition of PORSYS is given in [43].

2.4.5 Pairing-based Cryptography: Bilinear Pairing

Pairing-based cryptography in general and Bilinear pairing in particular, has been used in constructing a number of cryptosystems such as identity based encryption, aggregate signatures, different variants of attribute-based encryption, and so on. Whereas [49] includes an excellent introduction of bilinear pairing, [50] includes a popular survey on Pairing-based Cryptography. Figure 2 includes a basic definition of bilinear pairing from [49].

2.4.6 Authenticated Encryption (AE)

Authenticated encryption (AE) schemes is a symmetric-key mechanism which transforms a message, M, into a ciphertext, C, in such a way that C protects both privacy and authenticity [51]. Authentication Encryption with Associated Data (AEAD) is a variant of AE, which considers the fact that many applications come with a mixture of secret and non-secret data, and thus it offers privacy for only the secret data and authenticity for both secret and non-secret data. The non-secret data is called the associated data or the header.

Let n be a prime number. Let $G_1 =< P >$ be an additively-written group of order n with identity ∞, and let G_T be a multiplicatively-written group of order n with identity 1.

Definition of Bilinear Pairing: A bilinear pairing on (G_1, G_T) is a map

$$\hat{e} : G_1 \times G_1 \rightarrow G_T$$

that satisfies the following conditions:

(1) (bilinearity) For all $R, S, T \in G_1$, $\hat{e}(R+S, T) = \hat{e}(R, T)\hat{e}(S, T)$ and $\hat{e}(R, S+T) = \hat{e}(R, T)\hat{e}(S, T)$.

(2) (non-degeneracy) $\hat{e}(P, P) \neq 1$.

(3) (computability) $\hat{e}$ can be efficiently computed.

Note that the security of many pairing-based protocols is dependent on the intractability of the following problem:

"Let $\hat{e}$ be a bilinear pairing on (G_1, G_T). The bilinear Diffie-Hellman problem (BDHP) is the following:

Given P, aP, bP, cP, compute $\hat{e}(P, P)^{abc}$."

Figure 2 Definition of Bilinear Pairing

Note that the need for AE emerged from two factors: (i) it had been identified that in many cases, people had been doing rather poorly when they tried to glue together a traditional (privacy-only) encryption scheme and a message authentication code (MAC) [52] [53]; (ii) a number of AE schemes [54] [55] did not work when they tried to glue together an encryption scheme and a MAC [51].

At present, even though a number of AE and AEAD schemes are available (such as CCM, EAX, Encrypt-then-MAC (EtM), and GCM), only a few of them is patent free [51]. The Galois/Counter Mode (GCM) of operation for block ciphers seems to be one of the most widely used patent free AEAD scheme [56]. It is also included in the NIST's recommended list for AE/AEAD schemes. The security notions of AE and AEAD has been discussed in [57].

2.4.7 Well known Cryptographic primitives

In our proposed solutions, we also use well-known cryptographic primitives such as cryptographic hash function, pseudo-random function, pseudo-random permutation, public key encryption, symmetric key encryption, Message Authentication Code (MAC), HMAC, digital signature and elliptic curve crypto. See [58] for definitions of these primitives.

2.5 Related Work

2.5.1 Significance of a solution for the Data Location problem & its associated challenges

The data location assurance problem (i.e., our defined DLA problem) has been identified as one of the main security issues in cloud computing by many researchers [10] [11] [12] [16] [29]. Research conducted by Gartner also suggests that cloud users' data location preference should be included in the Service Level Agreement (SLA) [12]. They also recommend that users should continuously check whether their data is still located in the appropriate jurisdiction.

Some existing cloud storage providers such as Amazon S3 and Microsoft Azure now include data location preferences in the SLA [7] [8]. Handling sensitive outsourced data on the cloud is very critical because access to this data is generally governed by various legal restrictions, such as the Health Insurance Portability and Accountability Act (HIPAA) and Control Objectives for Information and Related Technology (COBIT) [29]. For example, the Canadian Personal Information Protection and Electronic Documents Act [23] requires that any PII related data has to be stored within Canada. Furthermore, restrictions on data storage and access can also vary among states within the same country, or between countries. For example within the EU, countries such as France and Denmark have broad restrictions, but countries such as Italy and Germany have limited or no restrictions on certain types of data.

[22] claims that along with the proliferation of outsourcing data into the cloud, customers and law makers are now adding more items to their compliance requirement list for outsourced data. It includes, but is not limited to, requirements in regards to restricted storage locations, storage duration, full-disk encryption, and so on. However, it is difficult to check

and enforce those data compliance requests for a customer. On top of this, cloud storage applications are becoming more versatile day by day; e.g., now, in many cases, multiple cloud service providers need to work together (e.g., in case of hybrid cloud and cloud federation use cases); those cloud service providers might be spread over in different parts of the world or might be using different infrastructures [59] [60] [22] [29]. As a result, in such systems, the cloud service providers mainly focus on the issues such as availability, reliability and performance and most of the time they ignore the demand for data compliance requirements [22] and thus are unable to offer any control to their customers to check the data compliance requirement such as checking the location of their outsourced data. The Intel IT Center surveys [61] [22] among 800 IT professionals that 78% of their organizations have to comply with regulatory mandates. Again, 78% of these organizations are concerned that cloud offers are unable to meet their requirements. In consequence, 57% of these organizations actually refrain from outsourcing regulated data to the cloud. Due to this lack of control over the treatment of outsourced data in cloud storage, a large set of clients are scared away from adopting cloud storage solutions. Note that this especially holds for organizations that deal with sensitive data such as the healthcare, financial, and government sectors [11].

2.5.2 Significant Attempts

In order to address the data location problem, we need to ensure at least the following:

- Cloud customers should be convinced by the cloud storage servers about the availability and integrity of the outsourced data at that point in time.
- Cloud customers should be convinced (preferably by the storage server) about that geo-location of the outsourced data; more precisely, at that point in time, outsourced data is actually stored /kept in the location where it is supposed to be.

Here we try to cover most of the significant research attempts which are relevant to address the data location problem.

Geo-Location Techniques: Geo-location techniques (e.g., ping-based solutions to estimate the server location by measuring the round trip time) could be used to get a 'rough'

estimate of the datacenter's /server's location and thus provide some knowledge to the users about their data location [62]. [35] [19] [36] [37] are some of the recent proposals for geo-location techniques. There are some initiatives (e.g., [16] [19] [37] [21] [23] [63] [64] [65] [66]) which incorporate geo-location based techniques to address the DLA problem of cloud storage environment. However, these geo-location technique based solutions are unable to fulfill one of the main requirements of the existing cloud storage environments (e.g., Amazon S3 [7], Microsoft Azure [8]) which is geo-replication of data. More precisely, geo-location based solutions do not work if a piece of data is stored/copied in multiple datacenters: what if a copy of your data is also stored outside your preferred location but you get the response to your query from a datacenter/storage server within your preferred location? Furthermore, in case of a geo-location technique based solution, the storage server is not formally involved in the data location finding process; everything takes place on the client side. As a result, the geolocation technique based solutions can be considered only as a passive means of getting location information about the outsourced data (unless the storage server formally vouches for it); it does not convey any non-repudiation guarantee from the storage server. So in case of a data location violation incident detected by these means, it cannot be used formally as an evidence against the storage servers/CSP.

Proof of Data Possession and solutions of this kind: The main purpose of any PDP scheme, and other schemes of this kind such as POR, is to perform integrity and availability checking on the data stored in an untrusted server. The cryptographic primitives, Proof of Data Possession (or Proof Of Retrievability (POR) [43]) have the potential to play an important role in devising a solution for the DLA problem, as the main reason for any PDP scheme is to offer the user to get verifiable assurance from the un-trusted storage server about the well-being of its data (i.e., the integrity of the data is still intact and the data is available). Most of the existing initiatives (e.g., [16] [64] [21] [67]) for solving the DLA problem use PDP schemes of different kinds.

After the proliferation of cloud storage environments, there have been a significant number of proposals for PDP schemes. While the objective of those PDP schemes is the same (offering data integrity and availability checking), PDP solutions vary among each other, and can be classified in terms of cryptographic primitives they use, additional services they offer

(e.g., supporting dynamic operation on data) and their efficiency (in terms of overhead regarding computation, communication and storage). Whereas some schemes are based on the RSA cryptosystem, some of them are based on different cryptosytems such as Elliptic Curve Cryptography, Paring-Based Cryptography and so on [68] [69]. In addition, these PDP schemes can be differentiated based on their support of dynamic operations on data. Some schemes support dynamic operation on outsourced data, while other schemes are only suitable for static/archival data. Furthermore, these schemes can be classified based on their offering for "public verifiability". Schemes with public verifiability enable anyone (not only the data owner) to perform the data integrity and availability checking. Similar to this, there exists another kind of PDP scheme which enables a proxy to perform remote data integrity checking on behalf of the original client/data owner [70]. In this case, the client has to delegate the remote data checking task to the proxy. Finally some of the attempts are for doing further enhancements in terms of performance and security over existing PDP schemes.

Recently there have been some interesting initiatives proposing multi-copy data possession scheme (e.g., [71] [72] [69] [73]; the objective of this kind of PDP scheme is to achieve data replication guarantees from the server. Many organizations pay for data replication services (for example storing n copies of data) to the CSPs to ensure higher availability. These multi-replica PDP schemes require the data owner to encrypt the file to generate n replicas of the same file. This kind of PDP scheme ensures that the CSP really stores exactly n replicas of its data and not less than that. However, this is totally different from ours. In this kind of solution, the goal is to find out whether n replicas are stored in the cloud as a whole; on the other hand, our research is to make sure user data is stored only in those locations (data centers) a user wants and nowhere else.

The papers [68], [74], and [75] include a survey on the existing PDP schemes and solutions of this kind. The following table provides a categorized list of these solutions:

Table 2 Classification of existing PDP schemes

Category	Existing PDP schemes
Built under different cryptographic assumption	[76] [77] [78] [79] [71] [80] [81] [82] [83] [84]
Supports only static data	[76] [77] [78] [79] [71] [80] [81] [82] [83]
Supports dynamic update	[85] [86] [87] [88] [89] [84]
Supports public verifiability	[72] [87] [73] [88] [89]
Assurance of multiple replica of data storage	[72] [87] [73] [69] [71] [90] [91] [92]

However, it is crucial to note that the assumption of a "single untrusted server" does not change on the recent PDP schemes proposed for cloud storage environment. In a cloud storage environment, there exists a number of storage servers/datacenters which can communicate with each other- so what if one storage server uses information or services from other servers to generate a false data possession proof. To use an existing PDP scheme in devising a DLAS solution, it is necessary to first consider the system model of the cloud storage applications and then apply the PDP scheme accordingly. Directly applying a PDP scheme in the cloud storage environment might not provide the "genuine" data possession guarantees by the storage server to the user because it does not automatically bind the data possession proof to the storage server (rather it binds it to the CSP as a whole). In section 5.3, we discuss in detail the limitation of existing PDP schemes in devising a DLAS solution for cloud storage environment.

Trusted Platform Module: Recently there have been a few proposals (e.g.; [66] [63] [93] [94] [95] [96]) where a Trusted Platform Module (TPM) has been used to achieve some sort of geo-location guarantee regarding virtual resources (and not specific to only cloud data, it could be a process as well). NIST initiated the first attempt [96] to do a prototype implementation of a solution of this kind where a server's geo-location information is written into the Program Configuration Registers (PCR) of the attached TPM. Whenever a resource (having a

geo-location requirement attached with it) comes to the server for service, the server checks the geo-location requirement of the resource and makes a service decision based on the geo-location policy it already has. More precisely, this solution is intended for the CSP's internal use and thus is not suitable for offering a data location assurance service to the cloud users. On the other hand, [95] proposes a mechanism based on TPM and a Trusted Authority for verifying the geographic location of any virtual resources (could be a process as well). Many of those TPM based solutions (e.g., [63]) mainly rely on the geo-location based techniques and thus inherit the limitations of those geo-location technique based solutions discussed earlier. Furthermore in some of those TPM based solutions (e.g., [66], [94], [95], [97]) TPM has to do most of the tasks and some of those tasks are beyond TPM's regular capabilities and thus require some modifications on the TPM to enable those additional capabilities. Most importantly, these TPM based solutions cannot address our main requirement which is offering verifiable data possession guarantees in case of geo-replication of data in different storage servers.

Some Proposals: To our knowledge, [16] is the first published work (a position paper) where the data location assurance (DLA) problem is addressed for the first time within the notion of "data sovereignty"; it provides a direction (not a solution) to develop a 'future tool' (i.e., a solution) based on geo-location techniques (i.e., IP ping-based) by which a user would be able to predict the location of their data (by observing the server's response time). Their suggested solution is based on the combination of 2 techniques: MAC-based PDP (Proof of Data Possession) scheme and geo-location technique. However, this kind of geo-location technique does not provide accurate results [62]. Furthermore, [16] is unable to address the geo-replication of data practice among cloud storage providers (e.g., Amazon S3, Microsoft Azure). Moreover, being a geolocation technique based solution, it also inherits the same limitation: whereas it can inform the client about a possible violation about their preference of data location, it cannot be used as a formal evidence against the storage servers / CSP.

Our proposed solution, DLAS (will be discussed in chapter 3), a data location assurance service solution proposed in [15] [98] is the first work which is able to handle geo-replication of data (i.e., DLAS works even if a piece of data is copied into multiple datacenters). Almost at the same time as the solution proposed in [15], both Watson et al. [23] and Albeshri

et al. [21] independently came up with a solution for the data location problem. Albeshri et al. later on proposed a solution [67] which is basically an improvement over their previous solution where they intend to reduce the storage overhead on the server side. The solution proposed in [23] is based on POR (Proof of Retrievability) and geolocaion techniques. However, their underlying system model is different and the authors of that paper [23] also agree with the fact that their solution is not suitable for existing cloud storage providers such as Amazon S3. Note that both of these solutions are based on geo-location techniques (IP ping based) and do not fulfil the geo-replication of data requirement of the existing CSPs (e.g., Microsoft Azure); furthermore their system models are either not suitable for existing CSPs (i.e., not suitable for cloud storage deployment) or are very restrictive [21] (they ask for a Third Party Auditor, TPA). As a continuation of their previous work [16], Peterson et al. recently proposed a solution which uses a constraint-based geo-location technique and PDP [64]. Complementary to [64], a similar solution has been recently proposed in [63]. However, unlike our proposed DLAS [15] [98], all existing solutions (e.g., [21] [67] [63] [64] [16]) do not work if a piece of data is copied into different storage servers.

Furthermore, in many of those solutions (specifically those based on geo-location techniques e.g., [23] [21]), the client does not receive a formal confirmation about the location of their outsourced data; he/she merely computes a rough estimate about his/her data location by himself/herself. Moreover, in most of those solutions, it is the user's task to do most of the required computations; the server's task is much less. However, we believe, from a user's perspective, one of the main reasons to go for a cloud solution is to transfer most of the (computation) burden, if not all, to the cloud.

Recently, we observe some attempts such as [29], [99], [100], where an encryption mechanism has been used to ensure geolocation restriction on accessing the outsourced data. For example, [29] tries to ensure geolocation guarantees by doing encryption on the outsourced data in a way that decryption is only possible within the legitimate region. This concept is almost identical to the "encrypt-then-store" technique we discussed in the first chapter. As we mentioned, in this thesis, we want to achieve "location guarantee" of the outsourced data, not the privacy guarantee. Furthermore, most of the organizations dealing with sensitive data (e.g., such organizations in Canada, US) will not even agree to store their encrypted data in a location

outside of their geographical jurisdiction. Moreover, some countries might enforce restriction on the use of strong encryption algorithms.

2.6 Solution Requirement & Strategy

In this section, we talk about the solution requirement and strategy for our proposed solutions. More precisely, here, first of all, we point out the requirements/characteristics we set for a solution of the DLA problem; we name this solution as DLAS (Data Location Assurance Service) solution. Then we talk about our strategy for devising a DLAS solution.

2.6.1 Solution Requirement

For a DLAS solution of cloud storage environments in geo-replication of data setting, we set the following requirements.

- It should allow a user to give both positive and negative geo-location preference for the data. For example: Keep my data in region X or Do not keep my data in region X.
- A DLAS solution should not put any constraint on the nature of the data; it should work for both encrypted and unencrypted forms of data.
- A client should be able to check/verify continuously the location of its data by launching the DLAS scheme anytime at her will. In return, a user will receive from the Cloud Storage Provider (more precisely from the storage server or the data center) a verifiable response regarding its data location and thus find out whether there is a violation regarding the user's data geo-location preference.
- The computation involved on the client side should be light weight so that any user can launch this protocol from any reasonably powerful device such as a laptop, tablet, and mobile phone and so on.
- The sole purpose of DLAS scheme is to continuously (e.g., a number of times per day) check whether a user's data geo-location preference has been violated or not, so the correctness of a DLAS scheme is of utmost importance here. But it is not time critical (as long as it takes a reasonable amount of time to get a response). Furthermore, communication and storage overhead are not of utmost importance here either; as long as a user will receive

a verifiable correct answer regarding its data location query from the CSP within a reasonable time (a few seconds), that user will be happy.

- The correctness and security of a DLAS solution should be formally analyzed.

2.6.2 Strategy

The Data Location Assurance (DLA) problem has been identified in the cloud computing literature as a very challenging and important problem which needs to be solved. The geo-replication of data practice among the existing CSPs makes it even more challenging. The objective of this research is to propose solutions for this challenging problem. As we pointed out in chapter 1, if the CSP is totally malicious on a permanent basis, there is no way to achieve a solution for the DLA problem in the geo-replication of data setting as CSP has total physical control over the data. But the actual scenario is promising, as in reality, CSPs will not be totally dishonest on a permanent basis for the sake of their business reputation. Our assumption is that almost all the time the CSP will follow the protocol correctly, but in the worst case, the CSPs might be malicious for a very short amount of time and it may try to generate a false proof when it gets a data location query from the user assuming that hiding the violation of user's data location preference is necessary for protecting its business reputation.

Our strategy for proposing the DLAS (Data Location Assurance Service) solution in the geo-replication of data setting is very simple, we just follow our research methodology stated in section 1.5.

At first, we study all potential organizations that require the DLAS solution for adopting any cloud storage solutions. Whereas any organization (i.e., cloud user) would love to have the DLAS solution, organizations that deal with sensitive data (e.g., health sector, financial sector, and defense) will definitely need the DLAS from the CSP. Besides studying the potential organizations (mainly public organizations and other organizations dealing with sensitive data and having geo-location restrictions on their data) seeking for the DLAS, we also study the existing CSPs (e.g., Amazon S3 [7], Microsoft Azure [8]) which are currently providing the storage services. We mainly study Amazon S3 [7], and Microsoft Azure [8] as they are not only very popular storage service providers but also practise the geo-replication of data in order to offer a higher availability guarantee. The purpose of this study is to characterize the

system and attack model for the DALS solution. During our study, our focus is to find the answer of the following question: " What are the characteristics of the underlying system and attack model if Organization X (dealing with sensitive data and having geo-location restriction on their data) wants to take storage service from the CSP Y (practising geo-replication of data)?"

After studying those potential cloud storage applications and existing CSPs, we come to the conclusion that we have to propose DLAS solutions for different system and attack models. Note that the characteristics/requirements of observed cloud storage applications (as well as the behaviour of the existing CSPs) play a profound role in defining all of our system and attack models.

Regarding **the system models**, we consider 2 kinds of system models for the DLAS solution: 3-entity based system model, and 2-entity based system model as it will cover all potential cloud storage applications requiring the DLAS solution. Whereas there are 3 entities (User, Enterprise and CSP) involved in the 3-entity based system model, a user directly interacts with the CSP in case of the 2-entity system model. Furthermore, a limited number of cloud storage applications fall into the 3-entity system model. On the other hand, the 2-entity system model is a very generic model and in many cases, it can be easily extended to a 3-entity system model and thus is able to cover almost all potential (and existing) cloud storage applications.

Regarding **the attack models**, as it is not possible to offer DLAS solution for a totally malicious attack model (discussed in 1.5), we decide to first propose a solution with a less adversarial attack model and progressively look for solutions with more adversarial attack models. In order to deal with the unique challenges involved in the DLA problem, in some cases, we have to define new attack models as it does not fall into any existing attack models.

2.7 Conclusion

In this chapter, we have provided a solid background to a reader to properly understand this thesis work. In order to do this, we have- (1) presented key aspects of cloud computing

and existing cloud storage environments such as Amazon S3, Microsoft Azure, and Google Cloud Storage (2) discussed briefly all important cryptographic primitives along with cryptographic tools, (3) covered most of the significant related attempts related to this thesis work, and (4) provided an overview of the solution requirement and strategy for the DLA problem.

3 Proposed Solution 1: DLAS

This chapter includes our first proposed solution for the Data Location Assurance (DLA) problem. We organize this chapter as follows: first we point out the objective of this solution, followed by the system and attack model, and cryptographic primitives used for this solution. We then discuss our proposed solution in detail.

Two research papers (one conference and one journal) have already been published from this proposed solution.

3.1 Objective of this solution

The main objective of this proposed DLAS solution is to facilitate cloud users not only to give preferences regarding their data location but also to receive verifiable assurance about their data location from the Cloud Storage Provider (CSP). Unlike other existing solutions (e.g., [16] [21] [23] [64] [67]), the DLAS solution works in the geo-replication of data setting of the cloud storage environment and allows its users to give also a negative location preference regarding his/her data (e.g., "do not keep my data in region X").

3.2 System and Attack Model

System Model: We consider a system model of a cloud storage environment which consists of 3 entities: Enterprise (E), Users (U) and Cloud Storage Provider (CSP). This system model is very common in the real world: any (cloud) service provider (not having the storage facility) could play the role of E by establishing an SLA with any cloud storage provider(CSP) in order to offer data storage service to any individual user/ company (U). In this case, it is obvious that different U (i.e.; different companies) will have different data location preferences. Another real-world example would be a university (i.e., E) interested to take storage service from a CSP for its employees and students (i.e., U).

Attack Model: Like other security solutions [24] [25] [26] [27] [28] , we consider the well-known "honest-but-curious" Cloud Server Model. That is to say, cloud storage servers will perform their regular tasks correctly; however, they may try to find out as much secret information as possible regarding their clients.

We consider the following attack scenarios where an adversary may act:

As the CSP: A CSP may try to store a piece of data in a location that does not satisfy user's data location preference and it tries to keep this fact undetected.

As the Enterprise: An Enterprise may try to find more information regarding the datacenters of a CSP.

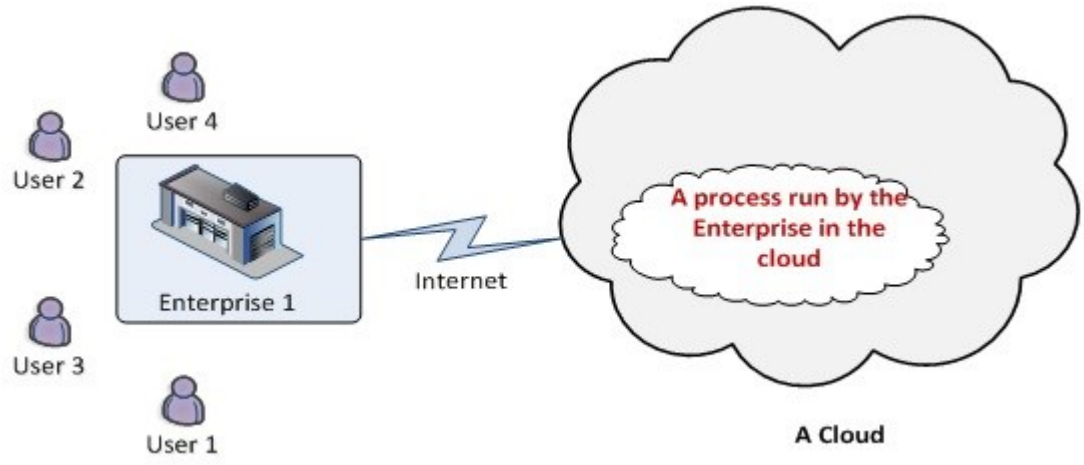

Figure 3 System Model considered for DLAS solution

Assumptions: For this proposed solution, we consider public cloud as the deployment model (for generality) and IaaS (Infrastructure as a Service) as the service model. Now we point out key characteristics of the system and attack model considered for the proposed DLAS solution:

- A CSP has a number of datacenters, each having a number of storage servers and those datacenters are scattered all over the world. However, we restrict our discussion of this proposed solution in terms of datacenters.
- Like other real cloud storage service providers (e.g., Microsoft Azure [8]), our system model also considers the fact of geo-replication of data (i.e., a piece of data

will be stored in multiple datacenters) in order to reduce the risk of damage due to natural disasters.

- An Enterprise[1] can execute an arbitrary number of processes (which also includes executing encryption algorithms) in the cloud to manage its users' data. In this case, an Enterprise has total control over those processes; the CSP cannot retrieve any (meaningful) information about those processes even though they are being executed in the cloud. Note that this is a valid assumption for the IaaS service model.
- A user does not need to perform any computation; all computations are performed in the cloud.
- A user does not need to be available all the time. A user will come online only when he/she needs to either make a data storage request or verify the location of a particular data item. In all cases, a user has no direct interaction with the CSP; a user will directly interact with the processes executed by the Enterprise in the cloud. However, those processes have regular interactions with the CSP.
- We assume that the CSP as well as the processes executed by the Enterprise[1] in the cloud are always available/online and they have sufficient computational power and storage capacity.
- The CSP is responsible for sending regular updates (once in a day; e.g., at 12:00 am) to the Enterprise which contains data location information for all data of an Enterprise. Many CSPs will do this because the proposed DLAS solution will enhance the trust level of users towards the CSP and eventually the CSP can attract more clients; furthermore, the market of cloud computing is becoming competitive.
- All communication channels (between user and the Enterprise, and between the CSP and the Enterprise), are assumed to be secure.

3.3 Cryptographic Primitives Used

Zero Knowledge Sets and CP-ABE are the main building blocks of this solution. The details about ZKS and CP-ABE were already provided in section 2.4.

[1] Throughout the discussion of our proposed solution, when we say "the Enterprise" it actually means the processes run by the Enterprise in the cloud.

3.4 The Proposed Solution: DLAS

The proposed solution, DLAS, has 4 important phases: Bootstrapping Phase, Offline Phase, User Data Registration Phase, and Data Location Verification Phase. Among the four phases, the Bootstrapping phase has to be executed first; the order is not important for other 3 phases; they may coincide with each other. Whereas the User Data Registration Phase and Data Location verification Phase are online (i.e. always available for execution), the Offline phase is executed once in a day.

The Bootstrapping phase has to be executed prior to the start of the data storage service. In this phase, the CSP generates a fixed number of pseudonyms (i.e., datacenter ID) for each datacenter it has, aiming to group each data location preference list with similar size (i.e., having a fixed number of datacenters). A data location preference list represents a particular region (could be a country or a continent). The CSP then commits its datacenter's location information for each data location preference list it has, to the Enterprise (E) using the ZKS commitment scheme. Because of the use of ZKS commitment scheme, the only thing an Enterprise can do is to verify whether a particular datacenter is within a particular region or not.

The User Data Registration phase not only facilitates a user to give preferences regarding their data location during making a data/file (we use the two terms data and file interchangeably) storage/registration request to the cloud through the Enterprise, but also provides the necessary credential to the user which will later on enable the user to verify the location of a piece of data. A user can express his / her data location preference in either one of two ways: "**Keep my data/file in region, R**" or "**Do not keep my data/file in region, R"** where R can represent a country (e.g., USA) or a continent (e.g., Europe). Depending on the CSP's internal storage policy, the CSP may store the user's data/file in more than one datacenter; however, those datacenters should satisfy user's choice about the data location (i.e., R).

In the Offline phase, the Enterprise, E, builds an encrypted database (using the CP-ABE encryption scheme) from the data location information it regularly received from the CSP about all users' data of E; this encrypted database is made in such a way that later only a legitimate user (having the right credential) is able to decrypt and thus retrieve data location Information (i.e., datacenter id/s) only for his/her data, and thus it works as a filter by separating a valid verification request from an invalid one in the verification phase. We call this phase an offline phase because the user (U) does not participate in this phase; only the CSP and the Enterprise are involved.

In the verification phase, a user can make a verification request regarding his/her data at any time and get a verifiable assurance about their data location from the CSP.

Notation: The important notation of our proposed solution is listed here.

Notation	*Description*
U,E,CSP	*General representation of User, Enterprise and Cloud Storage Provider respectively*
userID, dataID, dcID	*Represents a User, a piece of data (e.g., a file) and a datacenter respectively*
R	*Represents user's data location preference (e.g., Europe)*
Δ,Δ'	*Implicit security parameter of the ZKS and CP-ABE protocols respectively*
σ, K,SK	*σ and K are random reference string and security parameter of ZKS protocol respectively. SK is the private key of the ZKS protocol*
PK, MK	*Public parameters and Master key of the CP-ABE protocol*
A	*Represents an access structure (can be based on attributes such as R, userID, dataID). It is an important parameter of the CP-ABE scheme.*
S	*A set of attributes; it could be dataID, dcID…*
Credential	*The output of the key generation algorithm of the CP-ABE protocol*
ρ	*Represents user inputs regarding the storage request (consists of file, selection of regions etc)*
Ω	*It may include time stamps and other parameters related to the storage policy.*
ID	*A set of pseudo IDs generated from a real ID, id*
DCID, DCID_res	*Both represent a set of dcIDs*
DB	*An elementary database generated from a given set*
DB_commit	*An encrypted database; generated by applying the commitment algorithm of ZKS protocol on the database, DB*
userDataStorageReqDB / DB2	*A database having following attributes: userID, dataID, dcID, R*
dataLocationDB_E /DB1	*A database having following attributes: dataID, dcID*
tempDB	*A database having following attributes: dataID, userID, dcID, R*
dataLocationFinderDB	*A database generated by encrypting each row of tempDB*

3.4.1 DLAS Scheme-Formal definition

Here we formalize the notion of DLAS, with the system and attack model outlined in section 3.2.

Definition 1. A DLAS scheme consists of 4 phases.

Phase 1: Bootstrapping Phase: This phase consists of 5 algorithms.

Setup_Securityparameter_ZKS (Δ) → (σ, K): is run by the CSP. It takes Δ (implicit security parameter) as input and outputs σ (a random reference string) and K (the security parameter). The output of this algorithm is publicly available.

Setup_CPABE (Δ') → (PK, MK): is run by the Enterprise, E. It takes Δ' (implicit security parameter) as input and generates the public parameters PK and master key MK.

Generate_pseudoID (id, n) → ID (a set of pseudo IDs): is run by the CSP. It takes the real id and n (number of pseudo IDs that need to be generated) as a parameter and outputs ID, a set having n pseudo IDs.

Convert_set_to_elementaryDatabase (S) → (DB): is run by the CSP. It takes a set (e.g., S = {x1, x2, x3, …}) as input and generates an elementary database, DB as output (where each row of DB is in the form of <xi, 1>). For each data location preference list, the CSP executes this algorithm.

Commit_ZKS (DB, σ, K) → (DB_commit, SK): is run by the CSP. It is the commitment algorithm of the ZKS protocol. This algorithm takes elementary database, DB, σ, K as inputs and generates the commitment, DB_commit (which could be also considered as the public key) for DB and the private key, SK. The CSP generates this commitment (i.e., DB_commit) for each data location preference database and sends them to the Enterprise, E.

Phase 2: User Data Registration Phase: This phase consists of 3 algorithms. This phase starts with a data/file storage request having a location preference, R, made by a user (U) to the CSP through the Enterprise, E. This results in the file to be stored in one or more of the datacenters of the cloud storage provider.

GenStorageReqParam (ρ) → (dataID, R, Ω): The Enterprise runs this algorithm once it receives the storage request from its user. This algorithm takes ρ (user inputs regarding the storage: file, selection of regions) as inputs and generates all necessary parameters (such as dataID, a set of data location preferences, R and Ω (which may include time stamps and other parameters related to the storage policy).After storing this storage request information in the database DB2, E forwards the storage request (E→CSP: <dataID, file, R, Ω >) to the CSP.

FindDataCenters (R, Ω) → DCID: is run by the CSP. It takes R (a set of data location preferences) and Ω (internal storage policy) as inputs and outputs DCID, a set of dcIDs (data-center id), where the user's file will be stored.

KeyGen_CPABE (MK, S) → (Credential): It is the key generation algorithm of the CP-ABE scheme which is run by E. This algorithm takes the master key MK and a set of attributes, S (which may include userID, dataID, R, timestamp) as inputs and outputs the Cre-dential, which will be sent to the user.

Phase 3: Offline Phase: This stage consists of 2 algorithms. This phase starts with the CSP who regularly sends the updated database, DB1 (which includes <dataID, dcID> information of the Enterprise, E) to the E. The Enterprise then builds an encrypted database using DBI and other information about its users.

CheckUpdate (DB1, prev_Hash) → $a \in \{yes, no\}$: is run by the CSP. It takes DB1 and prev_Hash (i.e., previous hash of the DB1) as inputs and determines whether there is an update/change in the database which needs to be send to the E.

BuildEncryptedDB (DB1, DB2) → DB3: is run by the E. It takes database DB1 and DB2 (which includes <userID, dataID, R> as inputs and generates an encrypted Database, DB3, using the encryption algorithm of the CP-ABE scheme.

Pseudocode:

1. *tempDB←select userID, DB1.dataID, dcID, R from DB1, DB2 where DB1.dataID=DB2.dataID*
2. *For each row(ri) in tempDB*
   2.1. A←set the access structure based on attributes such as R, userID, dataID of ri
   2.2. ri'(i-th row of DB3)←Encrypt_CPABE(PK, ri, A)

Phase 4: Data Location Verification Phase: This stage consists of 2 algorithms. This phase starts with a data location verification request made by the user and ends with producing a verifiable assurance regarding the user's data location (if the request is valid).

VerifyRequest (DB3, userID, dataID, R, credential) → DCID_res or ⊥: is run by the Enterprise, E. It takes userID, dataID, R and credential (SK) as inputs and outputs DCID_res (a set of dcIDs) if the data location verification query is valid; otherwise it returns ⊥.

Pseudocode:

1. *DCID_res←{}*
2. *count←0*
3. *validUser←false*
4. *For each row(ri) in DB3*
   4.1. temp←Decrypt_CPABE(PK, ri, credential)
   4.2. if (temp) //decryption is successful
      4.2.1. validUser←true
      4.2.2. dcIDtemp← select dcID from ri
      4.2.3. DCID_res← DCID_res ∪ {dcIDtemp}
      4.2.4. count←count +1
5. *if (NOT validUser)*
   5.1. return ⊥
6. *if (count>0)*
   Return DCID_res
   else if (count = 0)
   "This data item is no longer available in the CSP"

Note that the decryption process of any row i of the database, DB3 will be successful if the user's credential satisfies the access structure of row i. In other words, a user will get dcID (or dcIDs if the data is stored in multiple datacenters) only for his/her data.

VerifyDataLocation (DCID_res, DB_commit_DBs, R) ←a ∈{yes, no}: is run jointly by the CSP and the Enterprise. It takes DCID_res, user's preferred storage location/s R and DB_commit_DBs (commitments for all location preference list DB) as inputs and produces a verifiable answer (in Boolean form) regarding user's data location in a given region.

Pseudocode:

1. *For each dcID in DCID_res*
   1.1. *For each database,DB_commit in DB_commit_DBs*
      1.1.1. *CSP generates a proof, π_x for dcID (dcID ∈ DB_commit or dcID ∉ DB_commit) by executing Proof_ZKS (DB, K, σ, PK, SK) and sends it to E.*
      1.1.2. *E verifies the membership/non-membership proof, π_x by executing Verify_ZKS(x, π_x , K, σ , PK): result ←Verify_ZKS(x, π_x , K, σ , PK)*
      1.1.3. *If ((result AND dcID ∈ DB_commit AND DB_commit Not IN R) OR (result AND dcID ∉ DB_commit AND DB_commit IN R))*
         1.1.3.1. *Return false*
      1.1.4. *Return true*

3.4.2 DLAS Construction

Here we describe how a DLAS scheme can be constructed. This DLAS construction relies heavily on two cryptographic primitives: the ZKs protocol and the CP-ABE encryption scheme.

3.4.2.1 Bootstrapping Phase:

In this phase, the CSP commits its datacenters' location information for each data location preference list it has, to the Enterprise (E); here a data location preference list represents a particular region (could be a country or a continent). However, there are some other tasks involved in this phase. Figure 4 provides an overview of the Bootstrapping phase. Whereas both the CSP and the Enterprise participate in this phase; most of the tasks are performed by the CSP. This phase works as follows:

Step 1: CSP executes the Setup_Securityparameter_ZKS() algorithm to generate necessary security parameters for the ZKS scheme. The Enterprise, E, also generates public parameters, PK, and master key, MK, by executing the setup algorithm of the CP-ABE scheme which is Setup_CPABE().

Step 2: CSP creates a fixed number of pseudo IDs for each (actual) datacenter (dcID) by executing the Generate_pseudoID(). The reason for doing this is to minimize the reuse of the same dcID so that it would be more difficult for both the user and the Enterprise to make any prediction regarding the datacenter's location without participating in the ZKS protocol.

Step 3: For each data location preference list, CSP creates an elementary database (EDB), DB, using the Convert_set_to_elementaryDatabase() algorithm. A simple way of creating an EDB from a given set is to use each element (in this case it is dcID) of the set as a key and assign an arbitrary value (e.g., 1) for each key. If we assume that a CSP has N data location preference lists for N Regions, there will be N EDBs.

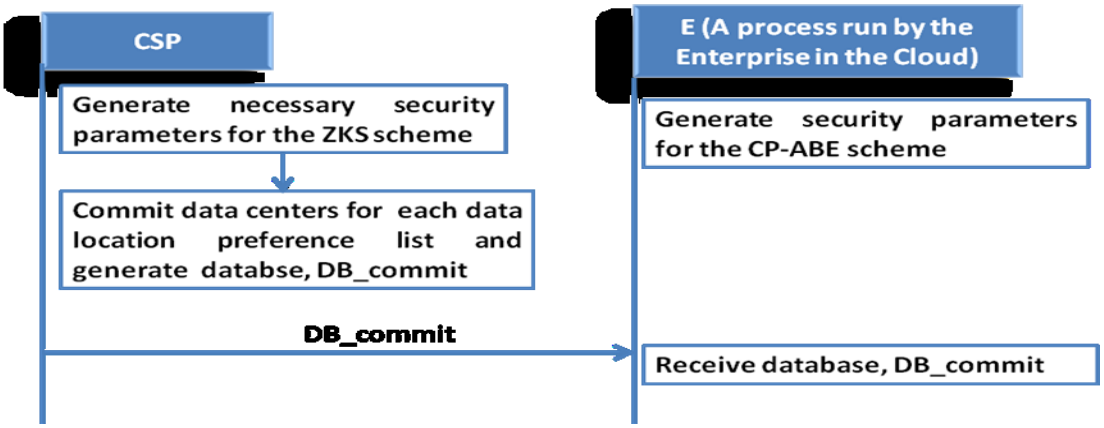

Figure 4 The Bootstrapping Phase

Step 4: At this point CSP executes the Commit_ZKS() algorithm to generate a commitment for each database, DB, generated in step 3; we call this database, DB_commit. This commitment algorithm also produces associated private key SK, which will later be used by the CSP in the data location verification phase.

Step 5: The CSP sends all DB_commit databases to E. Note that whenever there is a change (i.e., a new datacenter is added into or removed from a particular region, R) in any data location preference list, the CSP must commit the updated preference list to the Enterprise.

3.4.2.2 User Data Registration Phase

User, Enterprise and CSP participate in this phase. In our model User cannot directly contact CSP; Enterprise resides in the middle between User and CSP. Figure 5 provides an overview of this phase. This phase works as follows:

Step 1: This phase starts when a user makes a data/file storage request with a particular storage location preference, R. After receiving the storage request from the user, the Enterprise executes the GenStorageReqParam() algorithm in order to generate necessary parameters (it might include dataID, user's location preference R, timestamp and so on) required for sending the real storage request to the CSP.

Step 2: The Enterprise, E, first stores all necessary information regarding the user's storage request in the database, userDataStorageReqDB. [Note that the CSP has no control over the databases created by E since we consider IaaS as the deployment model.] It then forwards the storage request to the CSP

$$E \rightarrow CSP: <dataID, R, \Omega >$$

Step 3: Upon receiving the storage request from step 2, the CSP executes the FindDataCenters() algorithm to find a set of datacenters (i.e. dcIDs) where the file can be stored. Upon successful storage, the CSP firsts updates its database dataLocationDB_E (<dataID, dcID>) and then sends an OK message to the Enterprise.

We assume that the CSP ensures the correctness of the database, dataLocationDB_E (i.e., are these data items really stored in those specified datacenters?) prior to sending it to the Enterprise.

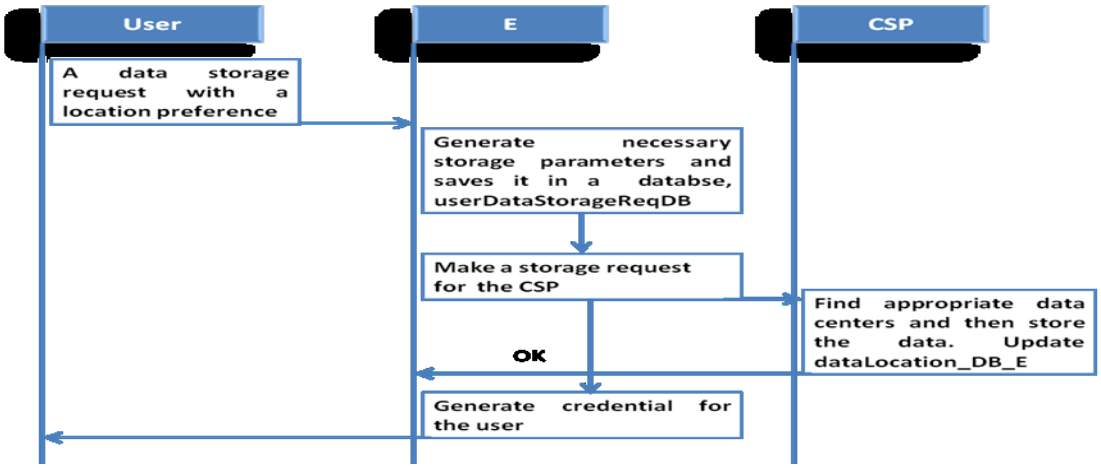

Figure 5 The User Data Registration Phase

Step 4: After receiving the OK message from the CSP, the Enterprise generates a credential by executing the GenCredentials() algorithm. It then sends the credential to the user as a response to this/her storage request.

E → U: <userID, dataID, credential, R>

This credential later on enables users to verify their data location.

3.4.2.3 Offline Phase

The CSP and the Enterprise, E take part in this phase. Figure 6 shows an overview of this phase. The Offline phase works as follows:

Step 1. The CSP first executes the CheckUpdate () algorithm in order to find out whether there is any change in the database datalocationDB_E. If the algorithm returns yes, the CSP sends the updated datalocationDB_E to E.

Step2. After receiving the database datalocationDB_E, the Enterprise executes the (BuildEncryptedDB (datalocationDB_E, userDataStorageReqDB)→ dataLocationFinderDB) algorithm to build an encrypted database, dataLocationFinderDB

3.4.2.4 Data Location Verification Phase

User, Enterprise and the CSP all participate in this phase. Figure 7 gives an overview of this phase. This phase works as follows:

Step 1: This phase starts when a user makes a data location verification request to the CSP through the Enterprise. Since the Enterprise, E, works as a proxy/middleman between User and CSP, the Enterprise receives the data location verification request.

U →E: (userID, dataID, credential, R)

Step 2: At this point, the Enterprise, E executes the [VerifyRequest(dataLocationFinderDB, userID, dataID, R, credential)] algorithm and finds out (i) whether the verification request is valid and (ii) if so, finds DCID_res, a set of all datacenter Ids where that particular data is stored.

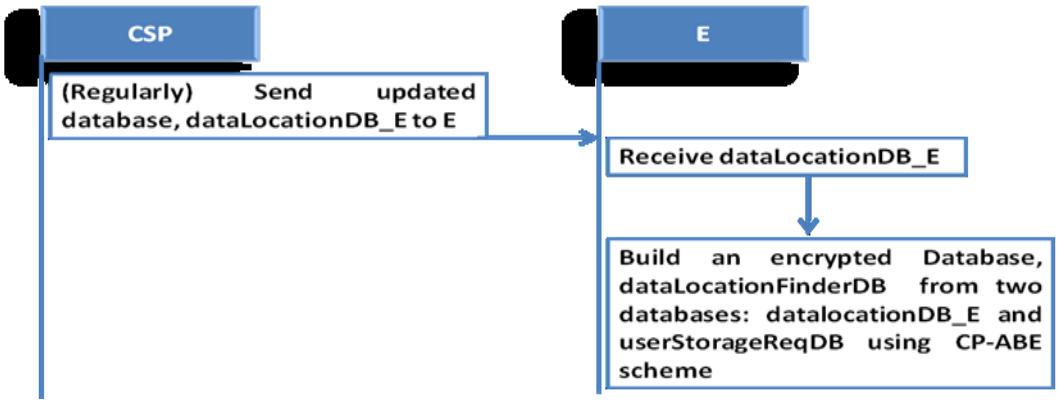

Figure 6 The Offline Phase

Note that this algorithm will also notify the Enterprise and a user, if a particular data item is no longer available in the CSP (e.g., this data item was accidentally deleted from the storage server/s).

Step 3: The Enterprise then executes the [VerifyDataLocation (DCID_res, DB_commit_DBs, R) ← a ∈{yes, no}] algorithm and finds out if there is any violation regarding user's data location preference and lets the user know about this. From the user's perspective, he/she will receive one of two messages as a response to their data location verification request; for example

"Yes, your data is now in your preferred location, Europe"

OR

"Your data location preference has not been properly maintained".

3.4.3 Discussion & Analysis

3.4.3.1 Security Analysis of DLAS

Here we do the security analysis of the DLAS solution for the attack scenarios considered in section 3.2.

Assumptions: Our security analysis is based on the following assumptions (which are already discussed in section 3.2).

Assumption 1: The CSP regularly (and correctly) sends the updated database (dataLocationDB_E) that contains data location information for all data of an Enterprise.

Assumption 2: The CSP has no control over the processes and databases) executed and managed respectively by the Enterprise, E, in the cloud.

Those assumptions are valid for the system model considered in section 3.2. Assumption 1 makes sense since in this case the CSP sends data location information (basically just the pseudo-IDs of datacenters) for all data, not for an individual data item; furthermore the database is sent offline and it is not sent as a response to some live request. Moreover the

activity of regularly sending this database to the Enterprise should be a part of the SLA between the CSP and the Enterprise. Furthermore, assumption 2 is a practical example of IaaS (Infrastructure-as-a-Service).

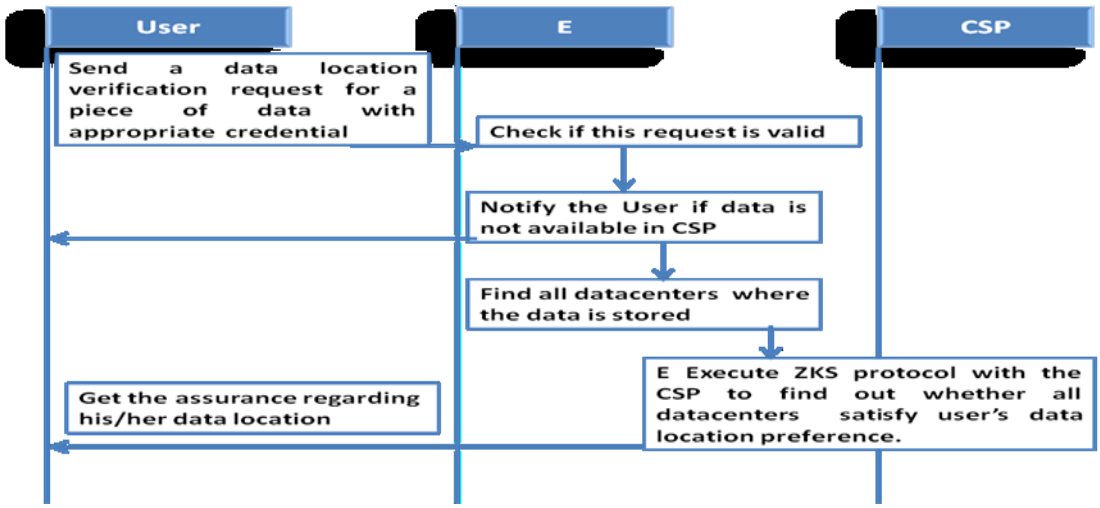

Figure 7 The Data Location Verification Phase

Case 1: Can the CSP lie about the data location for a particular data item: We make the following claim for this case:

Claim 1: For the system and attack model considered for the DLAS solution, if the soundness property of the ZKS protocol holds, the CSP cannot lie about the location of the data (i.e. datacenter's location).

Proof strategy: Claim 1 is of the form A → B where

A: the soundness property of the ZKS protocol holds

B: The CSP cannot lie about the location of the data

In order to prove claim 1, we will show that ¬ B → ¬ A

Proof: This can be proved easily. First assume that the CSP can lie about the location of the data.

Now from assumption 1, the database dataLocationDB_E is correct. Furthermore the CSP cannot associate a specific user with specific data since it cannot get any information from the processes executed by the Enterprise in the cloud (based on assumption 2).

At this point, the CSP will **only** be able to lie about the location of the data by telling lies about the location of the datacenters. However, the CSP has already committed all data-centers in the bootstrapping phase (which is assumed to be correct). So in order to be able to tell a lie about the location of the data, the CSP has to be able to generate both membership and non-membership proofs for the same elements (i.e., for a dcID) in a given set (e.g., data-center list in Europe). In other words, a CSP will be able to lie about the location of the data if it can show that:

(dcID $\in$ DCID) AND (dcID $\notin$ DCID)

Thus, if the CSP can lie about data location, then the ZKS protocol is not sound, and so assuming $\neg$ B implies $\neg$ A, which proves claim 1 $\Diamond$.

Note that Micali et al. has proved that the soundness property of the ZKS protocol holds (i.e., In the ZKS protocol, the probability for producing two proofs ($x \in S$) and ($x \notin S$) for the same element, x in a given set, S, by a prover and remain undetected to a verifier is very negligible); the proof can be found in [38].

Case 2: Enterprise's Knowledge about the CSP's datacenters: We make the following claim for this case:

Claim 2: For the system and attack model considered for the DLAS solution, if the zero knowledge property of the ZKS protocol holds, the Enterprise cannot learn any additional information (regarding the datacenters of the CSP) other than just verifying the data location of a particular data item.

Proof strategy: We follow the same proof strategy as for claim 1. We first assume that the Enterprise learns additional information regarding CSP's datacenter ($\neg$ B) and then it causes

the violation of zero knowledge property of the ZKS protocol (¬ A); in other words we show ¬ B → ¬ A.

Proof: The proof is straight forward.

Assume that the Enterprise can learn additional information about the CSP's datacenter location. According to the system model, this is possible only if the Enterprise can extract more information (apart from checking the (non-) membership relation of a given set) from the committed database, DB_commit, which is only possible if the zero knowledge property of the ZKS protocol does not hold.

Informally 'zero knowledge' means that an efficient Verifier learns only the values D(x) from his interaction with the Prover, and nothing else (not even the size of D). In particular the Verifier does not learn the values D(x') for an x' not queried to the Prover. The formal definition and proof of "zero knowledge" property of the ZKS protocol are given in [38]. In the context of the DLAS solution, it means that the Enterprise can know the location of at most n datacenters (i.e. n dcIDs) if all of its users' data is stored in n datacenters. Most importantly, the Enterprise will not be able to know even how many datacenters are located in a particular region (e.g. in USA). In addition, due to the adoption of the pseudo ids for the dcIDs, the Enterprise will see less repetition of dcIDs.

So we can conclude that ¬ B implies the violation of the zero knowledge property of the ZKS protocol (i.e., ¬ A) and thus claim 2 holds ◊.

3.4.3.2 Performance and Implementation Issues

Here we discuss performance and implementation issues for each phase of the proposed solution, DLAS, in order to justify the feasibility of adopting this DLAS solution in real cloud environments.

Bootstrapping Phase: This phase is off the critical path of the user's request. As a result, the performance (more precisely the timing issue) of this phase is not a significant issue for the DLAS solution.

Committing datacenters for each location preference list (i.e. coming up with DB_commit databases) using the ZKS protocol is the main operation of this phase. The commitment phase of the (original) ZKS protocol [38] is fast; in order to commit an elementary database, DB, it requires three modular exponentiations and one hashing for each element in the set (i.e. the EDB, DB). Note that in the DLAS solution, each element of an EDB represents a datacenter and in practice the number of datacenters in a particular region R (where R represents a country or a continent) would be in the range of 10-20 (at most). As a result, the EDBs used in the DLAS solution, will have a reasonable size.

Data Registration Phase: The performance of this phase is very important since it is on the critical path of a user's request.

The most expensive operation of this phase is the credential generation against a user's data storage request. Note that the credential is generated by executing the key generation algorithm of the CP-ABE algorithm. The key generation algorithm requires two exponentiations for every attribute assigned to the user, and the private key consists of two group elements for every attribute [41].

Offline Phase: This phase is off the critical path of a user request and thus its performance is not very crucial from the user's perspective. Recall that this phase only occurs once in a day.

The main goal of this phase is to build an encrypted database using the encryption algorithm of the CP-ABE scheme. The encryption algorithm requires two exponentiations for each leaf in the ciphertext's access tree and the ciphertext size will include two group elements for each tree leaf. As noted earlier, the detailed description regarding the performance and implementation issues of the CP-ABE scheme is given in [41].

Data Location verification Phase: The performance of this phase is very crucial since it is on the critical path of a user's request.

This phase includes 3 main operations: decryption of the dataLocationFinderDB database (performed by the Enterprise), execution of the proof generation algorithm of the ZKS protocol (performed by the CSP) and the verification algorithm of the ZKS protocol (performed by the Enterprise).

According to [41], the decryption algorithm requires two pairings for every leaf of the access tree that is matched by a private key attribute and one exponentiation (at most 2) for each node along a path from such a leaf to the root. Note that in case of the DLAS solution, the access structure (i.e. access tree) would be very simple having a small number of nodes, which makes the decryption algorithm faster.

Between the proof generation algorithm and the verification algorithm, the proof generation algorithm dominates (in terms of computation and space requirements). In the proof generation algorithm of the ZKS protocol, the need for memory (not the computation) increases with the number of new proofs of non-membership; fortunately the memory should not be an issue in case of the DLAS solution as the CSP will generate the proof and it has adequate computation and storage capability. Furthermore, it is also possible to adopt an improved ZKS protocol [39] which is able to generate shorter proofs than the original ZKS protocol; it can generate proofs of membership that are (approximately) 33% shorter, and proofs of non-membership that are almost 73% shorter than the original ZKS protocol.

[39] also includes a complete performance analysis of the original ZKS protocol for different security settings, it indicates that for a moderate size of EDB (having 200 elements with key of 120 bits), verification of a membership (or non-membership) proof requires less than 1 sec (and less than 5 seconds for non-membership) for a standard PC (e.g., Intel Core 2 Duo CPU at 2.4 GHz). Note that the EDBs of our DLAS solution should have less than 200 elements (which would represent datacenters in a given location) in real scenarios.

3.4.3.3 Further Comments on the Proposed Solution

Why did we choose CP-ABE? In the proposed DLAS solution, it might be possible to consider other means of access control solutions instead of the CP-ABE scheme. However, one of the main reasons for choosing CP-ABE is that it is very generic and flexible as well. On one hand, it can be used to develop the typical encryption scheme which allows only a valid user to decrypt; on the other hand, it can also be used in the RBAC (Role Based Access Control) model where a user's access will depend on his/her role. Another significant advantage of CP-ABE is that you can even put the encrypted database (using CP-ABE) into a malicious server. Last, but not least, we also consider the fact that the implementation of the CP-ABE scheme is publicly available [42].

Room for improvements in the ZKS protocol: The ZKS (Zero Knowledge Set) protocol is one of the main building blocks of our proposed DLAS solution. At present there exist a number of papers [39] [69] which have proposed some sort of improvements (in terms of performance or security) over the original ZKS protocol proposed by Micali et al. in [38]. However, we considered the original ZKS protocol for our DLAS solution since it is comparatively simple; furthermore, in this paper our intention is just to present a basic construction of DLAS solution, and for this, the original ZKS protocol is more suitable than [39] [101].

Further comments on the assurance received from the CSP: Note that the verifiable assurance a user receives in the DLAS solution reflects (at most) 24 hours' earlier state of their data location since the database dataLocationDB_E is sent to the Enterprise once in a day (at 12:00 am every day). However, the CSP cannot lie to the users about that state.

3.5 Conclusion

In this chapter, we have proposed the DLAS, the first non-geolocation based data location assurance solution for cloud storage environments. The system model we considered (the 3-party model) for our DLAS solution is very common in real cloud environment settings; furthermore our attack model ('honest-but-curious server model') is also very well-known and

widely used in many security solutions for cloud environments. The main reason behind coming up with a non-geolocation based solution (instead of a geolocation based solution) is that a "geo-location" based solution cannot fulfill one of the important needs of existing cloud storage environments, which is "geo-replication" of data; more precisely, existing geolocation technique based data location assurance solutions are unable to provide data location assurance to the users when a piece of data is stored/copied in multiple datacenters/servers.

Our DLAS solution allows users to give their data location preference in the following way: "Keep/ do not keep my data in region R (e.g., R can be a country or continent)"; later, users will receive provable assurance of their data location from the CSP (Cloud Storage Provider) with respect to their data location verification request. We also include a security and performance analysis of the DLAS which shows the feasibility of adopting the DLAS solution for real cloud environments. Unlike other existing solutions [21] [67], the DLAS solution also satisfies the geo-replication practice (i.e., a piece of data can be stored in multiple datacenters) of the existing CSPs (e.g., Microsoft Azure).We also believe that DLAS has the potential to be used as one of the building blocks of the "accountable cloud" [102].

4 Proposed Solution 2: HDLAS: A 2-entity based solution

This chapter includes our second proposed solution for the Data Location Assurance (DLA) problem. We name this solution as HDLAS- **H**ardware-based **D**ata **L**ocation **A**ssurance **S**olution.

One conference paper [103] has been published from this proposed solution.

4.1 Motivation & Objective of this solution

Motivation: There are a number of limitations of our 3-entity based DLAS solution discussed in the previous chapter. Not only does the 3-entity based system model of the DLAS not cover all existing cloud storage applications, but also the attack model considered for DLAS is very restrictive. Furthermore, the data location assurance generated against a user's request does not reflect the current data location status; rather it can be up to 24-hour old status. Recall that CSP sends location information to the enterprise for all of its clients' data once in a day (maybe at 12:00 am every day) and the data location verification phase is based on that data).

This Hardware-based Data Location Assurance Solution (HDLAS) can be considered as an improvement over the DLAS solution as it tries to mitigate the above limitations of the DLAS solution. Our proposed HDLAS solution is a 2-entity based solution which is able to cover almost all existing system models and works for a more adversarial attack model than that of DLAS.

Objective: The main objective of this HDLAS solution is to come up with a 2-entity based solution which is able to support all existing cloud storage applications. Trusted Platform Module (TPM) and a cryptographic scheme called Proof of Data Possession (PDP) are the basis of our solution. This HDLAS solution can be considered as an improvement over the DLAS solution. Similar to DLAS, HDLAS also provides geo-location assurance of user data for the geo-replication of data setting. But unlike DLAS, it is able to cover almost all existing cloud storage applications and considers a new attack model which is more adversarial than that of

DLAS and practical for the existing cloud storage scenario. Furthermore in contrast to DLAS [15] [98], HDLAS is able to offer its clients not only accurate geo-location of their data through the use of GPS receiver but also hardware-based root of trust for that.

4.2 System and Attack Model

System Model: For this proposed solution, we consider a 2-entity system model which covers essentially all service / business models of cloud computing that exist today. In our system model, a client can directly communicate with the CSP (Cloud Storage Provider). For our proposed solution, we consider a realistic Cloud Storage Provider which might have a number of datacenters and each datacenter might have a number of storage servers.

Attack Model: For our proposed solution, we define a new attack model which appears to be very practical for existing cloud storage applications. More precisely, the CSP (and their storage servers) will normally follow the protocol correctly; however, in case of incidents such as accidental data loss or moving data to a more economic storage server (in a different region), if a cloud user sends a data geo-location query to the CSP, the CSP then may try to hide this fact from the client (and thus act maliciously at that moment), perhaps for a short period of time until it has fixed the issue. We name this attack model "occasionally-malicious-but-constrained". The reason for giving such a name is that in our proposed solution, whenever the CSP misbehaves, it will be caught.

Assumptions: We assume that each storage server has a Trusted Platform Module (TPM). This is a reasonable assumption, as at present we see the use of TPMs in laptops and desktops; most importantly, the use of TPM is getting popular in servers [32]. This TPM will ensure the correct execution of certain processes through attestation. More precisely, we assume that a TPM either has an inbuilt GPS receiver or, through attestation, it is able to ensure the correct execution of a remote GPS receiver; thus by using TPM, the storage server is able to offer hardware root of trust to its clients regarding the accurate geo-location of their data. Furthermore, through remote attestation, TPM also ensures the correct execution of the necessary PDP processes run by each storage server.

4.3 Cryptographic Primitives Used

Trusted Platform Module (TPM) and Proof of Data Possession (PDP) are the main building blocks of the HDLAS solution. The reason behind using TPM in our proposed solution is that TPM has the capability to do integrity measurement and remote attestation. On the other hand, proof of data possession (PDP) is a cryptographic scheme executed between a client and a server where the server can prove to the client that it did not tamper with its data. We use the PDP scheme proposed in [76] as it is one of the most popular PDP schemes and this PDP scheme is not only able to offer provable security but is also suitable for practical deployment. This PDP scheme is a collection of four polynomial-time algorithms: *KeyGen, TagBlock, GenProof and CheckProof***.** Figure 8 includes the details of those 4 algorithms of this PDP scheme

4.4 Overview of the Proposed Solution: HDLAS

Let a Cloud Storage Provider (CSP) have a number of datacenters; each datacenter has one or more storage servers and each storage server has a TPM. Now when a client wishes and signs a contract to take data storage service from a cloud storage provider (CSP), a CSP has to commit/publish all storage servers, and the corresponding public keys of the TPMs attached to those storage servers, to the client. Furthermore the CSP also issues a client id to the client.

Actual data storage service starts with a client's data storage request to the CSP which includes client ID, client's data storage location preference, R (perhaps in terms of a city/country/continent) and the data/file itself. Upon receiving the storage request, the CSP (actually a process run by the CSP) generates a data ID, dataID (e.g., by calculating the hash of the file) and chooses one or more storage servers (a list of $storageserverID$) from one or more datacenters which satisfy the client's data location preference list, R, and asks those storage servers to store that piece of data (e.g., a file F). Let $datalocationDB$ be the database the CSP manages to keep storage record of each piece of data; so in a typical scenario, upon getting storage confirmation of that piece of data ($dataID$) from each storage server, the CSP will add a tuple such as $< dataID, storageserverID >$ in $datalocationDB$. The CSP then sends the storage confirmation to the client which essentially includes the $dataID$.

Let f be a pseudo-random function, π be a pseudo-random permutation, and H be a cryptographic hash function

KeyGen(1^k):

Generate $pk=(N,g)$ and $sk=(e,d,v)$, such that

$ed \equiv 1 \pmod{p'q'}$, e is a large secret prime such that

$e > \lambda$ and $d > \lambda$, g is a generator of QRN and $v \xleftarrow{R} \{0,1\}^K$.

Output (pk, sk).

TagBlock(pk, sk, m, i):

1. Let $(N,g) = pk$ and $(d,v) = sk$. Generate $W_i = v \,||\, i$.
   Compute $T_{i,m} = (h(W_i) \cdot g^m)^d \bmod N$.
2. Output $(T_{i,m}, W_i)$.

GenProof(pk, $F = (m_1, \ldots, m_n)$, $chal$, $\sum = (T_{i,m_1}, \ldots, T_{n,m_n})$):

1. Let $(N,g) = pk$ and $(c, k_1, k_2, g_s) = chal$.
   For $1 \le j \le c$:
   - compute the indices of the blocks for which the proof is generated: $i_j = \pi_{k1}(j)$
   - compute coefficients: $a_j = f_{k2}(j)$.
2. Compute $T = T_{i1,mi1}^{a_1} \cdot \ldots \cdot T_{ic,mic}^{a_c} =$
   $(h(Wi1)^{a_1} \cdot \ldots \cdot h(Wic)^{a_c} \cdot g^{a_1 m_{i1} + \ldots + a_c m_{ic}})^d \bmod N$
   (note that Ti_j, mi_j is the i_j-th value in $\sum$).
3. Compute $\rho = H(g_s^{a_1 m_{i1} + \ldots + a_c m_{ic}} \bmod N)$.
4. Output $V = (T, \rho)$.

CheckProof(pk, sk, $chal$, V):

1. Let $(N,g) = pk$, $(e,v) = sk$, $(c, k1, k2, s) = chal$ and $(T,\rho) = V$.
2. Let $\tau = T^e$. For $1 \le j \le c$:
   - compute $i_j = \pi_{k1}(j)$, $Wi_j = v \,||\, i_j$,
     $aj = f_{k2}(j)$, and $\tau = \dfrac{\tau}{h(Wi_j)^{a_j}} \bmod N$

   (As a result, one should get $= g^{a_1 m_{i1} + \ldots + a_c m_{ic}} \bmod N$)
3. If $H(\tau^s \bmod N) = \rho$, then output "success".
   Otherwise output "failure"

Figure 8 The original PDP scheme proposed by Ateniese et al.

Once the client wants to verify its data location, it sends data location request for a piece of data, $dataID$ to the CSP. If the client's query is authentic, the CSP (i.e., a process run by the CSP) searches in $datalocationDB$ for $dataID$ and returns a set of $storageserverID$ as search result.

For each storageserverID, the client launches the PDP scheme with the storage server and finds out whether the piece of data is still there. Furthermore the client will ask the storage server to give a signed version of its GPS reading and upon getting the reply from the storage server, a client will be able to determine the exact geo-location of the storage server and thus of its data. [Note that the use of TPM ensures the correct execution of important processes (e.g., processes related to PDP and GPS) run by the CSP and storage servers.] In this way, the client is able to find out whether the CSP has violated the client's data location preference or not.

4.5 Hardware-based DLAS Scheme-Formal definition

Here we formalize the notion of HDLAS, with the system and attack model outlined in section 3.2.

DEFINITION 1. A Hardware-based DLAS scheme consists of 3 phases.

Phase 1- Bootstrapping Phase. This phase consists of 2 algorithms:

1. **Publish_storageServersPublicKey(SSID)→(SSID_pubKey)**: is run by the CSP. Here for each storgareserverID in SSID(a list of all storage servers IDs of a CSP), the CSP commits and publishes the public key of the TPM attached to each storage-server to the client or make them publicly available so that later on a client can correctly identify a storage server and may establish secure communication with it, if needed.

2. **Issue_clientID (client_info) →(clientID):** is run by the CSP. When the client finalizes to take the storage service, the CSP issues a clientID from client_Info. In a simple case, it could be the client's email address she provided to the CSP.

Phase 2- Client Data Registration Phase. This phase consists of 4 algorithms. This phase starts with a data/file storage request with a data storage location preference list, R, made by a client to the CSP; this results in the file to be stored in one or more of the storage servers, located in one or more datacenters, of the cloud storage provider. Similar to the definition of our previous solution, DLAS [15] [98], here R also represents a list of city/country/continent where the client (does/does not) want to store its data.

1. **FileEncoding_PDP (F) → (F, ∑):** Here the client executes 2 algorithms of the PDP scheme to do the file encoding. First it executes the KeyGen(1^k) algorithm to generate the public-private key pair <pk, sk>. It then executes the TagBlock (pk, sk, m_i) algorithm of the PDP scheme to generate verification metadata, ^Tm_i for each block of the file, F where $1 \leq i \leq n$. The client then stores <pk, sk> and sends < pk, F, ∑ > along with its storage location preference, R(in terms of city/ country/ continent) to the CSP where ∑ represents all verification metadata of F, i.e., $\sum = (^Tm_1\ ^Tm_2\ \ldots\ ^Tm_n)$.

   Client → CSP: < pk, F, ∑, R >

2. **Gen_dataID (F) → (dataID):** is run by the CSP. It generates a dataID for the data, F. It could be generated using a hash function.

3. **FindStorageServers(R, Ω) → SSID_res:** is run by the CSP. It takes R (client's data location preferences) and Ω (internal storage policy) as inputs, and outputs a set of storage servers IDs, SSID_res (i.e., SSID_res ⊆ SSID) where the client's file will be stored.

4. **Build_and_Manage_datalocationDB (dataID, SSID_res) → datalocationDB**: is run by the CSP. Upon successfully having stored client's file into appropriate storage servers (i.e., SSID_res), the CSP inserts <dataID, storageserverID> tuple (for each storageserverID in SSID_res) into the database, datalocationDB. The CSP then sends the dataID back to the client.

   CSP → Client: dataID

Phase 3- Data Location Verification Phase. This phase consists of 3 algorithms. This phase starts with a data location verification request made by the client to the CSP and ends with producing a verifiable assurance regarding the client's data location.

1. **Retrieve_storage_servers_IDS → SSID_res'**: Here a client sends a data location query for a piece of data to the CSP.

   Client → CSP: "Tell me all the storage server IDs that have my data, dataID, at the moment."

   The CSP then searches on the database, datalocationDB for that dataID and sends SSID_searchresult (a list of storageserverIDs) back as the search result to the client.

   Client → CSP: SSID_ searchresult

   Note that the correctness of the search result, SSID_searchresult is ensured through TPM's remote attestation procedure to the clients.

2. **Verify_Data_Possession_from_storageserver():** is run by both the client and storage server. A client initiates this step to cryptographically validate whether its data is still intact in those storage servers (SSID_ searchresult). In order to do that, for each storageserverID in SSID_ searchresult, the client first establishes a secure communication link with the storage server (storageserverID) and then launches the PDP scheme with that storage server. It consists of the following 3 steps:

   a) *Generate challenge***:** The client first generates a challenge, chal that among other things indicates the specific blocks (i.e. a block number of the file, F) for which the client wants a proof of possession from the storage server. The client then sends it to the storageserverID:

   Client → storageserverID: chal

   b) *Generate proof*: Upon receiving the challenge, chal, the storage server (storag-eserverID) executes the GenProof (pk,F,chal,$\sum$) algorithm of the PDP scheme and generates the proof, V as output. The storage server then sends V to the client:

   storageserverID → Client: V

c) *Validate proof*: Upon receiving V from the storage server, the client runs the Check-Proof(pk, sk, chal, V) algorithm of the PDP scheme to validate V. If V is the correct proof, CheckProof() will output "success", and "failure" otherwise..

The client can continue the above steps for any number of challenges (e.g., *n* times if the client wants to check proof of possession for n blocks of the file) with a particular storage server. If for all challenges it receives success as output, the client will be convinced that its data (dataID) is still intact (i.e., not corrupted) in that particular storage server (i.e., storag-eserverID).

3. **GetAccurateDataLocation_from GPS (SSID_searchresult, dataID, R)** → is run jointly by the client and (each) storage server. For each storageserverID in SSID_ searchresult, the client first establishes a secure connection with the storage server (storagerserverID) and then asks for its (i.e., storage server's) current GPS reading. The storage server then takes its GPS reading, signs it with its private key (i.e. the attached TPM's private key), and sends it to the client. Note that this GPS reading cannot be tampered with as its correctness is ensured by the TPM through integrity measurement and remote attestation.

The client then verifies the signature using the corresponding public key it received in the bootstrapping phase and thus finds out the authenticity of that GPS reading. If the GPS reading of all storage servers have been found as authentic, the client can then decide the exact geo-location of all storage servers that have its data, dataID, and know whether its data location preference, R has really been met by the CSP.

Note that the order of algorithms 2 and 3 in Phase 3 is not important; one can first check the geo-location of storage servers that have the data(i.e., dataID) and then launch the PDP scheme with all the storage servers one after another to identify whether they really have the data at the moment.

4.6 How to Construct Hardware-based DLAS

Hardware-based DLAS construction for our system model (i.e., the 2-entity system model where a client directly communicates with the CSP), is very straightforward. In that

case, all 3 phases of the Hardware-based DLAS definition have to be executed exactly as they are defined in definition 1.

4.6.1 Adopting Hardware-based DLAS in Microsoft Azure:

First we briefly discuss Microsoft Azure (previously known as Windows Azure [30]) to identify the key facts regarding its data storage operation and then propose the changes that need to be made in Microsoft Azure to support Hardware-based DLAS.

Data Storage Location and Data Movement in Microsoft Azure: In Microsoft Azure [8] [30], there is a central server/engine called Location Service (LS) which is responsible for keeping track of which data is located in which storage servers (in their term, "storage stamps"). Microsoft Azure does geo-replication of data and it is also handled by the LS. In Microsoft Azure, any data storage request made by a client first comes to LS and then the LS will determine 3 storage stamps (one of these should be the primary one) for the data storage and instructs those storage stamps to store that data. It also chooses a storage stamp for data geo-replication purpose. Note that LS will be notified if there is any movement of data from one storage stamp to another.

Necessary Changes: From the above discussion, it is obvious that LS has already been managing a database like datalocationDB. In order to adopt Hardware-based DLAS in Microsoft Azure, only a trivial change needs to be done which is adding TPM in LS and all the storage servers of Microsoft Azure. [Even though we did not find any evidence in [8], it might be the case that Microsoft Azure has already been using TPM in its storage stamps since now the inclusion of TPM is quite common in servers due to its ability of verifying the correctness of any processes through integrity measurement and remote attestation]. Furthermore, Microsoft Azure has to publish the public keys of all of its storage stamps (actually the public key of the attached TPMs).

With the above changes, Microsoft Azure is now ready to offer Data Location Assurance Service to its client by exactly following definition 1.

4.7 Discussion & Analysis

4.7.1 Security Analysis of Hardware-based DLAS

TPM and PDP are the two building blocks used for ensuring security in Hardware-based DLAS. Due to the integrity measurement and remote attestation services provided by TPM, a CSP/storage server cannot tamper with the GPS reading of the storage server and the search result of the database, datalocationDB (in response to a client's search query for a piece of data). So the only reasonable attack in our Hardware-based DLAS would occur when a piece of data is either corrupted or not in a storage server where it should be and yet the storage server is able to convince the client about the possession of that data. It is only possible if during the PDP scheme execution with the client, the (occasionally malicious) storage server is able to generate a false proof of possessing a data that it does not have at the moment. In other words, it is similar to breaching the security of the PDP scheme. In order to discuss the feasibility of such attack and thus to analyze the security of the PDP scheme, as in [76], we also formalize the security for a PDP system using a game that captures the data possession property (definition 2). Intuitively, the Data Possession Game captures that an adversary cannot successfully construct a valid proof without possessing all the blocks corresponding to a given challenge, unless it guesses all the missing blocks.

DEFINITION 2. Data Possession Game:

Setup: The challenger runs (pk, sk) ← KeyGen(1^k), sends pk to the adversary and keeps sk secret.

Query: The adversary makes tagging queries adaptively: It selects a block m1 and sends it to the challenger. The challenger computes the verification metadata T_{m1} ← TagBlock(pk, sk, m1) and sends it back to the adversary. The adversary continues to query the challenger for the verification metadata T_{m2} , . . . , T_{mn} on the blocks of its choice m2, . . . ,mn. As a general rule, the challenger generates T_{mj} for some $1 \leq j \leq n$, by computing T_{mj}← TagBlock(pk, sk,mj).

The adversary then stores all the blocks as an ordered collection F = (m1, . . . ,mn), together with the corresponding verification metadata T_{m1} , . . . , T_{mn}.

Challenge: The challenger generates a challenge chal and requests the adversary to provide a proof of possession for the blocks mi_1 , . . . ,mi_c determined by chal, where $1 \leq i_j \leq n$, $1 \leq j \leq c$, $1 \leq c \leq n$.

Forge: The adversary computes a proof of possession V for the blocks indicated by chal and returns V. If CheckProof(pk, sk, chal, V) = "success", then the adversary has won the Data Possession Game.

Based on the Data Possession game (Definition 2), now we formalize the security of the PDP scheme as follows:

DEFINITION 3: A PDP scheme (KeyGen, TagBlock, GenProof , CheckProof) guarantees data possession if for any (probabilistic polynomial-time) adversary A the probability that A wins the Data Possession Game on a set of file blocks is negligibly close to the probability that the challenger can extract those file blocks by means of a knowledge extractor E.

In case of Hardware-based DLAS, storage server plays the role of the adversary, A. Note that it can be formally proved that an adversary (i.e. the storage server) cannot win the data possession game and breaking this scheme would be as difficult as breaking the RSA (i.e. factoring a large prime) and KEA1-r assumptions [76] (which are known to be hard problems).

4.7.2 Implementation and Performance Issues

TPM and the PDP scheme are the main building blocks for Hardware-based DLAS. TPM is mainly used for ensuring the correctness of the GPS reading through remote attestation. NIST gives a step-by-step discussion of a prototype implementation of TPM in a cloud environment to check and validate the original location of a resource (e.g., a process) running

on that server [95]. It will be very helpful for any prototype implementation of Hardware-based DLAS.

Unlike TPM, the PDP scheme is very critical to the performance of this proposed solution. However, the PDP scheme has been proven to be very practical for real deployments. Now we consider a concrete example to analyze the performance numbers of the PDP scheme. Let F be a 4GB file which has 1000000 4KB blocks. If the storage server deletes 1% of F, the client can detect the misbehaviour of the storage server with probability over 99% by asking proof for only 460 (in the worst case) randomly selected blocks. [76] includes a detailed discussion of the performance numbers for their prototype implementation of the PDP scheme which indicates that the processing/computation time of the PDP scheme is very reasonable (e.g., building proof of possession with 99% confidence for any file up to 64MB takes only 0.4 seconds) and thus it is suitable for practical use.

Note that in the PDP scheme, most of the computation has been done by the client, proof generation is the only task done by the storage server. Similar to our previous DLAS solution [15] [98], a resource constrained client can also take IAAS service from the cloud to do all processing tasks involved in the PDP scheme.

4.8 Conclusion

In this chapter, we have proposed Hardware-based DLAS, a complete data location assurance solution for cloud storage environments which is suitable for almost all existing cloud storage deployment models available today. Our proposed solution is mainly based on TPM and a cryptographic scheme called Proof of Data Possession. It facilitates a cloud user to get accurate geo-location of a piece of data through technical (and reliable as well) means as here the geo-location of data is coming from a GPS receiver (whose correct execution is ensured by TPM through remote attestation). Like our previous solution, DLAS [15] [98], Hardware-based DLAS also allows users to give their data location preference in the following way: “Keep/ do not keep my data in region R (e.g., R can be a city, country or continent)”; later, users will receive provable assurance of their data location from the CSP (Cloud Storage

Provider) with respect to their data location verification request. Whereas most of the existing solutions [21] [67] [23] [16] [64] [95] do not work in the case of geo-replication of data in different storage servers; like our previous solution, DLAS, Hardware-based DLAS, works even if a piece of data is stored/copied in multiple storage servers and thus it supports the geo-replication practice adopted by many existing CSPs (e.g., Microsoft Azure). In contrast to our previous solution, DLAS [15] [98], Hardware-based DLAS has less overhead in terms of both computation and communication. In this chapter, we also discussed how easily Hardware-based DLAS can be adopted into existing cloud storage providers such as Microsoft Azure. We believe that Hardware-based DLAS also has the potential to be used as one of the building blocks of the "accountable cloud [102]".

5 Proposed Solution 3: A PDP Solution for Cloud Storage Environments

In this chapter, we present one of the main contributions of our research. Here we propose the first Proof of Data Possession (PDP) solution which is specifically built for the cloud storage environments. We also discuss how a DLAS solution can be constructed using our proposed PDP solution.

5.1 Motivation & Objective of this Solution

Motivation: Our previous 2-entity based HDLAS solution discussed in the previous chapter, is mainly based on TPM and PDP. Whereas a PDP scheme is able to offer a verifiable data possession proof by the storage server, one of the main roles of the TPM is to bind the execution of PDP scheme to a specific storage server, as existing PDP schemes do not cryptographically bind the proof generation process to a specific storage server. Definitely this is an additional load to the TPM. We want to come up with a DLAS solution that does not require a TPM to do that additional work. For this reason, we want to come up with the first PDP scheme that can also offer the data binding guarantees (proof of data possession is bound to a specific storage server) and thus can be easily deployed in the Cloud Storage Environment.

Objective: Our main objective is to propose an efficient PDP solution is able to meet the need of existing cloud storage environments. More precisely, this PDP solution should be able to bind the data possession proof to a specific storage server. This PDP solution has the potential to be used in building the DLAS solution for a more adversarial system and attack model.

Note that this PDP solution is intended for the storage of archival/static data where a client normally intends to keep its data for a long duration of time.

5.2 System and Attack Model

5.2.1 System Model

Similar to existing cloud storage providers such as Amazon S3 and Microsoft Azure, we consider a cloud storage environment which might have a number of datacenters; each datacenter might have a number of storage servers. In addition, we assume that among those storage servers, there will be at least one (can be more than one) storage server in each data center which will be responsible for performing PDP operations with the client. We name these storage servers as "Authoritative Storage Server". Those authoritative storage servers have a unique public-private key pair and the public key is known to all clients. One concrete example of it would be assuming that all authoritative storage servers have a TPM with Elliptic curve cryptography (ECC) support. As a simple realization of our system model, we can assume that each of those authoritative storage servers has a TPM attached to them. The main purpose of the TPM is to offer non-repudiable identity guarantees to those servers as a TPM comes with a built-in <public, private> key pair. Moreover, in order to offer the PDP service to the clients, it is the responsibility of the Cloud Storage Provider (CSP) to publish the location information of each authoritative storage server; in other words, a CSP has to publish which authoritative server is located in which region. Furthermore a CSP has to make the public keys of those authoritative storage servers accessible/available to its clients. Note that, an authoritative storage server can also be called as an "*authoritative location server*"; it is because we use the authoritative server's location information (more precisely, a server's bindings to a particular region) as the key factor to offer the data location guarantee.

We name this system model as "*DLASSystemModel3***"** and it has the following properties:

- Similar to our previous HDLAS solution, this system model is also a 2-entity based model, where a user tries to get verifiable data possession guarantees about a piece of its data from an authoritative storage server by directly launching the PDP protocol with the (authoritative) storage server.

- Each authoritative storage server has an inbuilt TPM. The main purpose of this TPM is to bind its in-built public-private key pair to the attached server.
- The public-private key-pair has to be generated from Elliptic Curve Crypto and thus TPM should have support for Elliptic Curve Crypto. Aside from performing regular in-built operations (such as hash, HMAC, and public key crypto), TPM 2.0 [33] supports 256-bit Elliptic Curve Cryptography.

Note that our proposed PDP solutions only requires the authoritative storage servers to have a uniquely identifiable and verifiable <public, private> key-pair. Considering the TPM for offering <public, private> key-pair is a simplistic (and yet practical) approach to realize the system model; however, it might not be the only solution. As long as the authoritative storage server has an elliptic-curve based <public, private> key-pair, our proposed solution will work.

- **Supports Distributed Data Storage:** Similar to existing CSPs such as Amazon S3 and Microsoft Azure, this system model also supports the practice of distributed storage of data where different parts of data can be stored on different storage servers/physical disks. In case of distributed storage of data, an authoritative storage server responsible for a particular data center and a specific client, can distribute the client's data into a number of storage servers/physical disks located in the same datacenter. However, that authoritative storage server has to keep track of it and should be able to fetch the data as if it is located on its own physical disk. From the client's point of view, the client is oblivious about the distributed storage of data; the only thing the client wants from the PDP scheme is to verify whether the authoritative storage server is able to provide a verifiable proof of having the client's data and the authoritative storage server needs to access that data to generate the data possession proof. We assume that due to the physical proximity between an authoritative storage server and other storage serves (where actual distributed data storage took place), data transfer among those servers takes a negligible amount of time and thus we will ignore this during the construction of this proposed PDP solution.
- This PDP solution is intended for the storage of archival/static data where a client normally intends to keep its data for a long duration of time.

- Our proposed PDP solution puts no restriction on the format of the data; our PDP solution can be applied on both encrypted and un-encrypted data.
- In our system model, data owner and data verifier are the same entity. In other words, it does not support public verifiability.

Figure 9 shows a pictorial representation of our system model.

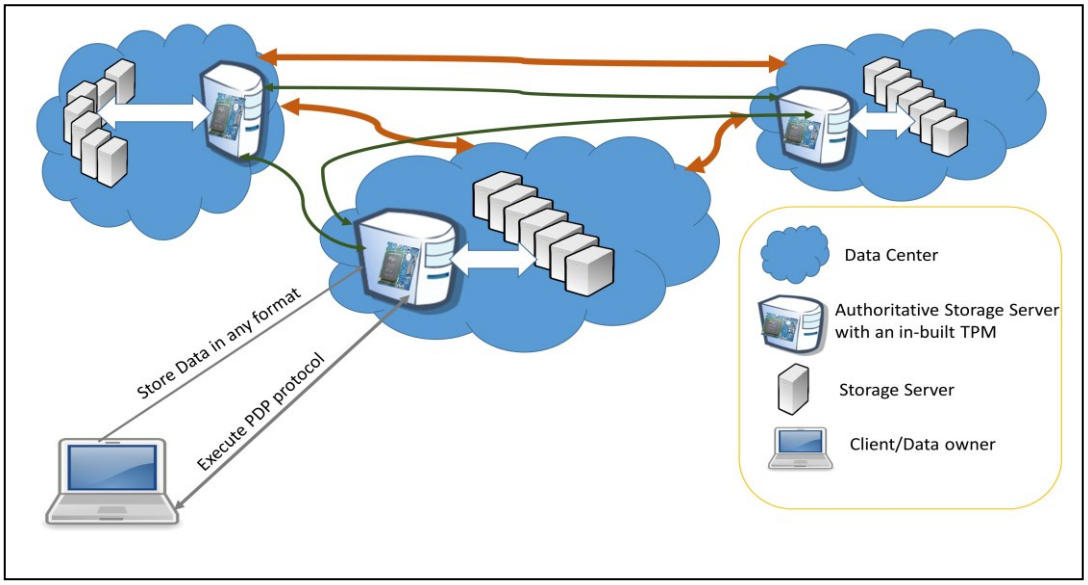

Figure 9 System Model for Proposed PDP Solution

5.2.2 Attack Model

For our proposed PDP solution, we define a new attack model which appears to be very practical for existing cloud storage applications. More precisely, the CSP (and the storage servers) will normally follow the protocol correctly; however, in case of incidents such as accidental data loss or moving data to a more economic storage server (in a different region), if a cloud user sends a data geo-location query to the corresponding authoritative storage server, the server then may try to hide this fact from the client (and thus act maliciously at that moment), perhaps for a short period of time until it has fixed the issue. Our attack model also considers the fact that, in order to achieve a false proof, the authoritative server can even work with the other authoritative servers to generate a false proof without possessing the data at that moment. We name this attack model "distributed occasionally-malicious-but-constrained". Formally this attack model can be defined as follows:

Definition of distributed occasionally-malicious-but-constrained Attack Model: An authoritative storage server will correctly follow the protocol but might communicate with other storage servers to generate a false proof when it does not have the data (or user data is corrupted).

Remark 1. This attack model is specifically suitable for the cloud storage environment where multiple storage servers are present. Furthermore this solution is proposed for the storage of static data where a client normally intends to keep its data for a long duration of time. So aside from accidental damage of the data, we are mainly considering economically-motivated CSPs that may attempt to (i) keep the client data in a cheaper location (i.e., storage server) instead of the places where the client wants it to be stored, (ii) store the data offline on physical disks assuming that those data will not be queried by client as this archived data is stored for archival purposes.

5.3 Limitations of Existing PDP schemes: Why do we need a new PDP solution for the cloud?

Proof of Data Possession schemes (and solutions of this kind such as Proof of Retrieval) were primarily proposed with a single un-trusted storage server in mind where the purpose of the PDP solution is to ensure that a user can get verifiable proof from the un-trusted storage server regarding the well-being of its outsourced data (i.e., the data integrity is intact and the server still has the data). Until today, a number of PDP schemes (e.g., [72] [73] [76] [77] [78] [79] [80] [81] [82] [83] [85] [86] [87] [88] [104]) have been proposed based on different cryptographic primitives (e.g., using RSA and homomorphic verifiable tag, Pairing-based crypto and so on), where attempts have been made to either optimize the performance or add more functionality such as supporting public verifiability, dynamic updates of the stored data and so on. However, all existing PDP schemes still consider the same single un-trusted storage server; none of them consider the presence of multiple storage servers which are able to communicate with each other. We identified the following reasons which we believe make it impossible for an existing PDP scheme to offer data possession guarantees by a specific storage server (or data center) in the cloud storage environment:

- Existing PDP schemes do not bind the data possession proof to the storage server (recall that it was built considering one un-trusted storage server in mind; as a result there was not a need for that). However, in cloud storage environments, there would be many storage servers (and many data centers) which are able to communicate with each other. As a result, in case of cloud storage environment, existing PDP schemes can offer only data possession guarantees by the cloud as a whole but not from a specific storage server/data center.
- In most of the well-known PDP schemes (e.g., [76] [77] [78] [79] [80] [81] [82] [83] [85] [86] [87] [88] [104]), the data possession proof generation phase does not make it mandatory for the storage server to deal with EACH single piece of challenged data; rather that proof can also be generated for example, from the sum ([76]) of all challenged pieces of data [see step 3 of the $GenProof()$ of Figure 8].

In a more adversarial system and attack model (e.g., in cloud storage environments where a storage server can communicate with other storage servers), it is possible for a storage server to communicate with other storage servers in order to generate a false data possession proof even if it does not have the data. For instance in case of the PDP scheme proposed in [76], any one of the storage servers (currently having the data) can generate the proof (i.e., performing step 2 and 3 of the $GenProof()$ of Figure 8) and pass it to the server to which the user asks for the data possession proof. Note that similar to this PDP scheme [76], almost all the existing PDP schemes have the same vulnerabilities.

5.4 Defining new requirements of a PDP Solution for cloud storage environments

Based on our findings in regards to the limitation of existing PDP solutions, we identify the following properties a PDP solution should have, in order to be used in the cloud storage environment:

- The proof generation process should bind all challenged data to the storage server generating the proof. This data binding can be achieved by forcing the server to use some knowledge (e.g., preferably the server's private key) that only the server knows during generating the data possession proof.

- The proof generation process must require the server to do processing for EACH single challenged data.
- The server should also offer non-repudiation guarantees regarding their generated data proof.
- The solution should not depend on the type of the data whether it is encrypted or not; it should work irrespective of underlying data type.
- The computation involved on the user side should not be expensive; it should be possible to do that computation even on a mobile device.

5.5 Proposed PDP Solution

5.5.1 Notation & Preliminaries

- F – outsourced data. We assume that F can be represented as a single contiguous file of d equal-sized data blocks:$< m_1 m_2 m_3 \dots m_i \dots m_d >$.The actual bit-length of a block is not germane to the scheme.
- c – an integer which represents the total number of randomly chosen data blocks involved in a single PDP challenge
- $Client$– the owner of the data.
- $Server$ – an authorative storage server of the cloud storage environment. Each Server has a unique $ServerID$ assigned by its CSP
- X – an integer which represents the total number of PDP challenges the client wants to have for the entire lifespan of the outsourced data.
- $H()$-cryptographic hash function. In practice, we use standard hash functions, such as SHA-2, SHA-3, etc.
- $H_k()$ – cryptographic HMAC function (i.e., a keyed hash function).
- $AE_k()$- an authenticated encryption scheme, that provides both privacy and authenticity. In practice, privacy and authenticity are often achieved by encrypting the message first and then computing a Message Authentication Code (MAC) on the result. However, a less expensive and easily available alternative is to use a mode of operation for the cipher that provides authenticity in addition to privacy in a single pass, such as GCM, OCB, XCBC, IAPM [55].

- $AE_k^{-1}()$ - decryption operation for the scheme introduced above.
- $\pi_k()$ – pseudo-random permutation (PRP) indexed under key. AES is assumed to be a good PRP. To restrict the PRP output to sequence numbers in a certain range, one could use the techniques proposed by Black and Rogaway [105]. The role of this is to randomly generate an index from the range$[1, d]$. It can be defined as:

  $\pi: \{0,1\}^l \times \{0,1\}^l \rightarrow \{0,1\}^l \; where \; l = \log d$
- $< K_S, k_s >$ – Server's <public-private> key-pair generated from elliptic curve domain parameters $<a, b, G, n>$
- K_U – a secret key for the $AE_k()$ scheme generated by the Client
- k_1 – a randomly generated key for the PRP, $\pi_k()$
- S_x, R – elliptic curve points where S_x served as the shared secret between $Server$ and $Client$ and R is a random elliptic curve point.
- $hmac_i - HMAC$ computed on the data block m_i. This is the verification metadata computed on a single block
- T_{xor} – result of applying xor operation on all c number of verification metadata involved in a single challenge

  i.e., $T_{xor} = hmac_1 \; xor \; hmac_2 \; xor \ldots hmac_i \; xor \ldots xor \; hmac_c$
- T_x – Encrypted verification metadata for a single PDP challenge $x \; where \; 1 \leq x \leq X$; among other things it includes T_{xor} , x, R, k_1 , $ServerID$
- T_{sign} – A digital signature computed on T_{xor} by $Server$

5.5.2 General Idea

Similar to the well-known PDP scheme [76] [85], our proposed PDP scheme is also a probabilistic solution where for each challenge a user will ask the server to generate a data possession proof using a subset of randomly chosen data blocks from F. Similar to [85], in this solution, $Client$ needs to also choose in advance a total number of PDP challenges s/he is going to query to the $Server$ for the entire lifespan of the outsourced data.

In our solution, verification metadata has been computed using HMAC, which is considered a very fast crypto operation. The key aspect of this solution is to facilitate both $Client$ and $Server$ the ability to generate a shared secret between them which would be ultimately

used as the HMAC key. The shared secret used in this solution, is essentially an elliptic curve point which is later on transformed into a HMAC key. $Client$ is the one who generates this shared secret for the first time and shared necessary information to the server so that only $Server$ is able to generate the same shared secret when it is needed.

As noted earlier, our proposed PDP solution binds the data possession guarantees to a specific authoritative storage server, not the entire cloud. And the usage of this shared secret in verification metadata generation is our main trick to achieve so. Along with it, in our solution, Server also signs the data possession proof it generates for a particular PDP challenge and thus our solution also offers a non-repudiation guarantee.

5.5.3 Definition of PDP solution for cloud storage environments

Our PDP definition follows the same approach as in the PDP definition of [76] .

Definition 1. A PDP (Proof of Data Possession) scheme is a collection of four polynomial-time algorithms ($KeyGen, GenTags, GenProof, CheckProof$) such that:

$\boldsymbol{KeyGen}(\mathbf{1}\boldsymbol{k})$ is a probabilistic key generation algorithm that is run by both $Server$ and $Client$ to set up the scheme. Whereas $Server$ generates and publishes elliptic curve domain parameters and its public key (from a chosen secret private key), $Client$ generates a symmetric key.

$\boldsymbol{GenTags}$ $(K_S, \boldsymbol{F}, \boldsymbol{X}, \boldsymbol{c}, \boldsymbol{k_1})$ is a (possibly probabilistic) algorithm run by the client to generate the verification meta-data. It takes as inputs the server's public key K_S, the database/file, **F** split into d equal blocks, a secret key $\boldsymbol{k_1}$ for the PRP function, **X** -total number of challenges for which the client wants to generate the verification metadata and c-number of data blocks involved in each single challenge, and returns the verification metadata T for the entire file.

$\boldsymbol{GenProof}(\boldsymbol{k_S}, \boldsymbol{F}, \boldsymbol{x}, \boldsymbol{c}, \boldsymbol{k_1}, \boldsymbol{R})$ is run by the server in order to generate a proof of possession for a given challenge $x\ where\ 1 \leq x \leq X$. It takes as inputs the server's private key $\boldsymbol{k_S}$, an ordered collection **F** ($< m_1 m_2 m_3 \ldots m_i \ldots m_d >$) of blocks, a challenge number- **x**, **c** - number of data blocks, $\boldsymbol{k_1}$ - a secret key provided by the client for finding all the data blocks

involved in the challenge, and **R** - a random elliptic curve point provided by the client. It returns a signed version of the proof of possession for the file- F that is determined by the challenge $\boldsymbol{x}$.

$\boldsymbol{CheckProof}$ (T_x,T_{sign}, K_U) is run by the client in order to validate a proof of possession. It takes as inputs K_U (the symmetric key known only to the client to decrypt T_x and extract Txor), the verification metadata computed earlier by the $Client$ using $GenTags()$ algorithm for a given challenge, and a signed proof of possession T_{sign}. It returns whether the proof of data possession for the challenge is correct and was actually generated by the intended server or not.

A PDP system can be constructed from a PDP scheme in two phases: Setup and Challenge. Whereas the $KeyGen()$ and $GenTags()$ algorithms are executed in the Setup phase, the latter two algorithms ($GenProof()$ and $CheckProof()$) are executed in the $Challenge()$ phase.

Setup: The Client and Server first executes the $KeyGen()$ algorithm to generate necessary keys. The client then chooses X, the total number of challenges for the entire lifespan of the file F. For instance X can be easily computed as

$X = 365 * No\_of\_Years * No\_of\_Challenge\_Per\_Day$.

The client then runs $KeyGen()$, followed by $GenTags()$ to generate verification metadata, T, for all the challenges(X) for the data file F. The client then sends (F, T) to the Server for storage and deletes F and T(verification metadata) from its local storage.

Challenge: The client sends a challenge request of proof of data possession to the server, among other things, which indicates a subset of random data blocks of F for which it wants a proof of data possession. The server then runs the $GenProof()$ algorithm and sends back the output of the $GenProof()$ algorithm as the signed proof of possession for that challenge to the client. Finally, the client executes the $CheckProof()$ algorithm to validate whether the proof is a correct proof and it was actually generated by the intended server.

Figure 10 includes the definition of our PDP scheme along with all 4 algorithms.

Notations: *Let π be a pseudorandom permutation, $\mathcal{H}$ be a cryptographic HMAC function, Sym be a symmetric encryption, a file $F =< m1\ m2 \ldots mi \ldots m_d >$ be divided in to d equal blocks, $k1$ - a key generated by the user with a A PRF which will be fed in $\pi()$, c be the the total number of randomly chosen data blocks needed for each data possession challenge*

DEFENITION: This PDP scheme consists of the following ALGORITHMS:

KeyGen *(1^k):*

Server generates and publishes: Elliptic curve dOmain parameters:(p, a, b, G, n, h) and Server's public key, $KS = k_s G$ where k_s is Server's private key

User generates a symmetric key: Ku

GenTags *$(Ku, KS, F =< m1\ m2 \ldots mi \ldots mn >, X, c, k1)$:*

1. *For $1 \leq x \leq X$ (i.e.., for each challenge):*
   - 1.1. *Generate a shared secret, $Sx = r.KS = r.k_s.G$ and then transform this EC point to a HMAC key*
   - 1.2. *Choose randomly a value for k_1*
   - 1.3. *For $1 \leq j \leq c$:*
     - 1.3.1. *compute the indices of the data blocks, $ij = \pi k1(j)$*
     - 1.3.2. *$hmacij = \mathcal{H}Sx(mij)$*
   - 1.4. *Generaterate the tag, Txor for challenge x by xoring all the HMACs computed in step 1.3.2*
   - 1.5. *Compute $\mathcal{R} = r.G$*
   - 1.6. *Compute $Tx = \mathcal{AE}_{Ku}(Txor \,||\, x \,||\, k1 \,||\, \mathcal{R} \,||\, ServerID)$*
2. *Outputs(F, T) to the server where $T =< T1\,T2 \ldots TX >$*

GenProof *$(kS, F =< m1\ m2 \ldots mi \ldots mn >, x, c, k1, R)$:*

1. *For challenge x: generate the shared secret, $Sx = kS.\mathcal{R} = k_s.r.G$ and then transform this EC point to a HMAC key*
2. *For $1 \leq j \leq c$:*
   - 2.1. *compute the indices of the data blocks, $ij = \pi k1(j)$*
   - 2.2. *$hmacij = \mathcal{H}Sx(mij)$*
3. *Generaterate the tag, Txor′ for challenge x by xoring all the HMACs computed in step 2.*
4. *Tsign = Sign(Txor′) //Signature will be done on Txor′; no hashing required*
5. *Outputs (Tsign)*

CheckProof *$(Tx, Tsign, KU)$:*

1. *$(Txor \,||\, x \,||\, k1 \,||\, \mathcal{R} \,||\, ServerID) \leftarrow \mathcal{AE}_{Ku}^{-1}(T)$*
2. *Use Txor to verify the signature Tsign*
3. *Implies SUCCESS if step 2 is correct; otherwise it implies FAILURE*

Figure 10 Definition of Our Proposed PDP Solution. It includes all 4 algorithms

5.5.4 Details of the Proposed PDP Solution

Similar to the well-known PDP scheme [76] [85], our proposed PDP scheme is also a probabilistic solution where for each challenge, $Client$ will ask $Server$ to generate a data possession proof for c randomly chosen data blocks from F.

Whereas Figure 11 shows all the interactions between $Client$ and $Server$ involved in constructing the proposed PDP solution, here we will discuss, in detail, the entire flow of our proposed solution which is divided into 2 phases: Setup Phase and Challenge Phase.

5.5.4.1 Setup Phase

The $KeyGen()\ and\ GenTag()$ algorithms are the main building blocks of the setup phase. The setup phase works as follows:

Step 1: In this step, $Server$ and $Client$ execute the $KeyGen()$ algorithm to generate the necessary keys for the proposed solution. $Server$ generates a public-private key pair and makes the public key information (which includes the elliptic curve parameters (a, b, G, n) and the public key point, $K_S = k_S.G$ where k_S is $Server's$ private key) available to its $Clients$. In the case of our system model, we can assume that the TPM inside the $Server$ already has this key pair; so it only needs to be published. The CSP can play the role to distribute public keys for each authoritative storage server (in this case, for all $Servers$) to all $Clients$ or just publish it publicly as a certificate which binds a public key to a particular Server. $Client$ also generates a symmetric key(K_U) which will be used later on to do authenticated encryption of the verification metadata/tags prior to sending them Server.

Step 2: In this step, $Client$ has to choose the values for the following important parameters:

d: the number of blocks the file F will be divided into

c: the number of random blocks involved in each PDP challenge

X: the total number of PDP challenges

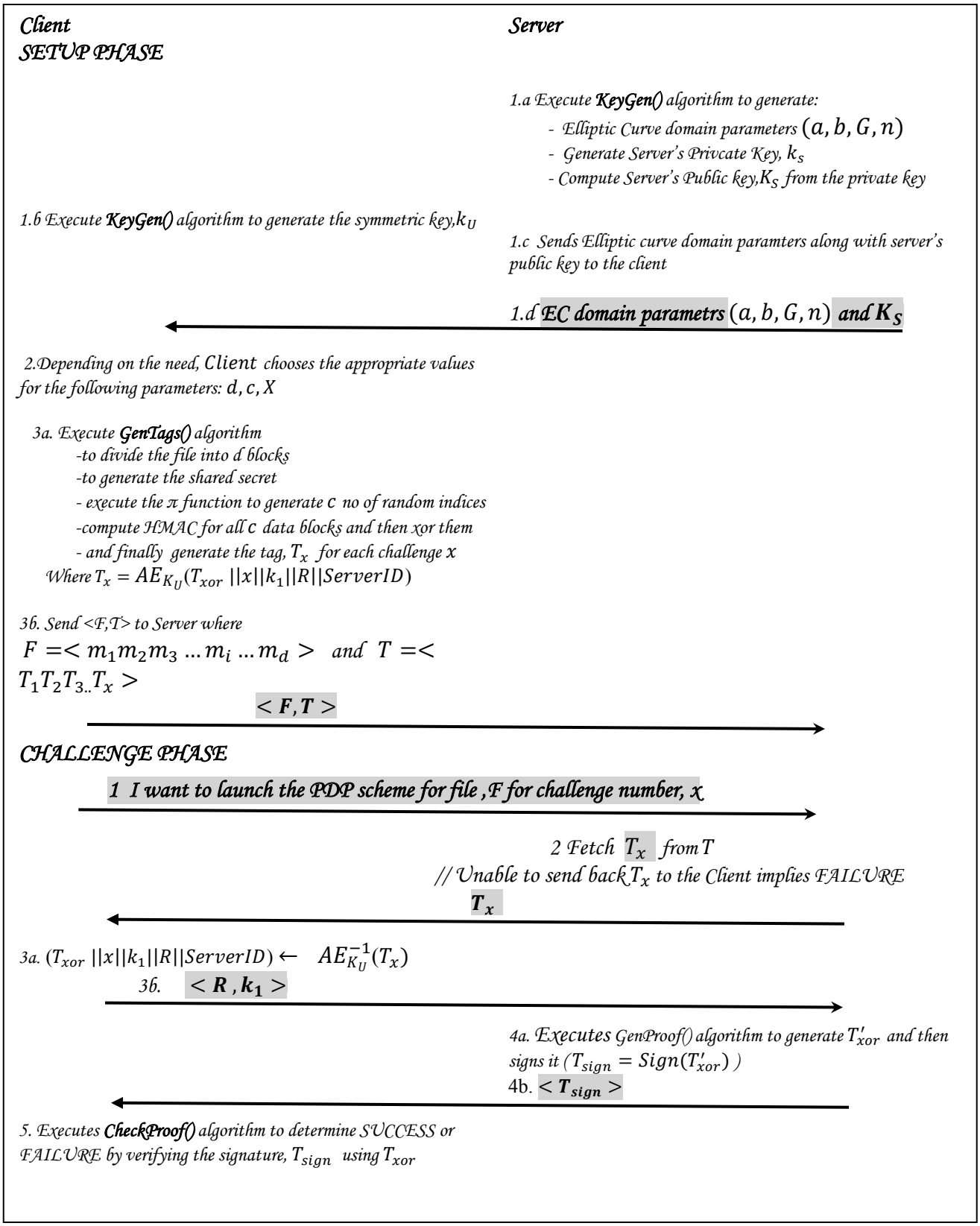

Figure 11 Execution flow of our proposed PDP scheme

Unlike most of the PDP solutions [76] [104] [69] [73] [71] [106], our proposed PDP solution does not put any constraint on either d or each block size. For instance, for the PDP solution proposed in [76], the data block size is dependent on one of the RSA public parameters, n. This is also the case for other PDP schemes [104] based on the RSA algorithm. In contrast to most of the existing PDP solutions [76] [104], computing the verification metadata on a single data block is done by a very simple crypto primitive, HMAC which does not put any constraint on the size of the data block.

Similar to [76], in our proposed PDP scheme c specifies the total number of random blocks involved in a PDP challenge; these c number of data blocks will be used in deciding the server's proof of data possession for a particular PDP challenge. Note that in case of achieving proof of data possession guarantees from Server with 99% confidence for 1% deletion of data, the value of c would be 460; that means the server has to generate the proof using those 460 piece of challenged data blocks. Although [76] includes the detailed procedure of how the detection probability is computed, for completeness we also describe it in Appendix 3.

Similar to [85], in this proposed PDP solution, the client has to decide the total number of PDP challenges (X) it wants to perform for the entire lifespan of the data in that storage server. For instance X can be easily computed as

$$X = 365 * No\_of\_Years * No\_of\_Challenges\_Per\_Day.$$

Note that, X is not dependent on the size of the data and the client can choose any big value as X.

Step 3: This step is executed by $Client$. The main task of this step is to generate tags, T_x where $1 <= x <= X$ for all X PDP challenges using the $GenTags()$ algorithms and then $Client$ sends the entire file along with the tags to $Server$.

Whereas Figure 10 includes the $GenTags()$ algorithm, here we point out the key components of the $\boldsymbol{GenTags}()$ algorithm.

Dividing the file into d equal blocks: In order to store archival data into the cloud with PDP support, the client first divides the entire data/file into d blocks; all of them should

be of the same size (a File $F =< m_1 m_2 m_3 \ldots m_i \ldots m_d >$). It can be considered as a preprocessing step for the $GenTags()$ algorithm.

Generating the Shared Secret: One of the main significant steps of the $GenTags()$ algorithm is to generate a unique shared secret using the $Server's$ Public Key for each PDP challenge. $Client$ generates this shared secret by applying EC point multiplication between a random value, r and the Server's Public Key (K_S). Note that this shared secret is nothing but an elliptic curve point and it will be used later on as the HMAC key during HMAC (i.e., the verification metadata) computation over a single data block. It is imperative that a further computation is required to transform the shared secret from an elliptic curve point into a HMAC key; it can be easily done by just applying a hash operation on x co-ordinate/y co-ordinate/(x||y coordinate) of the shared elliptic curve point.

Tag Generation: In order to generate tags for each PDP challenge, $Client$ first generates the shared secret. $Client$ then chooses a key k_1 as the key for the π function and selects all c number of random challenged blocks. $Client$ then applies HMAC operation, $H_k()$ on each challenged data block and finally applies an x-or operation (T_{xor}) on all computed HMAC tags, $hmac_i \; where \; 1 \leq i \leq c$. $Client$ appends this T_{xor} with $<x, k_1, R(= r.G), ServerID>$ and then applies the symmetric encryption (more precisely authenticated encryption) on it and thus produces the final tag for the PDP challenge x, T_x

where $T_x = AE_{K_U}(T_{xor} \; ||x||k_1||R||ServerID)$

Note that the sole purpose of including R in T_x is to enable the storage server to create the same shared secret during the challenge phase.

$Client$ finally sends the file (F) along with the encrypted tags to $Server$ for storage.

$User \rightarrow Server: < F, T >$ where $T =< T_1 T_2 \ldots T_X >$

At this point, the client can delete the file, F, as well as T. The only thing $Client$ needs to know is its symmetric key. Note that due to the symmetric encryption (i.e., authenticated encryption), $Server$ will not be able to decrypt T_X and thus not be able to retrieve tag related information.

$Client$ is now ready to check whether $Server$ still has its data which will occur in the Challenge Phase.

5.5.4.2 Challenge Phase

The $\boldsymbol{GenProof}()$ and $\boldsymbol{CheckProof}()$ algorithms are the main building blocks of the challenge phase. Whereas Figure 11 shows the overall execution of the challenge phase of the proposed PDP scheme, here we will provide further details about the important steps of this phase.

Step 1: The challenge phase is initiated by $Client$ where the Client sends a request to $Server$ to launch a PDP scheme for file F for challenge number, x. Note that, here the client always sends a fresh PDP challenge to the Server.

Step 2: In this step, $Client$ expects to receive T_X from $Server$. Inability to receive T_X from the storage server would imply Failure for the launched PDP scheme; in other words it would imply that $Server$ does not have some/all part of the file, F.

Step 3: Upon successfully receiving T_X in step 2 from $Server$, $Client$ decrypts T_X using the symmetric key (by applying authenticated decryption, $AE_{K_U}^{-1}$) he/she only knows and extracts ($T_{xor}\ ||x||k_1||R||ServerID$). Note that this step is a part of the $CheckProof()$ Algorithm. By executing this step, $Client$ is able to validate whether this is for the right $Server$ and challenge number for this PDP challenge by checking the serverID and x. In case of a negative finding here, $Client$ can exit the protocol; it would imply FAILURE regarding server's possession of file F.

Step 4: $Client$ sends (R, k_1) to the storage $Server$ and asks if to generate a PDP proof for challenge x.

Step 5: This step is performed solely by $Server$ where it executes the $GenProof()$ algorithm to generate the PDP proof for challenge, x.

The key components of the $GenProof()$ algorithm are as follows: $Server$ first computes the shared secret ($S_x = k_S.R = k_S.r.G$) using its private key k_S. It also executes the π function with the given key k_1 to identify indexes for those c number of data blocks involved in this PDP challenge. $Server$ then uses S_x to compute HMAC for those c number of data blocks. Finally it does xor-ing of all challenged data blocks' hmac (T_{xor}'), then signs it and sends $(T_{sign} = Sign(T_{xor}')$ to the user.

$Server \rightarrow Client: T_{sign}$

Note that no hash computation is required during the signature computation; rather signature will be computed directly on the original T_{xor}' (its size is same as that of a hash). In this case, $Server$ does not also need to send T_{xor}' for signature verification purpose as $Client$ can instead use T_{xor} received in step 3.

Step 6: This step is performed solely by $Client$ where $Client$ executes step 2 of the $CheckProof()$ algorithm. By executing this step, $Client$ is able to make the final "SUCCESS or FAILURE" decision regarding $Server's$ possession of file F by verifying the signature T_{sign} with T_{xor} .

5.5.5 An example of the PDP scheme

For a concrete example of using our proposed PDP, we consider one of the safe curves, SECP256k1- a 256 bit elliptic curve over F_p and a 4 GB file/database F which has n = 1, 000, 000 4KB blocks. As noted in [85], AES is used as the π function to select the random data block indices. As the Authenticated-Encryption scheme, $AE_k()$, we choose AES with GCM mode as its implementation is now publicly available in openssl,oracle jdk 8 and so on. In both instances of AES, the key size is 128 bits. SHA-256 has been used as the hash function, $H()$; it is also used inside the HMAC, $H_k()$ we used in our proposed solution. We store the EC point in compressed form which requires 33 bytes. In addition, for x and $ServerID$ we allocate 3 bytes and 1 byte respectively.

5.6 Security Analysis

5.6.1 Offering 128 bits of security

Our proposed solution can offer 128 bits of security. We generate a 128 bit HMAC key from a point of the SECP256k1 elliptic curve. [As mentioned earlier, we applied a hash operation (SHA-256) on the co-ordinates of that point and take the first 128 bits from there as the HMAC key]. We also use SHA-256 as the hash of the HMAC. So our HMAC offers 128 bits of security since our HMAC key is 128 bits long and SHA-256 used in the HMAC offers 128 bits of security. In addition, as an Authenticated Encryption scheme, we use AES128-GCM which offers 128 bits of security. Furthermore for the Digital signature we also use ECDSA-256 which also offers 128 bits of security.

So our proposed solution offers 128 bits of security. However, depending on the requirement, the security of this solution can be increased/decreased by choosing appropriate crypto primitives such as AES (AES128 /AES192 /AES256), Hash algorithms (SHA1/SHA2/SHA3), Digital Signature algorithms and Elliptic curves.

5.6.2 Offered Security Guarantees

Our proposed solution claims the following security guarantees**:**

Claim 1: For the system and attack model considered for this PDP solution, if Authenticated Encryption (AES-GCM) is secure, Server/s cannot produce a PDP proof without accessing/having the data**.**

Proof strategy: Claim 1 is of the form $A \rightarrow B$ where

A: Authenticated Encryption (AES-GCM) is secure

B: Server/s cannot produce a PDP proof without accessing/having the data

In order to prove theorem/claim Y, we will show that $\neg B \rightarrow \neg A$

Proof: Assume that - "Server/s **can** produce a PDP proof without accessing/having the data"(i.e., $\neg B$).

In our proposed solution, the only feasible way for $Server$ to generate a proof without using/having the data if it can access T_{xor} from the encrypted verification-metadata/tag, T_x for

the PDP challenge x. In that case, Server can directly use T_{xor} to generate a legitimate proof by signing it $\left(i.e., T_{sign} = Sign(T_{xor}) \right)$ without even using any challenged data blocks involved in this PDP challenge. However, it would be **only** possible if $Server$ can break the security of the Authenticated Encryption Scheme and retrieve T_{xor} from T_x.

Apart from that the Server has 2 other alternatives:

a. The Server guesses T_{xor}: It has negligible probability since T_{xor} is 256 bits in length.
b. If a previous PDP challenge is repeated: In that case the Server can resend the previous T_{sign} and thus was able to send a PDP proof to the Client without accessing the data. However, for this proposed PDP solution, the Client is always going to send a fresh PDP challenge to the Server.

In other words, it is **only** possible if "Authenticated Encryption (AES-GCM) is **not** secure" (i.e., $\neg A$). So assuming $\neg B$ implies $\neg A$.

But, we know that Authenticated Encryption (in our case AES-GCM) is secure; more precisely, it is IND-CCA Secure [57] and AES-GCM has been widely used and recommended as an Authenticated-Encryption operation in TLS security [56].

So we can conclude that our Claim 1 holds.

Claim 2: For the system and attack model considered for this PDP solution: if the elliptic curve discrete logarithm problem is hard to break, an illegitimate server cannot generate a false data-binding proof.

Proof strategy: Claim 2 is of the form $A \rightarrow B$ where

A: Elliptic curve discrete logarithm problem is hard to break

B: An illegitimate Server cannot generate a false data-binding proof.

In order to prove Theorem/claim 1, we will show that $\neg B \rightarrow \neg A$

Proof: Assume that "An illegitimate Server **can** generate a false data-binding proof"(i.e., $\neg B$).

The usage of HMAC during the tag generation process binds the data to a specific storage server. In our system and attack model, the HMAC key is assumed to be a shared secret only between client and the legitimate (authoritative) Server. In our solution, an illegitimate server (i.e., any server other than the one with whom the client generates the shared secret) can generate the data binding proof if it can compute the HMAC correctly on each randomly chosen challenge data block. In order to achieve that, the illegitimate server will require the correct HMAC key. For an illegitimate server it is only possible if it can extract $r\ or\ k_s\ \ from\ R\ (where\ R = r.G)$ and $K_S\ (whereK_S = k_s.G)$ resepectively which implies breaking/solving the Elliptic curve discrete logarithm problem. In that case, the illegitimate server can then create the shared secret $S_x = K_S.r = k_s.r.G$ or $S_x = R.k_s = r.G.k_s$ and then transform it into the HMAC key by applying a Hash operation as indicated in the previous section. In other words, the illegitimate server can achieve this if "the Elliptic curve discrete logarithm problem is **not** hard" (i.e.,$\neg A$). So assuming $\neg B$ implies $\neg A$.

But the Elliptic Curve Discrete Logarithm problem is widely believed to be a hard problem [107]. In our case, we use the SECP256k1 elliptic curve and breaking the elliptic curve discrete logarithmic problem for this curve is also hard. This curve has been used in many secure applications such as bitcoin [108].

So we can conclude that our Claim 2 holds.

Claim 3: For the system and attack model considered for this PDP solution, if Authenticated Encryption (AES-GCM) is secure, the data possession proof generated by the Server is fresh.

Proof: The proof of this claim is similar to Claim 1; so we do not repeat the formal proof here. Rather here we provide a brief summary of this claim.

Note that data possession proof is dependent on the following: (a) all challenged data blocks involved in a particular challenge, and (b) the HMAC key generated from the shared secret. Server would be able to generate data possession proof in advance only if it knows (a)

and (b). The only way for the server to know (a) and (b) is if it can break authenticated encryption and decrypt the tag, T_X. However, Authenticated Encryption is assumed to be secure and thus the proof generated by Server is fresh and cannot be computed beforehand.

5.7 Performance Analysis

Here we first analyze the performance of our proposed solution; we then compare our performance numbers with some significant related work. We also compare the performance numbers (e.g., computation, communication and storage) of our proposed solution with some existing PDP solutions and solutions of this kind.

5.7.1 Performance Numbers

A. Computation Overhead

In contrast to most of the existing PDP solutions [76] [104] [68] [70], our proposed solution relies on significantly less expensive computations. For each PDP challenge, the most expensive operation of this proposed PDP solution is the scalar point multiplication on the elliptic curve; both Client and Server have to do one scalar point multiplication to generate the shared secret between Client and Server. Whereas Client generates this shared secret in the setup phase (i.e., during data storage), Server has to generate this shared secret in the challenge phase. The main advantage of our solution is that the verification metadata/tag has been computed using HMAC, which is considered to be a very fast operation.

Setup Phase: During the setup phase, for each PDP challenge:

Client has to do these computations: $1\,mult + 1\,hash + c\,(HMAC + PRP) + (c-1)\,XOR + 1\,AE_k()$ to generate the (encrypted) verification metadata.[here *mult* refers to the scalar EC point multiplication which is needed to generate the shared secret, and 1 hash operation is required to transform this shared secret into a HMAC key]. So if Client chooses to have X PDP challenges for the entire lifespan of the outsourced data, the total number of computations done by the Client would be X times the above mentioned computations; more precisely it would be:

$X * (1\,mult + 1\,hash + c\,(HMAC + PRP) + (c-1)\,XOR + 1\,AE_k())$

In this phase, **Server** does not need to do any computation but storing the file along with verification metadata for X PDP challenges.

Challenge Phase: Unlike the setup phase, Challenge Phase is time critical for a PDP scheme because in this phase Client/verifier interacts with Server/prover to get a verifiable PDP guarantee of its outsourced data from the Server for a particular PDP challenge. The performance numbers of the challenge phase play an important role in choosing the best PDP solution for a particular use case for the deployment of a PDP solution.

During the challenge phase, for each PDP challenge-

- ✓ Client has to do the following computations:

$1\ AE_k^{-1}() + 1\ SigVerify$

- ✓ Server has to do the following computations:

$1\ mult + 1\ hash + c(HMAC + PRP) + (c-1)\ XOR + 1\ Signature$

It is clear that Server has to do a Signature and Client has to verify that signature in the challenge phase.

B. Communication Overhead

The communication overhead involved in our PDP scheme is very reasonable and it mainly depends on the size of the signature and the size of an elliptic curve point. For communication overhead, we consider only the Challenge phase.

According to our proposed solution described in the previous section, for each PDP challenge, the communication (in bits) for Client and Server includes:

$|T_x| + |k1| + |EC\ Point| + |Signature|$ where

$|T_x| = (|HMAC| + |k1| + |EC\ Point| + \lceil \log X / \log 2 \rceil + |ServerID|)$.

In our case, $|T_x| = (32 + 16 + 33 + 3 + 1)$ Byte $= 84$ bytes. We save the EC point in compressed form which requires 33 bytes. In addition, for x and ServerID we allocate 3 bytes and 1 byte respectively. Note that the communication overhead can be further reduced by choosing a more efficient (in terms of size) signature scheme such as Short Signature scheme proposed by Boneh et al. [109] (instead of ECDSA) where the signature can be as low

as 20 bytes. However, for short signature, we need to also use an additional crypto primitive called Pairing-based cryptography (see section 2.4.5).

C. Storage Overhead

Similar to [85], in our proposed PDP solution the storage overhead does not depend on the total no of data blocks or the size of the data blocks; rather it depends on the total number of data possession challenges (i.e., X) which is a constant. Unlike other existing PDP schemes, the underlying crypto primitives do not put any upper limit on either total number of data blocks of the outsourced data or each data block size; they can be of any size. So in contrast to many existing PDP solutions [76] [104] [69] [73], in our solution, the storage overhead does not increase when the total number of data blocks increases (in other words when file size increases).

In order to quantify the storage overhead, we need to consider the Setup phase. Storage overhead can be defined as any additional data stored on the Server besides the outsourced data. According to our proposed solution described in the previous section, the storage overhead of our solution is:

$X * (|T_x|)$ where $|T_x| = 84$ bytes (as above).

An example: Assume that Client wants to periodically (every M minutes) obtain a proof of possession and wants the tokens to last Y years. The number of verification tokens required by our scheme is thus: (Y × 365× 24× 60)/M. For example, in case of our proposed solution, each tag size is 84 bytes and thus with only about 47 Mbytes, outsourced data with any size can be verified each hour for 64 years, or, equivalently, every 15 minutes for 16 years.

Remark 2: In our proposed solution, storage overhead is a constant and is not dependent on the size of the outsourced data. In addition, in the case of high volume of outsourced data, the storage requirement for verification tag can be significantly lower. A Client can perform in-house storage of the verification tags (with 500 Mbytes, it could ensure data possession verification in every 5 minutes for 56 years). Note that doing in-house storage of verification tags on the Client side, significantly enhances the security and performance of our proposed solution.

5.7.2 Performance Numbers Comparison

In Table 3, we compare our proposed PDP solution with some existing related works with respect to their computation, communication and storage overhead. Some of the performance numbers of some existing PDP solutions have been taken from [106]. Note that all existing solutions of Table 3 except our proposed solution can offer up to 80 bits of security (look at their underlying crypto and group element size), whereas our solution can offer up to 128 bits of security as we are using 256 bit EC.

Table 3 Performance Comparison with some existing PDP schemes

In each scheme, a challenge set $C \subset [0, d-1]$ contains l block indices and can be compactly represented with 280 bits due to results of [65] [75]. In the table, "exp", "mul" and "add" represent exponentiation, multiplication and addition in the corresponding groups/fields; "samp" represents the sample method given by Goldreich [75]; notation $|F|$ denotes the outsourced data/file size in bits. Here λ denotes the bit-length of group element size, $s(=|m|)$ denotes the block size, and l denotes the number of data blocks accessed during one verification. Note: In Ateniese et al. [39] [73]'s PDP scheme (based on RSA), exponentiation with a large integer exponent of size $s\lambda$ is required. We represent such exponentiation as a number of s normal group exponentiation exp, where the exponent is λ bits long. For our scheme, assume that, X is the total number of PDP challenges Client wishes to have for the entire lifespan of the outsourced data and T_x is the verification-metadata for any challenge x where $1 \leq x \leq X$ and $|T_x| = (|HMAC| + |k1| + |EC\ Point| + \lceil \log X / \log 2 \rceil + |ServerID|)$=(32+16+33+3+1) Byte=84 bytes

Scheme	Group Element Size(bits)	Communication (bits)	Storage Overhead	Computation (Prover)	Computation (Verifier)
Ateniese et al. [39] [73]	$\lambda = 1024$	$2\lambda + 520$	$\|F\|/s$	$(l+s)exp + 2l\ mul + l\ add + 1\ hash + 1samp$	$l(exp + mul) + 1\ hash + 1\ samp$
S.W. [66]	$\lambda = 80$	$(s+1)\lambda + 360$	$\|F\|/s$	$sl(add + mul) + 1\ samp$	$(l+s)(add + mul) + l\ PRF + 1\ samp$
EPOR(E.C) [74]	$\lambda = 160$	$3\lambda + 440$	$\|F\|/s$	$(s-1)exp + (sl + s + l)(add + mul) + 1\ samp$	$2\ exp + l\ (add + mul) + l\ PRF + 1\ samp$
EPOR(Z_q^*) [74]	$\lambda = 1024$	$3\lambda + 440$	$\|F\|/s$	$(s-1)exp + (sl + s + l)(add + mul) + 1\ samp$	$2\ exp + l\ (add + mul) + l\ PRF + 1\ samp$
EPOS [68]	$\lambda = 1024$	$2\lambda + 416$	$\|F\|/s$	$(s+1)exp + (sl + s + l)(add + mul) + 1\ samp$	$2\ exp + l\ (add + mul) + l\ PRF + 1\ samp$
Our Work	$\lambda = 256$	$\|T_x\| + \|k1\| + \|EC\ Point\| + \|Signature\|$	$X\ (\|T_x\|)$	1 mult $+1\ hash + l\ (HMAC + PRP) + (l-1)\ XOR + 1\ Signature$	$1\ AE_k^{-1}() + 1\ SigVerify$
Note: For comparison purpose, we can always replace our $l\ PRP\ functions\ with\ 1\ samp$ [68]					

It is visible from Table 3 that our proposed solution is better than the other existing PDP solutions in terms of computation (in terms of both server side and client side computation) and communication. As we mentioned earlier, the storage overhead in our proposed solution is independent of the data size; rather it depends on X-the total number of PDP challenges the Client wishes to have for the entire lifespan of its data. Whereas for existing PDP schemes,

storage overhead will increase as the size of the data increases (and thus the number of data blocks), it will remain the same in our proposed solution.

In order to further quantify the performance comparisons, in Table 4 we provide a concrete comparison with respect to the setting mentioned in the following example. The following example has been chosen to have a common platform for comparison with existing solutions.

Example: the file size is 1GB, block size is m = 100, and storage overhead due to authentication tags is about 10MB for all schemes except ours. We assume that all scheme choose the same procedure for selecting the challenged data block set involved in each PDP challenge. System parameter l represents the size of the challenged data block set. All computation times are represented by the corresponding dominant factor. *exp* and *mul* denote group exponentiation and group multiplication respectively in the corresponding group. Note that one 1024 bit modular exponentiation and one 160 bit elliptic curve exponentiation takes roughly 5 millisecond in a standard PC [106] . For simplicity, assume that one 160 bit elliptic curve point multiplication also takes the same time.

Note that our proposed solution involves 256 bit elliptic curve multiplication (and offers up to 128 bits of security). For the sake of comparison, we can also make the assumption of using a 160 bit elliptic curve multiplication if we consider offering 80 bits of security similar to all existing solutions mentioned in Table 4.

As you can see from Table 4, our proposed solution is significantly better than the existing PDP solutions in terms of communication and computation overhead. More precisely, in case of computation (in case of both client side and server side computation), our proposed solution outperforms all existing solutions with a high magnitude and makes it very suitable for resource constrained devices. We do not perform a direct comparison for the storage overhead as in case of our proposed solution, it also depends on the total number of PDP challenges, X

Table 4 Performance comparisons with an concrete example

Comparison with an example among the PDP scheme by Ateniese et al. [39] [73], the POR scheme by Shacham andWaters [66], the POR scheme named EPOR proposed by Xu and Chang [74], the PDP scheme,EPOS proposed in [68] . After erasure encoding, the file size is 1GB, block size is s = 100. We assume that all schemes choose the same procedure for selecting the challenged data block set involved in each PDP challenge. System parameter l represents the size of challenged data block set. **All computation times are represented by the corresponding dominant factor**. "*exp*" and "*mul*" denote the group exponentiation and group multiplication respectively in the corresponding group. Note: (1) In the Ateniese et al. PDP scheme, exponentiation with a large integer exponent of size $s\lambda$ is required. We represent this exponentiation as a number of s normal group exponentiations *exp*, where the exponent is λ bits long. (2) One 1024 bit modular exponentiation or one 160 bit elliptic curve exponentiation (also point multiplication) takes roughly 5 milliseconds in a standard PC of s normal group exponentiations exp, where the exponent is λ bits long. Similar for the RSA based scheme.(3) For our scheme, assume that X is the total number of PDP challenges Client wishes to have for the entire lifespan of the outsourced data and T_x is the verification-metadata for any challenge x where $1 \leq x \leq X$ and $|T_x| = (|HMAC| + |k1| + |EC\ Point| + \lceil \log X / \log 2 \rceil + |ServerID|)$=(32+16+33+3+1) Byte=84 bytes.

Scheme	Group Element Size (bits)	Communication (bits)	Stor-age Over-head	Computation (prover)	Computation (Verifier)
Ateniese et al. [39] [73]	$\lambda = 1024$	$2\lambda + 520 = 2568$	10MB	$(100 + l)$ exp $over\ Z_N^*$	l exp $over\ Z_N^*$
S.W. [66]	$\lambda = 80$	$(s+1)\lambda + 360 = 8440$	10MB	$100l\ mul\ over\ Z_P^*$	$(100 + l)\ mul\ over\ Z_P^*$
EPOR(E.C) [74]	$\lambda = 160$	$3\lambda + 440 = 920$	10 MB	100 exp $over\ Elliptic\ Curve$	3 exp $over\ Elliptic\ Curve$
EPOR(Z_q^*) [74]	$\lambda = 1024$	$3\lambda + 440 = 920$	10 MB	100 exp $over\ Z_q^*$	3 exp $over\ Z_q^*$
EPOS [68]	$\lambda = 1024$	$2\lambda + 416 = 2464$	10 MB	102 exp $over\ Z_N^*$	3 exp $over\ Z_N^*$
Our Work	$\lambda = 256$	$\|T_x\| + \|k1\| + \|EC\ Point\| + \|Signature\| = 84 * 8 + 128 + 33 * 8 + 512 = 1576$	$X\ (\|T_x\|) = X * (84\ B)$	1 mult $over\ \ Elliptic\ Curve$	1 $SigVerify\ using\ ECDSA$

5.8 Construction of DLAS using our proposed PDP

A DLAS solution for the system and attack model given in section 5.2, can easily be constructed using our proposed PDP solution. Note that we can follow the same procedure used in the HDLAS solution for constructing the DLAS solution. We call this solution as $\boldsymbol{DLAS_{PDP}}$. In short, here a user has to launch the proposed PDP scheme with all authoritative storage servers of user's interest and based on the outcomes of launched PDP scheme/s, Client finds out whether "there are any violations in terms of user's geo-location preference for a particular data".

Unlike the HDLAS solution, this DLAS solution does not require any additional support (e.g., ensuring correct execution of PDP processes) from the TPM; in our solution a TPM has only to do its regular operations which are built-in inside the chip. Furthermore, unlike HDLAS, there might be no need for the use of the GPS receiver as long as all storage servers' public keys along with their geo-graphical bindings are publicly available and regularly updated by the CSP. The main advantage of this DLAS solution over the HDLAS is that it can work in a more adversarial system and attack model (than that of HDLAS) where a server can ask for help to other servers for generating data possession proof.

Remark 3: In this proposed DLAS solution, we are not using any GPS receiver to verify the location of the outsourced data. Rather, here we achieved it by binding the data possession proof with the authoritative storage server. Our assumption is that assigning an identity (e.g., by disclosing the public-private key-pair of the attached TPM) to an authoritative storage server would be a big event for any CSPs and once it is done, a CSP will not just change it for the sake of telling a lie about the location of a piece of user data. This event could be considered is as big as the event of a "root key ceremony" for a Certificate Authority.

Remark 4: In this proposed solution, a cloud user receives the data possession proof from the authoritative storage server. In other words, it only proves that the authoritative storage server has access to all challenged blocks of data at that point in time. Logically, it is possible that the authoritative storage server does not keep the data all the time, rather it gets all the challenged blocks of data from another authoritative storage server once it received a DLAS/PDP query from a cloud user. A good countermeasure against this scenario would be to make a DLAS/ PDP query within a short interval (e.g., every 5 to 10 minutes). In this case, due to the distributed architecture of the cloud storage environment (where bandwidth is considered to be more expensive than storage), it would not be feasible for an authoritative storage server to fetch challenged data from another authoritative storage server in every 5-10 minutes.

5.9 Conclusion

In this chapter, we have proposed the first PDP solution where the data possession proof is cryptographically bound to a server. It is specifically designed for the cloud storage environments where the client will receive the verifiable data possession proof from a specific storage server located in a particular data center rather than from the cloud as a whole. This unique feature of binding the data possession proof to a particular server and thus to a particular location in terms of a data center opens new avenues of cloud storage applications. For instance, now an individual or an organization will be able to verify whether their data is really kept in the data center located in Region X (say North America) where it is supposed to be stored. As a result, organizations that dealt with sensitive data can now adopt a cloud storage solution which was not possible in the past due to the absence of a data location assurance guarantee.

Our PDP solution is proposed for static data and it is provably secure. This PDP solution is not only suitable for the cloud storage environments but is also very much suitable for the typical PDP scheme's usage scenario of having a single un-trusted storage server. It has the potential to outperform many existing PDP solutions in terms of computation, communication and storage overhead. Furthermore it can be even used for resource-constrained environments such as mobile cloud computing. Here, we have also discussed how a DLAS solution can be constructed based on our proposed PDP solution.

It is important to note that both the PDP and $DLAS_{PDP}$ solutions proposed in this chapter, offer a data possession guarantee from a particular authoritative storage server which is valid at a particular point in time. It is theoretically possible that the actual data is stored in another datacenter and that particular authoritative storage server just fetches all pieces of challenged data when it receives a PDP / DLAS challenge request from a cloud user. However, as we mentioned earlier, it becomes a very improbable case when a cloud user makes these PDP / DLAS challenges within a short interval (e.g., every 5 to 10 minutes). Note that for the existing CSPs, bandwidth is considered to be more expensive than storage and thus any existing CSP would try to avoid any frequent (which is also unnecessary and avoidable) data transfer attempt. By considering the above, we can say that our proposed PDP/ DLAS solutions explicitly guarantee the following: "*The data is not where it is not supposed to be for a period longer than between 2 PDP /DLAS requests*."

6 Proposed Solution 4: Further Enhancement of DLAS Solutions

This chapter is comprised of 3 contributions. First, we discuss adding the *public verifiability* feature into our proposed DLAS solution. Second, we show how the *dynamic operation on data* feature can be incorporated into our proposed DLAS solution. Finally, we present a DLAS solution which has the potential to offer the optimal security guarantees for our most adversarial system and attack model. The main intention of this solution is to capture the security properties, which an ideal DLAS solution should have. This should facilitate any future initiative of proposing or enhancing a DLAS solution for any system and attack model.

6.1 Enabling Public Verifiability in our proposed solution

In all of the DLAS solutions, we proposed so far, we consider that the data owner and the cloud data user are the same entity. As a result, in all of our proposed solutions only the data owner (i.e., the Client in our previous PDP and DLAS solution) would be able to get the Data Location Assurance (DLA) service (also true for our proposed Proof of Data Possession scheme). However, in cloud storage environments, there are some scenarios where the data owner and the cloud users can be different entities. An example is the health-related data storage scenario where many people have to share the same data. For those scenarios, it is important for the cloud data user (aside from the data owner) to have the capability of executing the DLA service (also the PDP service). This is known as public verifiability where, along with the data owner, a cloud data user with a proper credential (supplied by the data owner) is also able to verify the data location guarantee received from a server.

Note that public verifiability is not on the requirement list we set for our proposed solutions (DLAS solutions and PDP solution) presented in the previous chapters. We plan to incorporate this public verifiability service into our proposed PDP as well as $DLAS_{PDP}$ solutions. This capability would make our proposed solutions suitable for a broad range of cloud storage

applications such as the health sector where outsourced data would be accessed by different cloud users (e.g., doctors, nurses, and other health professionals).

In the following subsections, we provide a rough sketch of how the public verifiability feature can be added into our proposed PDP solution (and thus our $DLAS_{PDP}$ solution). More precisely, first we present the general idea of incorporating public verifiability into our proposed PDP solution and then, based on the type and size of cloud users, we discuss two possible constructions for adopting public verifiability into our proposed PDP solution (and thus our $DLAS_{PDP}$ solution).

6.1.1 Enabling Public Verifiability- General Idea

In order to facilitate public verifiability in our proposed PDP solution (and thus in our $DLAS_{PDP}$ solution) presented in the previous chapter (see chapter 5), we need to ensure that the cloud user (not the data owner) is able to execute the $CheckProof$ (T_x,T_{sign}, K_U) algorithm of our proposed PDP solution. Note that one of the major steps of this algorithm is to decrypt T_x $where$ $T_x = AE_{K_U}(T_{xor} ||x||k_1||R||ServerID)$ and for this, one has to know the symmetric key (for authenticated encryption), K_U. Currently, in our proposed PDP solution, data owner (i.e., the $Client$ in our PDP and $DLAS_{PDP}$ solution) only knows K_U and thus is able to execute the $CheckProof()$ algorithm and verify the proof of data possession received from Server for its outsourced data.

A cloud user, other than the data owner, will be able to execute the $CheckProof$ () algorithm only if he/she has the knowledge of K_U. The basic idea of incorporating public verifiability into our proposed PDP solution (and thus our $DLAS_{PDP}$ solution) is as follows:

- ✓ We need to come up with a mechanism to ensure that cloud users (those who want and are authorized to perform PDP verification) have secure access to K_U.

In the following subsections, we discuss two possible constructions for incorporating public verifiability into our proposed PDP solution, which differ in terms of the mechanism chosen for providing secure access to K_U for cloud users.

Remark 1: Note that, in case of our proposed "public verifiability" support, the term "public" does not literally include everyone; rather it refers to a set of registered cloud users who might also want to perform data possession verification. In our case, the data owner (i.e.; the Client) needs to know those cloud users so that he/she can facilitate those cloud users, later on, to perform the data possession verification. [We believe, for cloud storage environments, it is a valid assumption that cloud users need to be registered or are known to the data owner (the Client); we just do not expect a completely unknown entity to perform a "data possession" verification.] So more precisely, our "public verifiability" support can be named as "public verifiability for registered cloud users". However, for simplicity, we still continue to use the term "public verifiability" while we discuss this solution.

6.1.2 Enabling Public Verifiability Using PKI

One very straightforward solution would be to use the traditional Public Key Infrastructure (PKI) to ensure secure access to K_U for all cloud users. It can be constructed as follows:

Assume that aside from the data owner, there are n cloud users and all of them have a <public-private> key-pair. The data owner and all cloud users also know and trust each other's public certificates.

In this case, the data owner can include all cloud users' public keys in its recipient list and encrypt K_U using each public key from the recipient list. Now the data owner can somehow make all ciphertexts of K_U available to all recipients. At that point, a cloud user will be able to extract K_U by successfully decrypting one ciphertext copy of K_U assuming that s/he is on the recipient list of the data owner and thus will be able to verify the data possession proof offered by the Server.

The following changes (mainly addition) need to be made to our proposed PDP solution to adopt this public verifiability feature:

Setup Phase: Modification in the GenTags() algorithm

- ✓ In step 2 of the $GenTags()$ algorithm of the proposed PDP solution (see Figure 10 and Figure 11), along with $< F, T >$ where $F =< m_1 m_2 m_3 \dots m_i \dots m_d >$ *and* $T =< T_1 T_2 T_3 .. T_x >$, the data owner (i.e., Client) would also send n ciphertext copies of K_U *to* $Server$ (assuming there are n cloud users). So the data owner or Client would send the following to the server:

  $\text{Client} \rightarrow \text{Server}: < \text{F}, \text{T}, KEY_{ENC} >$

  where $KEY_{ENC} =< key_{enc1} || key_{enc2} || \dots || \dots || key_{encn} >$

Challenge Phase: Modification in the $CheckProof()$ algorithm

- ✓ This step should be added as the first step of the Challenge phase. As the first response to a cloud user's interest in pursuing a data possession challenge, Server sends KEY_{ENC} along with the encrypted tag,T_x.

  $\text{Server} \rightarrow \text{Client}: < KEY_{ENC}, T_x >$

- ✓ In step 1 of the $CheckProof()$ algorithm (see Figure 10), a cloud user will try to extract K_U by decrypting each $keyi_{enc}$ $where$ $1 \leq i \leq n$ of KEY_{ENC} using its private key. Note that he should be able to decrypt one of those n ciphertexts successfully if s/he is included in the data owners' recipient list. The cloud user then uses K_U to decrypt T_x and follows the remaining steps of the $CheckProof()$ algorithm as-is to verify the data possession proof received from the Server.

6.1.2.1 Discussion

Security: This PKI-based public verifiability enhancement implies that now the overall security of our proposed PDP (and thus $DLAS_{PDP}$ solution) relies on how we protect the K_U to provide secure access to all cloud users. In other words, it now mainly relies on the security strength of the PKI-based encryption scheme we use to protect K_U. If, similar to our initial proposed PDP solution, we want to achieve the same 128 bits of security, we need to use a PKI-based encryption scheme that can offer 128 bits of security. For this, similar to our original PDP solution, we could still use the same elliptic curve, SECP256k1 or choose any safe

EC curve with a 256 bit prime modulus for EC curve based Public Key Encryption (more precisely ECIES). Alternatively, it is also possible to choose the RSA Encryption scheme and in this case, we need to use RSA 3072 to ensure 128 bits of security.

Performance: The public verifiability enhancement comes with the additional work of encrypting and decrypting K_U for ensuring secure access to it by all cloud users. In terms of computation overhead, the data owner has to do n encryptions and the cloud user has to do n decryptions in the worst case to retrieve K_U. Most likely it would be a onetime computation job for both the data owner and the cloud user.

In terms of communication overhead, this enhancement has the following overhead:

- The data owner has to send n ciphertext copies of K_U to the Server once during the data storage attempt i.e., in the Setup phase.
- The server sends n ciphertext copies of K_U to the cloud user in response to the first PDP challenge received from that cloud user.

In regards to the storage overhead, now the Server has to additionally store n ciphertext copies of K_U. With some further additional storage overhead (for making a link to identify which ciphertext is for which cloud user), communication and computation overhead can be further optimized. For instance, in regards to the communication overhead, now the server can send only the ciphertext copy of K_U intended for that particular cloud user, and as far as computation overhead is concerned, the client now only needs to perform one decryption attempt instead of n.

Note that most of the overhead involved in this enhancement is only dependent on the size of cloud users, n; it does not depend on either the size of the outsourced data or the total number of PDP challenges. So this PKI-based enhancement would work well as long the size of n is reasonable.

6.1.3 Enabling Public Verifiability Using CP-ABE

The PKI-based public verifiability construction works well when the size of *n* is reasonable. However, when the size of *n* becomes very large, the efficiency of that construction might become questionable.

There exist some cloud storage applications (e.g., in the health sector) which may involve a large number of cloud users with different roles (doctors, nurses, administrative personnel and so on). For those cloud storage scenarios, we can use CP-ABE to enable public verifiability in our proposed PDP and $DLAS_{PDP}$ solution. This construction would be similar to that of the PKI based one with the exception that here we are using CP-ABE instead of a Public Key based encryption scheme to encrypt/decrypt K_U.

Since we already discussed the CP-ABE in our DLAS solution (see chapter 3), we will not repeating it here.

Note that this CP-ABE based enhancement is very suitable for a cloud storage scenario for the health sector. For the CP-ABE scheme, we need to have a trusted entity/central authority who is responsible for generating and distributing keys for all users. A cloud storage application for the health sector is a perfect fit for deploying CP-ABE as we can also assume a central authority here. Whereas the cloud user and the data owner would be involved in doing CP-ABE encryption and decryption, Server would be totally unaware of any CP-ABE activity. From the Server's perspective, it just stores CP-ABE encrypted data and sends it back to the cloud user once it is needed.

Most importantly, CP-ABE can be configured similar to RBAC (Role-Based Access Control); in that case a (cloud) user will be able to access a document/resource based on their role. It offers further granularity of ensuring access control to the organization's outsourced data.

6.1.3.1 Discussion

Security: Unlike the PKI-based public verifiability scheme, this CP-ABE based public verifiability scheme is based on the bilinear pairing. Note that we already talked about CP-ABE in one of our previous DLAS solutions (i.e., in chapter 3). According to [41], it is possible to

achieve up to Chosen Ciphertext Attack (CCA) security for the CP-ABE scheme. The details security analysis of the CP-ABE scheme is also available in [41].

Performance: Unlike the PKI-based public verifiability scheme, in this case, the data owner has to encrypt K_U only once. Moreover it allows the fact that a user may not exist while encryption of K_U was taken place, but later on a new user would be able to still decrypt if his/her role is included in the access tree structure set for the ciphertext (see section 2.4.2 for further details on access structure which also includes an access tree structure example).

Note that the efficiencies of the key generation and encryption algorithms are both straightforward. According to [41], the encryption algorithm requires two exponentiations for each leaf in the ciphertext's access tree and the ciphertext includes two group elements for each tree leaf. In addition, the key generation algorithm requires two exponentiations for every attribute given to the user, and the private key consists of two group elements for every attribute. Furthermore, the decryption algorithm needs two pairings for every leaf of the access tree that is matched by a private key attribute and one exponentiation for each node along a path from such a leaf to the root.

It is important to note that in our case, only the decryption task is on the critical path; the timing for the key generation and the encryption are not of utmost importance here. Furthermore in our case, access tree structure for the CP-ABE encryption would be very simple (e.g., if we want to give access to every user of an organization, we might only need to use one attribute and one leaf node); in that case, it would further enhance the overall performance.

CP-ABE has a public implementation (cpabe tool, [42]) and a detailed performance analysis of cpabe tool has been included in [41]; it seems that it takes only a few milliseconds (less than 20 milliseconds for each operation executed on a Pentium 4 processor with a more complicated access tree structure than ours) for key generation, encryption and decryption operation.

6.2 Supporting Dynamic Operation on Outsourced Data

The main focus of this thesis is to come up with DLAS solutions for static/archival data, and thus, all the solutions we proposed so far work only for static/archival data. However, having "dynamic operation on data" is always a plus and can add more value to a DLAS solution. A DLAS solution with "dynamic operation on data", allows a user to perform an update operation (e.g., modify, delete, append and so on) on the outsourced data and yet continue to get the data location assurance service. In this section, for demonstration purposes, we show how the "dynamic operation on data" feature can be added in our PDP and thus in our $DLAS_{PDP}$ solution**.** More precisely, we need to add support of "dynamic operation on data" capability into our proposed PDP solution; this capability would be automatically enabled into our $DLAS_{PDP}$ solution (as $DLAS_{PDP}$ solution is based on our PDP solution).

6.2.1 General Idea

In general, a very simple solution for "dynamic operations on data" would be for Client (i.e., the data owner) to download from Server the entire outsourced data, F, for each dynamic operation and to re-execute the "Setup Phase". This would clearly be secure but highly inefficient.

In order to efficiently handle dynamic operations on outsourced data, we need to make some changes in our proposed PDP solution. The mechanism we apply to make those changes in our PDP solution for enabling "dynamic operation on data" capability is similar to the one presented in [85].

Here we point out the key factors involved in enabling "dynamic operation on data" capability into our proposed PDP solution:

- **Recall from our proposed PDP solution**: Client stores $< F, T >$ into the Server where $F(=< m_1 m_2 m_3 \dots m_i \dots m_d >)$ is the data and $T(=< T_1 T_2 T_3 .. T_x >)$ consists of verification tag for each PDP challenge $1 \leq x \leq X$ where X is the total number of PDP challenge for the entire lifespan of data, F. In addition, the tag for challenge number-x, T_x is in the following form: $T_x = AE_{K_U}(T_{xor} \,||x||k_1||R||ServerID)$. Since T_{xor} would be the main focal point for enabling dynamic operations in our proposed PDP solution, here we explain

again how T_{xor} was originally computed in our proposed PDP solution- for each single PDP challenge (x), Client first applies HMAC operation, one at a time, on c number of randomly chosen data blocks involved in that PDP challenge and then perform "x-or" operation on them to generate T_{xor}. For illustration purpose, T_{xor} computation of our PDP solution presented in the previous chapter, can be presented with the following pseudocode:

Pseudocode for $\boldsymbol{T_{xor}}$ computation in our PDP solution

Assumption: For simplicity assume that c number of randomly chosen data block involved in a single PDP challenge (and thus in corresponding verification tag) has been kept in a temporary array, temp.

1. For i=1 to c
   a. $hmac_i = H_k(temp[i])$
2. $T_{xor} = hmac_1 \oplus hmac_2 \oplus hmac_3 \oplus \ldots \oplus hmac_{c-1} \oplus hmac_c$

Figure 12 Pseudocode for T_{xor} computation of our previously proposed PDP solution

- Assume that Client wants to do a dynamic operation (e.g., update, delete) on the i-th data block. In that case, Client cannot disclose to Server which verification tag includes the i-th block. The reason is simple: if Server knows which verification tags include the i-th block, it can simply delete i-th data block if and when it knows that no remaining verification tags include it. Because of this reason, Client must modify all verification tags for any dynamic operation on data.
- For enabling dynamic operation in our PDP solution:
  - We have to update only the T_{xor} from T_x; all other components of T_x will remain unchanged. Performing x-or operation between 2 identical elements, cancels it out, and we manipulate this property to do any dynamic operation on the outsourced data.
  - We have to make a trivial change in the verification tag structure, more precisely in step 2.a of the pseudocode for T_{xor} computation described in Figure 12. " $hmac_i = H_k(temp[i])$ " should be changed to $hmac_i = H_k(i||temp[i])$.

The reason is that two blocks with identical content would produce the same hash and would cancel each other out due to the exclusive or operation and including a unique index in each block hash addresses this issue. Note that for supporting "dynamic operation", we must ensure that it does not happen in case of the presence of two identical data blocks in a single PDP challenge verification tag.

Similar to [85] , we can offer the following dynamic operations on outsourced data: update, delete, batch update, batch delete, append, and batch append. Note that we use the same mechanism presented in [85].

We now describe how to incorporate this "dynamic operation on data" capability into our PDP solution.

6.2.2 Block Update

Figure 13 includes our update algorithm for a single data block. All the computations involved in this algorithm are executed in the Client (data owner) side. In a short, what this algorithm does is it detects every occurrence of the data block m_i (*which we want to replace with* m_i') on the remaining verification tags and then apply 2 x-or operations with the existing T_{xor} : It first does x-or with hmac of the same data block m_i (i.e., $H_k(i||m_i)$ to cancel out its effect on the verification tag and then performs another x-or with hmac of the updated data block, m_i'.All remaining steps are identical to the $GenTags()$ algorithm of our PDP solution.

Algorithm: Block Update

Assumption: *we want to update the i-th data block i.e.,* m_i *with* m_i'. *Also for simplicity assume that no verification tag has been used so far.*

1. *Client* → *Server: <UPDATE Request>*
2. *Server* → *Client:* $T(=< T_1 T_2 T_3 \dots T_x .. T_X >$
3. *For each tag,* T_x in T
   a. *Decrypt* T_x *by applying* $AE_{K_U}^{-1}(T_{xor}||x||k_1||R||ServerID)$
   b. *Identify which data blocks are involved in* T_x
   c. *If i-th data block is present in* T_x
      i. *retrieve* T_{xor} *from 3.a*
   d. T'_{xor}=$T_{xor} \oplus H_k(i||m_i) \oplus H_k(i||m_i')$
   e. $T_x^i = AE_{K_U}(T'_{xor}||x||k_1||R||ServerID)$
4. $T' =< T_1' T_2' T_3' \dots T_x' \dots T_X' >$
5. *Client* → *Server:*$< i, m_i', T' >$

Figure 13 Block Update Algorithm

6.2.3 Block Deletion

There might be a need to delete certain blocks of data from the outsourced data. However, we assume that the number of blocks to be deleted is small relative to the size of the outsourced data, because if large portions of the outsourced data have to be deleted then the best approach would be to rerun our PDP scheme on the new data. Furthermore deleting a large number of data blocks would also affect the detection probability.

For block deletion, we choose the same approach of our block update algorithm except for the following change. More precisely, we apply the same block update algorithm where the deleted data block will be replaced with a predetermined special data block.

So our block deletion algorithm would be similar to our block update algorithm presented in Figure 13 with the following change in step 3.d:

T'_{xor}=$T_{xor} \oplus H_k(i||m_i) \oplus H_k(i||m_i')$

is replaced with

T'_{xor}=$T_{xor} \oplus H_k(i||m_i) \oplus H_k(i||fixedDataBlock)$

6.2.4 Batching Updates and Deletions

For the client, it is not practical to execute the "block update" or "block deletion" algorithm each time for each single data block update/deletion since it involves the cost of updating all remaining verification tags. So instead of doing each single data block update/deletion separately, it would be more efficient if we could facilitate batching of multiple operations of block update and block deletion. Fortunately, it is possible in our case; any number of block updates and deletes can be performed at the cost of a single data block update or delete. For that, we only need to update the for-loop in our block update algorithm presented in Figure 13 to take into account both deletions and updates at the same time. Due to the simplicity of the needed modifications, we do not use a separate algorithm for describing it.

6.2.5 Block Append

Our approach to facilitate a single-block append operation is similar to that of [85]. In our PDP scheme, the Client already made a decision in advance about the total number of data blocks as well as total number of PDP challenges (and thus the total number of verification tags) just prior to outsourcing its data to the cloud. In other words, those numbers are hard-coded into our PDP scheme and thus cannot be changed. So in order to facilitate block append operation, we first make an assumption of the data structure of the outsourced data and then we would be able to perform "block append" operation as a "block update" operation.

Change in the logical data structure of the outsourced data:

Recall that in our PDP solution, outsourced data, $F(=< m_1 m_2 m_3 \dots m_i \dots m_d >)$ consists of d number of equal-sized contiguous data blocks. In order to support block append, we now consider that F has a logical bi-dimensional structure. In that case, a single data block append would be similar to adding the new data block to one of the original *d* data blocks in a round-robin fashion. More precisely if Client, later on, wants to append k number of more data blocks (i.e., $m_{d+1}\, m_{d+2} .. m_{d+k}$) where $k < d$, then a logical representation would be as follows:

Row \ Col	Original Data block	New data block Appended		
1	m_1	m_{d+1}		
2	m_2	m_{d+2}		
.	.	.		
k	m_k	m_{d+k}		
.	.			
d	m_d			

Figure 14 Logical representation of outsourced data for supporting append operation

Note that we consider this logical data structure for verification tag computation; for actual physical storage of outsourced data, they could be still stored as a sequential block of data.

By considering this logical data structure, we can execute the same "block update" algorithm to do a "block append" operation for any block of data. For instance, if we want to do a single append for an actual data block,$(d + k)$, we execute "block update" algorithm where we would use $(m_k||m_{d+k})$ to replace data block, m_k.

The main advantage of this solution is that we can just execute the "block update" operation to append blocks so we can even batch several appends. Notice also that the probability for any block to be selected during a challenge is still 1/d (because we still consider d rows of several blocks combined together). So, in practice, Client should execute a number of block appends as a batch operation.

On the downside, Server will now have to access more blocks against each PDP challenge and in the long run, it may become increasingly expensive for Server as the number of blocks appended to the database increases.

6.2.6 Discussion

Our "block update" algorithm is the core of all other dynamic operations (such as block delete, block append and batch operations on block update/delete or append) we proposed for our PDP solution and thus for our $DLAS_{PDP}$ solution.

Similar to [85], it is possible to also construct a solution for "bulk append" and "insert" operation, but it would be inefficient and thus we do not discuss it here.

Now we provide an overview on the performance and security for our proposed dynamic operations, focussing on the update operation.

Security Analysis:

The security of our block update algorithm follows directly from the security proof of our PDP solution presented in the previous chapter. Note that in our block update algorithm, the Client modifies all verification tags regardless of whether they contain the i-th block of data or not. More precisely, the Client at least re-encrypts all verification tags using authenticated encryption and thus the Server cannot identify which verification tags include the i-th block of data. In addition, similar to our previously proposed PDP solution, the Client (and also all cloud users) has to make sure that is is not reusing any PDP challenge. In this circumstance, the Server does not get any additional advantages that it could use to break the security of our solution.

This same security analysis is also applicable to our other dynamic operations such as block delete, block append, and batch operations that can be performed on them.

Performance Analysis:

Here we talk about the additional costs of adopting these dynamic operations into our previously proposed PDP solution for static data. Note that all of our dynamic operations would have the similar performance numbers. As we mentioned earlier, we suggest using batch operations whenever possible. Roughly, a dynamic operation (better to consider it as a batch operation) would incur an additional cost similar to the cost involved in the Setup phase (more precisely the $GenTags()$ algorithm) of our PDP solution (see chapter 5 for details). More

precisely, all the computation involved in our dynamic operation, is performed by a Client and is similar to the cost of $GenTags()$ algorithm.

In terms of communication overhead, the additional cost involved in a dynamic operation is:

$$communication\ overhead\ involved\ in\ GenTags()\ algorithm\ +\ the\ size\ of\ T$$

$where\ T(=< T_1 T_2 T_3 .. T_x >)$ represents verification tags for all PDP challenges.

Finally, our solution for offering dynamic operations on data does not include any additional storage overhead.

6.3 $\mathbf{DLAS^{\Delta}_{PDP}}$: A DLAS solution with the possible optimal security guarantee

In this section, we discuss our final DLAS solution (we name it as $\mathrm{DLAS}^{\Delta}_{\mathrm{PDP}}$) that has the potential to offer the optimal security guarantee. Similar to our previously proposed $DLAS_{PDP}$ solution**,** the $\mathrm{DLAS}^{\Delta}_{\mathrm{PDP}}$ solution is based on a PDP solution; we name that PDP solution as $^{\Delta}_{\mathrm{PDP}}$**.** Our main intention here is to identify the important characteristics of $^{\Delta}_{\mathrm{PDP}}$ ***and*** $\mathrm{DLAS}^{\Delta}_{\mathrm{PDP}}$ solution; more specifically here our main focus is to identify two things: (i) what are the key characteristics of the system and the attack model? and (ii) how should a solution of this kind be constructed? (In particular, what are the key requirements for constructing such a solution?) Note that the efficiency of the $^{\Delta}_{\mathrm{PDP}}$ ***and*** $\mathrm{DLAS}^{\Delta}_{\mathrm{PDP}}$ solution is not the main goal here.

6.3.1 System & Attack Model

As discussed earlier, it is not possible to propose a DLAS solution for a totally dishonest server model. We identify the Distributed occasionally-malicious-but-constrained Attack Model (see page 77) considered for the previously proposed PDP and $DLAS_{PDP}$ solution as the most adversarial attack model possible for the existing cloud storage environments as here a storage server will correctly follow the protocol but can communicate with other servers to

generate a false data possession proof. Making the attack model more adversarial will cause the server to not follow the protocol correctly and thus make the server totally dishonest, in which case it would not be possible to offer any DLAS solution.

However, still there is scope to further improve the $PDP\ and\ DLAS_{PDP}$ solution with respect to the system model (i.e., DLASSystemModel3 on page 74). Recall that the key element of the PDP and $DLAS_{PDP}$ solution is the shared HMAC key generation by the TPM attached to the Server from its private key. Furthermore, the HMAC computation of each data block can be performed inside the TPM (all TPMs have built-in HMAC capabilities). Here our intention is to come up with an improved version of $DLAS_{PDP}$ (we already named it as $DLAS_{PDP}^{\Delta}$) which would work for the same attack model (i.e., DLASAttackModel3) as $DLAS_{PDP}$ but with a more robust system model than DLASSystemModel3. We name this system model as $\boldsymbol{DLASSystemModel3^{\Delta}}$ where the sole purpose of the TPM would be to bind a server with public key and nothing else. The other properties of $DLASSystemModel3^{\Delta}$ are the same as DLASSystemModel3.

Devising an instance of $^{\Delta}_{PDP}\ and\ thus\ DLAS_{PDP}^{\Delta}$ would be very challenging due to its underlying system and attack model (mainly the system model). Due to the challenge involved in the $DLASSystemModel3^{\Delta}$, the following requirements must be met to devise an instance of $^{\Delta}_{PDP}$ and $DLAS_{PDP}^{\Delta}$ solution**:**

- During the proof generation phase, now the storage server must directly use its private key (instead of using any other value created from the private key) in order to bind each challenged data block.

Now achieving an instance of $^{\Delta}_{PDP}\ and\ DLAS_{PDP}^{\Delta}$ would implicitly offer the following security guarantee:

"Given that a Server is following the protocol correctly, it will not be able to generate a false proof even if it communicates with other storage servers. The only way to generate a false PDP proof would be if a Server shares its private key with other Servers."

We think this is the optimal security guarantee a DLAS solution (also true for a PDP solution for cloud storage environments) can offer for the existing cloud storage environments.

6.3.2 Key Properties of Proposed Solution

$\overset{\Delta}{PDP}$ *and* $DLAS^{\Delta}_{PDP}$ solution for the above system and attack model must meet the following requirements in order to offer the optimal security guarantee:

- In the challenge phase, Server must use its private key on each challenged block of data during generating its data possession proof.
- The data possession proof generated by Server should be fresh. In other words, Server must not be able to provide an old proof of data possession for a particular PDP challenge to Client without being detected.
- Similar to all other PDP solutions (as well as our proposed DLAS solutions), it should also guarantee the integrity of the outsourced data at that point in time.
- Additionally, the Server's data possession proof should come with a non-repudiation guarantee.

6.3.3 Proposed Solution Description

For demonstration purposes, here we present a rough sketch of a $\overset{\Delta}{PDP}$ solution. The procedure for building a $\text{DLAS}^{\Delta}_{\text{PDP}}$ solution from the $\overset{\Delta}{PDP}$ solution would be identical to the procedure described in the previous chapter and thus will not be discussed here.

Whereas we can think of several constructions of a $\overset{\Delta}{PDP}$ (and thus $\text{DLAS}^{\Delta}_{\text{PDP}}$) solution that can fulfill the above requirements, here we discuss how we can construct a $\overset{\Delta}{\text{PDP}}$ solution by modifying our previously proposed PDP solution.

$\overset{\Delta}{PDP}$ construction through changing our PDP solution: We need to make the following changes to our previously proposed PDP solution:

Purpose for HMAC: The key component of our previously proposed PDP solution was the usage of a shared secret which not only binds the data possession proof to the Server but also offers integrity and freshness guarantees of the data possession proof. Even though we can still

use the same mechanism, it is not mandatory for $\overset{\Delta}{PDP}$. For a $\boldsymbol{DLAS}^{\Delta}_{\boldsymbol{PDP}}$ solution, HMAC would be only used for ensuring the integrity and freshness guarantees of the data possession proof, not for binding the data possession proof to the Server. So here client can alternatively choose a secret only known to him/her as the HMAC key.

Signing Each Block of Fresh Challenged Data to Bind the Data Possession Proof to Server: This can be considered as a major change over our previously proposed PDP solution. In our PDP solution, Server does the signing of T'_{xor} (i.e., xor of all tags involved in a PDP challenge). However in $\overset{\Delta}{PDP}$ solution, the Server signs each data block involved in a PDP challenge. In order to ensure that the Server always does a fresh signing on each data block, a fresh nonce would be appended with the data block prior to the signing.
So it implies the following changes:

1. Client has to share the information to Server of how those nonces can be generated. Similar to [76], Client can use a k bit random number, we call it k_2 to generate that nonce for each data block. In the Setup phase, Client can add this k_2 into its tag (see GenTags() algorithm),

   $T_x = AE_{K_U}(T_{xor}\ ||x||k_1||k_2||R||ServerID)\ \ where\ 1 \leq x \leq X$

   Also, Client now applies the HMAC operation on $(W_i||m_i)$ instead of m_i (see step 1.3.2 of $GenTags()$ algorithm) where $W_i \leftarrow H(k_2||i)$. Note that Server applies the HMAC operation in the same way as in step 2.2 of $GenProof()$ Algorithm

2. In the $GenProof()$ algorithm of the Challenge phase, Server does the signing of each challenged data as follows:

   For each data block $m_i\ where\ 1 \leq i \leq c$:

   $W_i \leftarrow H(k_2||i)$

   $M_i \leftarrow W_i||m_i$

   $Hash_{M_i} \leftarrow H(M_i)$

   $Sig_{M_i} \leftarrow Sign(Hash_{M_i})$

   bundles $(M_i\ ||\ Sig_{M_i})$

As a data possession proof, along with T'_{xor}, now Server sends to Client

$(M_i \,||\, Sig_{M_i})$ for $all\ i, where\ 1 \leq i \leq c$

3. In the $CheckProof$ algorithm of the Challenge Phase, now client essentially does the following 2 things:
    a. Compares $T'_{xor}\ with\ T_{xor}$ and makes a decision about the integrity and freshness of the Server's data possession proof for that PDP challenge
    b. Verifies Server's signature for each challenged data block to confirm whether the data possession proof is really bound to Server. It also gives the non-repudiation guarantee.

$DLAS^{\Delta}_{PDP}$ construction from $^{\Delta}_{PDP}$: As mentioned earlier, $DLAS^{\Delta}_{PDP}$ can be constructed from $^{\Delta}_{PDP}$ in the same way $DLAS_{PDP}$ was constructed from PDP solution and thus will not be discussed here.

6.3.4 Discussion

Here we discuss informally the key points of this proposed solution in regards to its security and performance (even though the efficiency of this proposed solution is not our main focus here).

Security: Similar to our previously proposed $PDP\ and\ DLAS_{PDP}$ solution, this proposed $^{\Delta}_{PDP}$, and $DLAS^{\Delta}_{PDP}$ solution can offer 128 bits of security. In both cases ($DLAS_{PDP}\ and\ DLAS^{\Delta}_{PDP}$), our solutions achieve the same level of integrity and freshness guarantee of the data possession proof generated by the Server. However, in this proposed solution ($^{\Delta}_{PDP}$, and $DLAS^{\Delta}_{PDP}$), we achieve the optimal "proof of data possession" binding guarantee (i.e., the Server does really have the possession of data) since by doing a signature on each single block of challenged data, the Server confirms the usage of its private key on each single block of challenged data in the process of generating the data possession proof for each single PDP challenge. In addition, all the security claims we made in the previous solution are also applicable here. The proofs of those security claims are also the same.

Performance: Due to its optimal "proof of data possession" binding guarantee, there is a noticeable overhead of communication and computation involved in this proposed $\overset{\Delta}{}_{PDP}$ and $DLAS^{\Delta}_{PDP}$ solution (in contrast to the previously proposed $PDP\ and\ DLAS_{PDP}$ solutions). The root cause for this is that now the server has to sign each data block involved in a PDP challenge. Along with the tag, T'_{xor}, it has to send the following to the Client:

$(M_i \,||\, Sig_{M_i})$ for $all\ i\ where\ 1 \leq i \leq c$

Definitely, it can be considered as a required trade off to achieve the optimal security guarantee (more precisely optimal "proof of data possession" binding guarantee).

6.4 Conclusion

In this chapter, we discussed how we can add features such as "public verifiability" and "dynamic operation on outsourced data" into our PDP solution and thus in our $DLAS_{PDP}$ solution. Here our main focus was to show how those capabilities can be added into our proposed solution; not on achieving optimal solutions for those capabilities. Furthermore, we also discussed here an ideal DLAS solution that has the potential to offer optimal security guarantees; in this case, the intention was to provide a solid base for any future research attempt of doing further enhancement of a DLAS solution.

7 Conclusions & Future Work

7.1 Conclusion

This thesis work focuses on the very challenging and previously unsolved Data Location Assurance (DLA) problem of the cloud storage environments in geo-replication of data setting. The main objective of our research is to come up with good and efficient solutions for this data location assurance problem of cloud storage environment which will not only work in geo-replication of data setting of cloud storage environments but also allows users to receive verifiable location guarantees about their data from the Cloud Storage Providers. This is a much-needed solution, especially for all organizations dealing with sensitive data in order to adopt any cloud storage solution (for reasons: see section 1.3.1).

As we mentioned in section 1.5, it is almost impossible to offer a DLAS solution for a totally dishonest cloud server model**.** So by following our research methodology (i.e., section 1.5) and solution strategy (i.e., section 2.6.2), we first define a number of system and attack models for the cloud storage environments and then propose a DLAS solution for each of the identified system and attack models (progressively going from a less adversarial system and attack model to more adversarial ones). In addition, we also propose the first PDP scheme specifically built for the cloud storage environment and then show how this PDP solution can be used as the main building block for constructing a DLAS solution. We consider this PDP solution as one of the most significant contributions of this thesis work. Unlike existing PDP solutions, our proposed PDP solution can not only bind the data possession proof to a particular storage server (known as authoritative storage server) but also is more efficient than most of the existing PDP solutions.

Through this thesis work, we define 4 system models and 3 attack models by closely observing existing cloud storage environments (including cloud storage applications as well as cloud storage providers) and propose 4 DLAS solutions: (i) DLAS (see chapter 3), (ii) HDLAS (see chapter 4), (iii) $DLAS_{PDP}$ based on our proposed PDP (see chapter 5) for those

different system models and attack models and (iv) $\text{DLAS}^{\Delta}_{\text{PDP}}$ (see section 6.3) that has the potential to offer the optimal security guarantee. Note that one of our key strategies for proposing those DLAS solutions is to identify limitations (in terms of system and attack model and meeting the requirements for existing cloud storage applications) of our previously proposed DLAS solution and then try to meet those identified limitations in our later proposal. Table 4 includes a brief summary of our proposed solutions. This table should also help to understand better the flow of our thesis work.

In addition, we propose some further enhancements for our proposed solutions in chapter 6, which includes adding features such as dynamic operation on data (see section 6.2), and public verifiability (see section 6.1) in our PDP solution (and thus in $DLAS_{PDP}$).

Furthermore, in our proposed $DLAS_{PDP}$ and $DLAS^{\Delta}_{PDP}$ solutions, the provable data possession guarantee from the Server also comes with a non-repudiation guarantee and it would be very useful in resolving any conflicts raised due to a violation of cloud user's data location preference mentioned in the SLA. The objective of the $DLAS_{PDP}$ (and the underlying PDP solution for the cloud) is to come up with a very efficient solution. On the other hand, the main objective of $DLAS^{\Delta}_{PDP}$ solution is to propose an ideal solution that can offer the optimal security guarantee; this solution can be used as a foundation for the future work on DLAS.

We believe, depending on the need, any potential cloud storage application should be able to choose a solution for the DLA problem from one of our proposed DLAS solutions. Our proposed solutions support the existing key practices among cloud storage providers such as AWS S3, Microsoft Azure, and so on. For instance, our proposed DLAS solution (i.e., $DLAS_{PDP}$ *and* $DLAS^{\Delta}_{PDP}$) supports distributed data storage practice in CSPs. Unlike many existing solutions [44] [110] [43], our proposed solutions do not put any restriction on the format of the data- they work for both encrypted and unencrypted data.

We believe our proposed solutions would help in the proliferation of cloud storage business by attracting more organizations (who need data location assurance guarantee) to

adopt cloud storage services. Additionally, our proposed DLAS solutions would enhance the trust level of users towards the CSP by offering the data location assurance service to the user. Similar to DLAS, there is other work [88] [111] which can also provide assurance to the cloud users in terms of data integrity (the data is intact and still available in the server) [88] and computational integrity (the computation done by the cloud server is accurate) [111]. Like [88] [111], we believe DLAS has also the potential to be used as one of the building blocks of the "Accountable Cloud [102]."[2]

Table 5 A summary of all proposed solutions

Solution Name	System Model & Attack Model	Cryptographic Primitives Used	Key Characteristics & Additional Comments
DLAS	**System Model**: 3-entity system model **Attack Model:** Honest-but-curious	Zero Knowledge Sets (ZKS) Cipher Policy-Attribute Based Encryption (CP-ABE)	• First DLAS solution in geo-replication of data setting • Data Centers' location might not be a public information • Secure and feasible for real deployment • Publication: 1 Journal paper and 1 IEEE conference paper
HDLAS	**System Model**: 2-entity system model **Attack Model**: Occasionally-malicious-but–constrained (OMBC)	Proof of Data Possession (PDP) Trusted Platform Module (TPM) GPS (Global Positioning System)	• Improvement over DLAS: ✓ Added data possession verification ✓ Supports more use cases ✓ Suitable for more adversarial system & attack model than DLAS • PDP ensures data possession guarantees • GPS receiver is also used to get accurate geo-location of the storage server • TPM is also used for remote attestation • Secure and feasible for real deployment • Publication: 1 IEEE conference paper

[2] Accountable cloud, a concept proposed by Haeberlen [70], means that a cloud should be made accountable to both the cloud user and the provider.

Solution Name	System Model & Attack Model	Cryptographic Primitives Used	Key Characteristics & Additional Comments
A PDP Solution for Cloud and **$DLAS_{PDP}$**	**System Model**: DLASSystemModel3 (an improved 2-entity system model) **Attack Model**: Distributed OMBC	Elliptic Curve Crypto TPM 2.0 Authenticated Encryption HMAC Digital Signature	• Improvement over HDLAS ✓ Unlike HDLAS, TPM has not been heavily used here. For instance, TPM is not used for remote attestation ✓ More adversarial system and attack model than HDLAS • Existing PDP has limitations; a PDP solution is required for the Cloud Storage Environment • First PDP solution for the Cloud Storage Environment is proposed • Then $DLAS_{PDP}$ is constructed using our proposed PDP • Initially intended for static data; also does not support public verifiability • Very efficient compared to existing solutions; offers 128-bit security and extendable • Further Enhancements: ✓ Public Verifiability ✓ Dynamic operations on data
$DLAS^{\Delta}_{PDP}$	**System Model**: DLASSystemModel3Δ (further improvement over DLASSystemModel3) **Attack Model**: Distributed OMBC	TPM (used only for binding server's identity) Digital Signature	• An ideal solution that can offer the optimal security guarantee • Improvement over $DLAS_{PDP}$ ✓ Improved system model ✓ Achieving optimal security guarantee • Can be considered as the foundation for future work on DLAS • Not intended for high efficiency

In general, all of our proposed DLAS solutions offer a data possession guarantee which is valid at a particular point in time. Considering the way the existing CSPs work, we argued

in section 5.9 that if a cloud user makes PDP/ DLAS queries within a short interval of time (e.g., every 5 to 10 minutes), our proposed solutions would be sufficient to achieve the overall objective of this thesis. In particular, our proposed solution explicitly offers the following guarantee (see section 5.9 for details):

"The data is not where it is not supposed to be for a period longer than between 2 DLAS requests."

7.2 Future Work

The following are directions for future work:

The key feature of our proposed PDP solution is that it can bind data possession proof to the Server. In addition to that, it also offers non-repudiation guarantees from the Server. We expect these features (more specifically, binding the data possession proof to the server) to influence a number of future research attempts in enhancing many existing cryptographic solutions for cloud such as different variants of PDP schemes (e.g., multiple-replica PDP solution which aims to ensure that exactly n copies of outsourced data and not any less, are stored on cloud), Proof of Retrieval schemes and so on.

In this thesis, we initially limit our scope to proposing DLAS solutions for cloud storage applications that deal with static/archival data and where a data owner would make an attempt to get data possession guarantees for his/her outsourced data. However, in chapter 6, as feature enhancements of our proposed DLAS solution (more precisely for $DLAS_{PDP}$), we also provide solutions for adding capabilities such as "dynamic operation on data" and "public verifiability". There is a good scope for conducting future research on doing further enhancement (in terms of efficiency) of dynamic operation on data, and public verifiability capabilities for our proposed solutions.

Last but not least, future research work can always be conducted to devise more efficient DLAS solutions for our defined system and attack models for the DLA problem. Our $DLAS^{\Delta}_{PDP}$ solution can play a significant role in this regard by offering a foundation for any future enhancement on DLAS.

REFERENCES

[1] G. Anthes, "Security in the cloud," *Communications of the ACM,* vol. 53, no. 11, pp. 16-18, 2010.

[2] J. Aikat, A. Akella, J. S. Chase, A. Juels, M. K. Reiter, T. Ristenpart, V. Sekar and M. Swift, "Rethinking Security in the Era of Cloud Computing," *IEEE Security Privacy,* vol. 15, pp. 60-69, 2017.

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

[4] L. Youseff, M. Butrico and D. Da Silva, "Toward a unified ontology of cloud computing," in *Grid Computing Environments Workshop, 2008. GCE'08*, 2008.

[5] L. M. Vaquero, L. Rodero-Merino, J. Caceres and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review,* vol. 39, no. 1, pp. 50-55, 2008.

[6] P. Mell and T. Grance, "The NIST definition of cloud computing," September 2011. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-145/final. [Accessed 15 December 2017].

[7] "Amazon Simple Storage Service (Amazon S3)," Amazon , [Online]. Available: https://aws.amazon.com/s3/. [Accessed 15 12 2015].

[8] "Microsoft Azure," Microsoft, [Online]. Available: https://azure.microsoft.com/en-us/. [Accessed 15 12 2015].

[9] "Google Cloud Storage," [Online]. Available: https://cloud.google.com/storage/. [Accessed 01 November 2017].

[10] I. M. Khalil, A. Khreishah and M. Azeem, "Cloud computing security: A survey," *Computers,* vol. 3, no. 1, pp. 1-35, 2014.

[11] S. A. de Chaves, C. B. Westphall, C. M. Westphall and G. A. Ger{\^o}nimo, "Customer security concerns in cloud computing," in *Proc. The Tenth International Conf. on Networks (ICN), The Netherlands*, 2011.

[12] J. Heiser and M. Nicolett, "Assessing the security risks of cloud computing," *Gartner Report,* 2008.

[13] K. Popovi{\'c} and others, "Cloud computing security issues and challenges," in *MIPRO, 2010 proceedings of the 33rd international convention*, 2010.

[14] Y. Chen, V. Paxson and R. H. Katz, "What's new about cloud computing security," *University of California, Berkeley Report No. UCB/EECS-2010-5 January,* vol. 20, no. 2010, pp. 2010-5, 2010.

[15] A. Noman and C. Adams, "DLAS: Data Location Assurance Service for cloud computing environments," in *Privacy, Security and Trust (PST), 2012 Tenth Annual International Conference on*, 2012.

[16] Z. N. Peterson, M. Gondree and R. Beverly, "A position paper on data sovereignty: the importance of geolocating data in the cloud," in *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, 2011.

[17] "Amazon AWS S3 outage is breaking things for a lot of websites and apps," [Online]. Available: https://techcrunch.com/2017/02/28/amazon-aws-s3-outage-is-breaking-things-for-a-lot-of-websites-and-apps/. [Accessed 15 November 2017].

[18] "The day Amazon S3 storage stood still," [Online]. Available: https://techcrunch.com/2017/03/01/the-day-amazon-s3-storage-stood-still/. [Accessed 15 November 2017].

[19] B. Biswal, S. Shetty and T. Rogers, "Classification Based IP Geolocation Approach to Locate Data in the Cloud Datacenters," 2014.

[20] "Dropbox Service Unavailble News," [Online]. Available: http://www.zdnet.com/dropbox-users-experiencing-upload-problems-aws-questioned-again-7000009680/. [Accessed 15 12 2015].

[21] A. Albeshri, C. Boyd and J. G. Nieto, "Geoproof: proofs of geographic location for cloud computing environment," in *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*, 2012.

[22] M. Henze, R. Matzutt, J. Hiller, E. Mühmer, J. H. Ziegeldorf, J. v. d. Giet and K. Wehrle, "Practical Data Compliance for Cloud Storage," in *2017 IEEE International Conference on Cloud Engineering (IC2E)*, 2017.

[23] G. J. Watson, R. Safavi-Naini, M. Alimomeni, M. E. Locasto and S. Narayan, "LoSt: location based storage," in *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*, 2012.

[24] G. Wang, Q. Liu and J. Wu, "Achieving fine-grained access control for secure data sharing on cloud servers," *Concurrency and Computation: Practice and Experience,* vol. 23, no. 12, pp. 1443-1464, 2011.

[25] M. Li, S. Yu, K. Ren and W. Lou, "Securing personal health records in cloud computing: Patient-centric and fine-grained data access control in multi-owner settings," in *Security and Privacy in Communication Networks*, Springer, 2010, pp. 89-106.

[26] J. Li, G. Zhao, X. Chen, D. Xie, C. Rong, W. Li, L. Tang and Y. Tang, "Fine-grained data access control systems with user accountability in cloud computing," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010.

[27] S. Yu, C. Wang, K. Ren and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *INFOCOM, 2010 Proceedings IEEE*, 2010.

[28] S. D. C. Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi and P. Samarati, "Over-encryption: management of access control evolution on outsourced data," in *Proceedings of the 33rd international conference on Very large data bases*, 2007.

[29] C. Esposito, A. Castiglione and K. K. R. Choo, "Encryption-Based Solution for Data Sovereignty in Federated Clouds," *IEEE Cloud Computing,* vol. 3, pp. 12-17, Jan 2016.

[30] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. u. Haq, M. I. u. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan and L. Rigas,

"Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, New York, NY, USA, 2011.

[31] "Trusted Computing Group.," Trusted Computing Group. , [Online]. Available: http://www.trustedcomputinggroup.org. [Accessed 15 12 2015].

[32] R. Perez, R. Sailer, L. van Doorn and others, "vTPM: virtualizing the trusted platform module," in *Proc. 15th Conf. on USENIX Security Symposium*, 2006.

[33] "Trusted Platform Module Library Part 1:Architecture," 2014.

[34] B. Gueye, A. Ziviani, M. Crovella and S. Fdida, "Constraint-Based Geolocation of Internet Hosts," *IEEE/ACM Transactions on Networking,* vol. 14, pp. 1219-1232, Dec 2006.

[35] J. Chen, F. Liu, X. Luo, F. Zhao and G. Zhu, "A Landmark Calibration Based IP Geolocation Approach," in *Availability, Reliability and Security (ARES), 2015 10th International Conference on*, 2015.

[36] J. Chen, F. Liu, T. Wang, X. Luo, F. Zhao and G. Zhu, "Towards region-level IP geolocation based on the path feature," in *Advanced Communication Technology (ICACT), 2015 17th International Conference on*, 2015.

[37] M. Fotouhi, A. Anand and R. Hasan, "PLAG: Practical Landmark Allocation for Cloud Geolocation," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, 2015.

[38] S. Micali, M. Rabin and J. Kilian, "Zero-knowledge sets," in *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, 2003.

[39] D. Catalano, D. Fiore and M. Messina, "Zero-knowledge sets with short proofs," in *Advances in Cryptology--EUROCRYPT 2008*, Springer, 2008, pp. 433-450.

[40] R. Gennaro and S. Micali, "Independent zero-knowledge sets," in *Automata, Languages and Programming*, Springer, 2006, pp. 34-45.

[41] J. Bethencourt, A. Sahai and B. Waters, "Ciphertext-policy attribute-based encryption," in *Security and Privacy, 2007. SP'07. IEEE Symposium on*, 2007.

[42] "Advanced Crypto Software Collection," [Online]. Available: http://acsc.cs.utexas.edu/cpabe/. [Accessed 15 12 2015].

[43] 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*, 2007.

[44] 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*, 2009.

[45] R. Curtmola, O. Khan and R. Burns, "Robust remote data checking," in *Proceedings of the 4th ACM international workshop on Storage security and survivability*, 2008.

[46] Y. Dodis, S. Vadhan and D. Wichs, "Proofs of retrievability via hardness amplification," in *Theory of cryptography*, Springer, 2009, pp. 109-127.

[47] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Advances in Cryptology-ASIACRYPT 2008*, Springer, 2008, pp. 90-107.

[48] J. Xu and E.-C. Chang, "Towards Efficient Proofs of Retrievability," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, New York, NY, USA, 2012.

[49] A. Menezes, "An introduction to Pairing BAsed Cryptography," [Online]. Available: https://www.math.uwaterloo.ca/~ajmeneze/publications/pairings.pdf. [Accessed 10 December 2017].

[50] R. Dutta, R. Barua and P. Sarkar, "Pairing-based cryptography: A survey," 2004.

[51] M. Bellare, P. Rogaway and D. Wagner, *EAX: A Conventional Authenticated-Encryption Mode,* 2003.

[52] M. Bellare, T. Kohno and C. Namprempre, "Authenticated Encryption in SSH: Provably Fixing the SSH Binary Packet Protocol," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2002.

[53] H. Krawczyk, "The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)," in *Advances in Cryptology --- CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19--23, 2001 Proceedings*, J. Kilian, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 310-331.

[54] C. S. Jutla, "Encryption Modes with Almost Free Message Integrity," in *Advances in Cryptology --- EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6--10, 2001 Proceedings*, B. Pfitzmann, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 529-544.

[55] P. Rogaway, M. Bellare and J. Black, "OCB: A Block-cipher Mode of Operation for Efficient Authenticated Encryption," *ACM Trans. Inf. Syst. Secur.,* vol. 6, pp. 365-403, #aug# 2003.

[56] J. a. C. A. Salowe and D. McGrew, "AES Galois Counter Mode (GCM) Cipher Suites for TLS," Network Working Group, 2008.

[57] M. Bellare and C. Namprempre, "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm," *Journal of Cryptology,* vol. 21, pp. 469-491, Oct 2008.

[58] B. Schneier, Applied Cryptography Protocols, Algorithms, and Source Code in C, Wiley, 1996.

[59] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond and M. Morrow, "Blueprint for the Intercloud - Protocols and Formats for Cloud Computing Interoperability," in *2009 Fourth International Conference on Internet and Web Applications and Services*, 2009.

[60] N. Grozev and R. Buyya, "Inter-Cloud architectures and application brokering: taxonomy and survey," *Software: Practice and Experience,* vol. 44, pp. 369-390, 2014.

[61] "Intel IT Center,“Peer Research: What’s Holding Back the Cloud?” Tech.," 2012.

[62] P. Gill, Y. Ganjali, B. Wong and D. Lie, "Dude, where’s that IP?: circumventing measurement-based IP geolocation," in *Proceedings of the 19th USENIX conference on Security*, 2010.

[63] D. L. Fu, X. G. Peng and Y. L. Yang, "Trusted Validation for Geolocation of Cloud Data," *The Computer Journal,* p. bxu144, 2014.

[64] M. Gondree and Z. N. Peterson, "Geolocation of data in the cloud," in *Proceedings of the third ACM conference on Data and application security and privacy*, 2013.

[65] C. Jaiswal and V. Kumar, "IGOD--Identifying Geolocation of Cloud Datacenter Hosting Mobile User's Data," in *Mobile Data Management (MDM), 2015 16th IEEE International Conference on*, 2015.

[66] N. Paladi, M. Aslam and C. Gehrmann, "Trusted Geolocation-Aware Data Placement in Infrastructure Clouds," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*, 2014.

[67] A. Albeshri, C. Boyd and J. G. Nieto, "Enhanced GeoProof: improved geographic assurance for data in the cloud," *International Journal of Information Security,* vol. 13, no. 2, pp. 191-198, 2014.

[68] M. Thangavel, P. Varalakshmi, R. Sindhuja and S. Sridhar, "A survey on provable data possession in cloud storage," in *2016 Eighth International Conference on Advanced Computing (ICoAC)*, 2017.

[69] A. F. Barsoum and M. A. Hasan, "Provable possession and replication of data over cloud servers," *Centre For Applied Cryptographic Research (CACR), University of Waterloo, Report,* vol. 32, p. 2010, 2010.

[70] H. Wang, "Proxy Provable Data Possession in Public Clouds," *IEEE Transactions on Services Computing,* vol. 6, pp. 551-559, Oct 2013.

[71] R. Curtmola, O. Khan, R. Burns and G. Ateniese, "MR-PDP: Multiple-replica provable data possession," in *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on*, 2008.

[72] Z. Hao and N. Yu, "A multiple-replica remote data possession checking protocol with public verifiability," in *Data, Privacy and E-Commerce (ISDPE), 2010 Second International Symposium on*, 2010.

[73] A. F. Barsoum and M. A. Hasan, "On Verifying Dynamic Multiple Data Copies over Cloud Servers.," *IACR Cryptology ePrint Archive,* vol. 2011, p. 447, 2011.

[74] W.-F. Hsien, C.-C. Yang and M.-S. Hwang, "A survey of public auditing for secure data storage in cloud computing," vol. 18, pp. 133-142, 01 2016.

[75] C. Liu, R. Ranjan, X. Zhang, C. Yang, D. Georgakopoulos and J. Chen, "Public Auditing for Big Data Storage in Cloud Computing -- A Survey," in *2013 IEEE 16th International Conference on Computational Science and Engineering*, 2013.

[76] 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*, 2007.

[77] Y. Deswarte, J.-J. Quisquater and A. Sa{\"\i}dane, "Remote integrity checking," *Proceedings of IICIS,* vol. 140, pp. 1-11, 2003.

[78] D. L. Gazzoni Filho and P. S. L. M. Barreto, "Demonstrating data possession and uncheatable data transfer.," *IACR Cryptology ePrint Archive,* vol. 2006, p. 150, 2006.

[79] P. Golle, S. Jarecki and I. Mironov, "Cryptographic primitives enforcing communication and storage complexity," in *Financial Cryptography*, 2003.

[80] F. Seb{\'e}, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte and J.-J. Quisquater, "Efficient remote data possession checking in critical information infrastructures," *Knowledge and Data Engineering, IEEE Transactions on,* vol. 20, no. 8, pp. 1034-1038, 2008.

[81] M. A. Shah, M. Baker, J. C. Mogul, R. Swaminathan and others, "Auditing to Keep Online Storage Services Honest.," in *HotOS*, 2007.

[82] M. A. Shah, R. Swaminathan and M. Baker, "Privacy-Preserving Audit and Extraction of Digital Contents.," *IACR Cryptology ePrint Archive,* vol. 2008, p. 186, 2008.

[83] K. Zeng, "Publicly verifiable remote data integrity," in *Information and Communications Security*, Springer, 2008, pp. 419-434.

[84] H. Yan, J. Li, J. Han and Y. Zhang, "A Novel Efficient Remote Data Possession Checking Protocol in Cloud Storage," *IEEE Transactions on Information Forensics and Security,* vol. 12, pp. 78-88, Jan 2017.

[85] G. Ateniese, R. Di Pietro, L. V. Mancini and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, 2008.

[86] C. C. Erway, A. K{\"u}p{\c{c}}{\"u}, C. Papamanthou and R. Tamassia, "Dynamic provable data possession," *ACM Transactions on Information and System Security (TISSEC),* vol. 17, no. 4, p. 15, 2015.

[87] Z. Hao, S. Zhong and N. Yu, "A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability," *Knowledge and Data Engineering, IEEE transactions on,* vol. 23, no. 9, pp. 1432-1437, 2011.

[88] Q. Wang, C. Wang, K. Ren, W. Lou and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *Parallel and Distributed Systems, IEEE Transactions on,* vol. 22, no. 5, pp. 847-859, 2011.

[89] C. Gritti, W. Susilo and T. Plantard, "Efficient Dynamic Provable Data Possession with Public Verifiability and Data Privacy," 2015.

[90] Y. Zhu, H. Hu, G.-J. Ahn and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," *Parallel and Distributed Systems, IEEE Transactions on,* vol. 23, no. 12, pp. 2231-2244, 2012.

[91] F. Armknecht, L. Barman, J.-M. Bohli and G. O. Karame, "Mirror: Enabling Proofs of Data Replication and Retrievability in the Cloud," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, Austin, 2016.

[92] J. He, Y. Zhang, G. Huang, Y. Shi and J. Cao, "Distributed data possession checking for securing multiple replicas in geographically-dispersed clouds," *Journal of Computer and System Sciences,* vol. 78, pp. 1345-1358, 2012.

[93] N. Paladi, C. Gehrmann and F. Morenius, "State of The Art and Hot Aspects in Cloud Data Storage Security," 2013.

[94] N. Paladi and A. Michalas, "“One of our hosts in another country”: Challenges of data geolocation in cloud storage," in *Wireless Communications, Vehicular Technology, Information Theory and Aerospace \& Electronic Systems (VITAE), 2014 4th International Conference on*, 2014.

[95] C. Krau{\ss} and V. Fusenig, "Using trusted platform modules for location assurance in cloud networking," in *Network and System Security*, Springer, 2013, pp. 109-121.

[96] E. Banks, M. Bartock, K. Firftal, D. Lemon, K. Scarfone, U. Shetty, M. Souppaya, T. Williams and R. Yeluri, "Draft trusted geolocation in the cloud: Proof of concept implementation," *NIST special publication,* vol. 7904, p. 42, 2012.

[97] N. Paladi, C. Gehrmann and A. Michalas, "Providing User Security Guarantees in Public Infrastructure Clouds," vol. PP, 01 2016.

[98] A. Noman and C. Adams, "Providing a data location assurance service for cloud storage environments," *Journal of Mobile Multimedia,* vol. 8, no. 4, pp. 265-286, 2013.

[99] A. Michalas and K. Y. Yigzaw, "LocLess: Do you Really Care Where Your Cloud Files Are?," in *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2016.

[100] J. Li, A. Squicciarini, D. Lin, S. Liang and C. Jia, "SecLoc: Securing Location-Sensitive Storage in the Cloud," in *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*, New York, NY, USA, 2015.

[101] M. Chase, A. Healy, A. Lysyanskaya, T. Malkin and L. Reyzin, "Mercurial Commitments with Applications to Zero-Knowledge Sets," *Advances in Cryptology - EUROCRYPT 2005,* vol. 3494, pp. 422-439, 2005.

[102] A. Haeberlen, "A case for the accountable cloud," *ACM SIGOPS Operating Systems Review,* vol. 44, no. 2, pp. 52-57, 2010.

[103] A. Noman and C. Adams, "Hardware-based DLAS: Achieving geo-location guarantees for cloud data using TPM and Provable Data Possession," in *Computer and Information Technology (ICCIT), 2014 17th International Conference on*, 2014.

[104] G. Ateniese, B. Randal, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson and D. Song, "Remote data checking using provable data possession," *ACM Transactions on Information and System Security (TISSEC),* vol. 14, no. 1, pp. 12:1-12:34, 2011.

[105] J. Black and P. Rogaway, "Ciphers with Arbitrary Finite Domains," in *Topics in Cryptology --- CT-RSA 2002: The Cryptographers´Track at the RSA Conference 2002 San Jose, CA, USA, February 18--22, 2002 Proceedings*, B. Preneel, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 114-130.

[106] E. C. J. Z. J Xu, "Towards Efficient Provable Data Possession in Cloud Storage," eprint.iacr.org, last accessed on Nov 15, 2017.

[107] D. Hankerson and A. Menezes, "Elliptic Curve Discrete Logarithm Problem," in *Encyclopedia of Cryptography and Security*, A. van Tilborg and Henk C. and S. Jajodia, Eds., Boston, MA: Springer US, 2011, pp. 397-400.

[108] "Bitcoin Wiki," [Online]. Available: https://en.bitcoin.it/wiki/Secp256k1. [Accessed 15 Nov 2017].

[109] D. Boneh, B. Lynn and H. Shacham, "Short signatures from the Weil pairing," in *Advances in Cryptology—ASIACRYPT 2001*, Springer, 2001, pp. 514-532.

[110] Y. Dodis, S. Vadhan and D. Wichs, "Proofs of Retrievability via Hardness Amplification," in *Theory of Cryptography: 6th Theory of Cryptography Conference,*

TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings, O. Reingold, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 109-127.

[111] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka and J. Molina, "Controlling data in the cloud: outsourcing computation without outsourcing control," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, 2009.

[112] O. Goldreich, "A Sample of Samplers: A Computational Perspective on Sampling," in *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation: In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, O. Goldreich, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 302-332.

APPENDIX 1 CLOUD COMPUTING HISTORY

This cloud computing history has been taken as is from [2].

Cloud Computing History: Modern cloud computing systems are the latest example of shared-infrastructure computing. The idea of "utility computing" on mainframes was a primary motivation for the Multics project, which inspired much early research on computer security. In such systems, users and businesses remotely accessed a central mainframe, amortizing the large cost over many users. With the rise of the World Wide Web in the late 1990s, public hosting centers grew in popularity and offered both bare-metal machines and managed web-servers. The free Linux OS and Apache webserver, launched in 1991 and 1995, respectively, and servers with commodity Intel and AMD processors, made it economical to run large server clusters to handle the increasing scale of Internet workloads.

Two innovations really drove the cloud revolution. First, VMware introduced a hypervisor for x86 processors, showing that virtualization could be efficient and useful. This was followed by the open source Xen project from Cambridge University in 2003. Second, Amazon launched the Elastic Compute Cloud that rented virtual machines with a new billing model based on short-term usage. For a few cents, a customer could rent a fraction of a physical machine for an hour. The combination of virtualization, efficient management, and short time-scale billing enabled cloud computing's boom.

Today's public clouds provide access to large-scale computing infrastructure in the form of virtual machines, applications, or software services. The common thread among all these is the dynamic sharing of infrastructure by multiple tenants, which is managed by a third party. In this common structure, clouds today can be divided roughly into three kinds. Infrastructure-as-a-service systems allow customers to launch virtual machines and control the guest OS as well as applications. In platform-as-a-service systems, customers can launch applications that execute in a provider-managed OS. Finally, a software-as-a-service cloud provides hosted application services for customers, such as email, databases, file storage, and the like. A single company, such as Google or Microsoft, might offer all three kinds of computing

APPENDIX 2 NIST DEFINITION OF CLOUD COMPUTING

It is taken from [6].

NIST 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. This cloud model is composed of five essential characteristics**,** three service models, and four deployment models.

Essential Characteristics:

On-demand self-service. A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

Broad network access. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

Resource pooling. The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.

Rapid elasticity. Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

Measured service. Cloud systems automatically control and optimize resource use by leveraging a metering capability1 at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored,

controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

Service Models:

Software as a Service (SaaS). The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure2. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

Platform as a Service (PaaS). The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider.3 The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

Infrastructure as a Service (IaaS). The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

Deployment Models:

Private cloud. The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.

Community cloud. The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.

Public cloud. The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.

Hybrid cloud. The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

APPENDIX 3 FURTHER EXPLANATION ON THE DETECTION PROBABILITY OF PDP SCHEME

Suppose the Client (i.e., the data owner) has uploaded *n* blocks to the Server (i.e., a storage server), where p number of data blocks have been modified or deleted. When the Client sends a PDP challenge to the Server with all necessary keys required by the Server to identify those c number of randomly chosen data blocks involved in that PDP challenge. Assume that the Server has already identified those *c* data blocks and among those c number of chosen data blocks, c_1 denote the number of data blocks chosen from p (where $c_1 \leq p$) number of modified or deleted blocks. In this case, it is obvious that the probability of detecting the data corruption event is equal to the probability of $c_1 \geq 1$.

Let P_a denote this probability value, then we have

$$P_a = P\{c_1 \geq 1\} = 1 - P\{c_1 < 1\} = 1 - P\{c_1 = 0\}$$

$$= 1 - \frac{n-p}{n} \cdot \frac{n-p-1}{n-1} \ldots \ldots \frac{n-p-c+1}{n-c+1}$$

It indicates that: $1 - \left(1 - \frac{p}{n}\right)^c \leq P_a \leq 1 - \left(1 - \frac{p}{n-c+1}\right)^c$

This equation demonstrates that in order to improve the error detection probability, the Server should adaptively increase the number of challenged data blocks especially when the proportion of corrupted data block is small. However, according to the evaluation conducted by Ateniese et al. in [76] [104], when 1% blocks of the file are corrupted, 300 challenged data blocks will achieve $P_a \geq 95\%$ and 460 data blocks will achieve $P_a \geq 99\%$.

Note that, in our proposed solutions, we have considered the 1% block of file corruption scenario and have tried to achieve $P_a \geq 99\%$ by using 460 number of random data blocks in each PDP challenge.