FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

On The Applicability Of Secret Sharing Cryptography In Secure Cloud Services

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Martin Kirchner
Matrikelnummer 0425028

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl
Mitwirkung: Dr. Dimitrios E. Simos

Wien, 13.01.2014

(Unterschrift Martin Kirchner)

(Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

On The Applicability Of Secret Sharing Cryptography In Secure Cloud Services

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Martin Kirchner
Registration Number 0425028

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl
Assistance: Dr. Dimitrios E. Simos

Vienna, 13.01.2014 (Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Martin Kirchner
Karolinengasse 6/10, 1040 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Martin Kirchner)

Danksagung

Ich danke meinen Eltern Sylvia und August sowie meinem Bruder Alexander dafür, dass sie mir durch ihre Unterstützung mein Studium ermöglicht haben.

Fur die gute Betreuung und die nützlichen Anregungen bei der Erstellung dieser Diplomarbeit möchte ich mich bei Dipl.-Ing. Aleksandar Hudic, Dr. Dimitrios Simos und Priv.-Doz. Dipl.-Ing. Mag. Dr. Edgar Weippl bedanken.

Abstract

Data outsourcing to third party providers enables significant cost and processing benefits. Therefore there is a high propensity of small and medium enterprises to outsource their data and services to third party service providers, but there is also a high concern when it comes to privacy of the outsourced data.

When referring to outsource sensitive information, outsourcing comes with an expensive price, it is entailed with security issues such as: lack of control of the outsourced data, vendor lock-in, malicious insiders, legislation issues, data locality, data and service availability etc.

The main goal of this thesis is to research the applicability of Secret Sharing Cryptography in cloud storage scenarios in order to solve the aforementioned challenges by designing models that distribute the shares across multiple providers. Providers with higher trust values will be preferred over less reliable ones.

In this thesis the idea of collaboration sets is introduced to describe how multiple providers may be bound by law for instance to reveal data and how we can solve this issue by taking this into account in our distribution strategies.

The research on the models was implemented in a proof-of-concept framework and existing secret sharing and multi-cloud libraries were evaluated for their performance and features to decide the feasibility of the framework in a real-world scenario.

Kurzfassung

Die Auslagerung von Daten in die Cloud ermöglicht signifikante Einsparungen von Kosten und nahezu unlimitierte Rechenleistung. Aus diesem Grund ist das Auslagern von Daten und für viele kleine und mittlere Unternehmen sehr verlockend, auch wenn es Ängste bezüglich der Speicherung von vertraulichen Informationen außerhalb des eigenen Unternehmens gibt.

Die größten Herausforderungen bei der Migration in die Cloud sind die mangelnde Kontrolle über die Daten, ein sogenannter "Vendor lock-in", bei dem man sich potenziell einem einzelnen Anbieter aufgrund fehlender Standards ausliefert, die Lokalität der Daten, rechtliche Problemzonen, Daten- und Serviceverfügbarkeit sowie "böswillige Insider bzw. Outsider", welche entweder in das System von außen einbrechen können oder direkt bei den Anbieter arbeiten.

Das Ziel dieser Arbeit ist die Anwendbarkeit von Secret Sharing Kryptographie in dieser Cloud Umgebung zu untersuchen, um die oben beschriebenen Herausforderungen lösen zu können. Dafür wurden Modelle entwickelt, welche eine Verteilung der von Secret Sharing generierten "Shares" auf verschiende Cloud Anbieter zu verteilen, sodass ein Anbieter alleine oder mehrere Anbieter zusammen die Originaldatei nicht rekonstruieren können. Dabei ermöglicht die Angabe von Vertrauenswerten einzelner Anbieter eine Präferierung verschiedener Anbieter gegenüber anderen.

In der Folge wird das Konzept der "Collaboration Sets" eingeführt, um Zusammenhänge zwischen mehreren Anbietern abbilden zu können, wie etwa Lokalität, um zu verhindern, dass zu viele Anbieter aus einem Land verwendet werden.

Diese Modelle wurden in der Folge in einer "Proof-of-concept"-Implementierung abgebildet, um deren Einsatzfähigkeit in praxisnahen Szenarien evaluieren zu können bezüglich der Leistung und Funktionalität.

Contents

1 Introduction **1**
1.1 Motivation, Problem Statement and Aim of the Work . . . 1
1.2 Methodological Approach . . . 2
1.3 Related Work . . . 2
1.4 Structure of the Work . . . 5

2 Cloud Computing **7**
2.1 What is Cloud Computing? . . . 7
2.2 Benefits And Challenges . . . 11
2.3 Interoperability . . . 18
2.4 Summary . . . 21

3 Secret Sharing Cryptography **23**
3.1 Introduction . . . 23
3.2 Shamir's Secret Sharing . . . 24
3.3 Adversary models . . . 26
3.4 Verifiable Secret Sharing . . . 27
3.5 Proactive Secret Sharing . . . 28
3.6 Social Secret Sharing . . . 31
3.7 Unencrypted Secret Sharing Techniques . . . 36
3.8 Summary . . . 37

4 Cloud Storage Models **39**
4.1 Motivation . . . 39
4.2 Model Overview . . . 40
4.3 A Model Without Collaboration . . . 40
4.4 A Model With Static Collaboration . . . 43
4.5 A Model With Dynamic Collaboration . . . 50
4.6 Summary . . . 52

5 A Secure Cloud Storage Framework **53**
5.1 Overview . . . 53
5.2 Transfer of Shares . . . 55

5.3 Critical Implementation Aspects 59
5.4 Summary 61

6 A Critical Reflection **63**
6.1 Comparison of Model 2 with Social Secret Sharing 63
6.2 Additional Extensions and Open Issues 64

7 Conclusion **67**

A Evaluation of Secret Sharing Libraries **69**
A.1 Evaluation of Secret Sharing Libraries 69
A.2 Detailed Performance Evaluation of GFShare 70

B Code Samples **73**
B.1 Decision Procedures and Test Runs 73
B.2 Additional Code Samples Related to Cloud Providers 79

Bibliography **81**

CHAPTER 1

Introduction

1.1 Motivation, Problem Statement and Aim of the Work

Data outsourcing to third party providers is nowadays an emerging paradigm which brings significant cost and processing benefits. Therefore there is a high propensity of small and medium enterprises to outsource their data and services to third party service providers, but there is also a high concern when it comes to privacy of the outsourced data. When we refer to outsourcing sensitive information, outsourcing comes with an expensive price, it is entailed with security issues such as: lack of control of the outsourced data, vendor lock-in, malicious insiders, legislation issues, data locality, data and service availability. Currently, privacy and security are primary related to trusted authorities.

According to a survey on behalf of Oracle [37] only roughly a third of the respondents could confirm that personally identifiable data was encrypted across all their systems and almost a third of the respondents admitted that they either had data breaches or that they did not know whether there were data breaches or not. In a cloud environment this level of protection may be insufficient, because the data you store on third party providers may be compromised from within the cloud service provider. For instance, a person within that organization with the required permissions and knowledge may gain access to your data if it is not encrypted before it is stored externally. There are also other concern regarding access and content control that is outside a company's influence when dealing with cloud service providers.

In order to lighten the security concerns and to enhance the privacy of the data, we will focus on theoretical and implementation aspects of secret-sharing cryptography applicable to cloud environments.

We will focus especially on answering the following questions:

- How to effectively distribute data across several Cloud providers in a safe manner?
- What effects do cryptographic primitives have on data distribution?

- Can the interoperability concepts, combined with cryptographic primitives, enhance security in third party environments?
- Can it be used for end-user scenarios that are time-sensitive? For example, uploading documents and images should not take a significantly longer time than without encryption mechanisms.

1.2 Methodological Approach

This thesis is focused on research and evaluation concepts of security and privacy with an emphasis on data outsourcing concepts to third party cloud service providers. In particular, we will look at current secret sharing concepts and provide a novel implementation of the current best practices with respect to secret sharing cryptography schemes applicable for cloud environments.

The first theoretical part of the thesis, consisting of Chapters 2 and 3 is related to the research and evaluation of current state-of-the-art technologies in respect to Cloud computing and secret sharing cryptography. We will introduce the basic principles of secret sharing, and more specifically the work that has been done on weighted social secret sharing and socio-rational secret sharing with respect to the cloud paradigm with particular focus on Cloud interoperability aspects such as multi-cloud, inter-cloud and federated cloud systems.

The second part, consisting of Chapters 4 and 5 is related to the definition of our cloud distribution models to achieve a secure cloud storage system and a prototype implementation, which will also feature an evaluation of current, state-of-the-art secret sharing implementations in terms of performance, features and applicability to our cloud environment scenario. This will be achieved by implementing a trust function and weight functions to achieve weighted social secret sharing for cloud environments.

Weight assigning to available cloud providers, depending on their trustworthiness, allows us to distribute the shares of a file across multiple cloud providers where by the likelihood of reconstruction of the file by the malicious insider is very low. For this task, we benefit from some recent results on the field of secret-sharing cryptography in terms of social and rational-social secret sharing. An additional important benefit of leveraging secret sharing concepts is the availability of the files, because we do not have to rely on the redundancy of a single cloud provider.

Finally in Chapter 6 the theoretical and practical work in this thesis will be evaluated, reviewed and compared to existing research, specifically Social Secret Sharing. This will be achieved through a proof-of-concept implementation of the distribution strategies and a performance and feature evaluation of Secret Sharing libraries for their practical usefulness in real-world scenarios.

1.3 Related Work

In this section we will introduce several research topics related to this thesis. The papers will be classified in the following categories:

- Searchable Encryption Schemes
- Content and Access Privacy
- Trusted Data Sharing on Untrusted Storage

Searchable Encryption

Searchable Encryption [49] is a promising technique to solve security risks with cloud providers. Searchable Encryption prevents the cloud provider from reading the data but at the same time requires additional work to enable fine-grained, policy-based access control as well as problems related to efficiently querying the data.

The authors of [11] introduce a framework for encrypted search in a XML database. The XML is converted and transferred into a predefined relational table with additional pre, post and parent fields, with which the tree structure is preserved. The framework supports create, read, update and delete methods, but insert operations are potentially expensive because the pre, post and parent columns have to be re-calculated for related nodes, unless the initial numbering left gaps in the sequence for such a case. The querying, which is based on XPath queries of the encrypted XML documents, only supports equality and containment operations. While it would be advantageous to have more operations that are commonplace in relational databases like ranges, greater/smaller than, like etc. it is still a vast improvement to have these two operations as opposed to having none at all. Another tradeoff is the encryption method used. While the encryption time is very good (linear to the size of the input), the ECB encryption that is employed might be a security problem, because it is a very simple encryption mechanism.

The papers [10, 12] introduce the idea of using secret sharing for searching in encrypted data. The framework in [10] uses a mapping table for the tags of the document and assigns a number to each tag. The document is then represented as a tree of polynomials starting at the leaf nodes in the form of (x - tag(x)) where tag(x) is the mapping table function. The non-leaf nodes are calculated as the product of the polynomials of all children of the node and the node's polynomial. After this step the data has to be split in a server and a client part. The client chooses a tree with random polynomials as its client part and then calculates the server part such that the sum of corresponding polynomials is equal to the original polynomial. This is a direct usage of Shamir's secret sharing scheme and can therefore be extended for arbitrarily large number of servers in a (t, n)-scheme, where the client and several servers can reconstruct the shared secret polynomial. This approach is expanded in [12] by using tries[15] to resolve the tags in [10] to individual letters. Therefore the mapping table does not grow with the size of the words in the document any longer as was the case in [10], because the alphabet is limited in size. It should be noted that in both cases it is necessary to keep the mapping table private, because it is the cipher of the encryption mechanism and with it anybody can decrypt the data. The framework supports equality and containment search, but the equality search is more expensive than the containment search, because polynomials closer to the root contain the polynomials farther from the root. Therefore equality requires more search steps until the node is found.

Content And Access Privacy

The authors of [36] propose a framework to solve the problem of content privacy and access privacy when outsourcing data to cloud services. The content is hidden (content privacy) from the server by encrypting it before the upload. Node swapping and redundancy checks then prevent malicious administrators or anybody else from monitoring the encrypted traffic and therefore gaining knowledge about the data by observation. Access redundancy in this context means that the server always sends a bigger data set than was requested by the user, which prevents the server host from knowing what was requested and sent to the user. By frequently swapping nodes the paths of the queries and the data that resulting from it changes frequently and therefore neither similarities in the pattern of requests nor intersections of the redundancy sets are easily trackable. A problem with this approach is that the redundancy set has to be re-encrypted in order to hide the location but with the benefit of having a random distribution of data. That ensures a random distribution of the data, which is very good, but could easily result in a bottleneck in computation power on the server side.

Add narrative regarding another paper on content privacy and access privacy, advantage only encrypt data once A problem in the last paper was the fact that nodes had to be re-encrypted frequently.

The authors of [55] propose a framework to share large amounts of data (in the range of peta-bytes) on the cloud in a owners-write-users-read setup. The goal is to grant fine-grained access control to the data by encrypting every data block with a different cipher and looks at key derivation mechanisms to solve the problem of key management. Another goals of the research is to store each encrypted block only once due to the size of the data, the number of operations should be minimal, because of pay-per-use hosting options, especially re-encryption of data due to access right changes should be avoided and only be needed when updating or inserting data.

It also allows fine-grained access control without having to resort to extensive key management systems due to efficient key-derivation mechanisms. It prevents attackers from eavesdropping and to achieve isolation by using over-encryption of the data before it is sent to a client or can fall back to lazy revocation in case the storage provider does not support over-encryption. Re-encryption is not necessary for revocation of access rights and the used symmetric encryptions schemes weak clients such as mobile phones or tablets can be easily used as well. A disadvantage is that the proposed scheme only supports user-read and limits the write access to (few) owners, making it impractical for collaborative efforts.

Trusted Data Sharing On Untrusted Storage

In [57] a system supporting trusted data sharing using untrusted storage providers like cloud service providers is proposed. It enables users to define access policies and prevents the cloud provider from unauthorized access to the stored data. The idea is to locally encrypt data before uploading it to a storage provider that ideally supports over-encryption and then when another user wants to access a certain file the owner can issue a credential that enables the user to access the file.

The security requirements are defined as follows:

1. data should be confidential, the cloud provider should not be able to compromise the data

2. sharing of data should be possible
3. permissions may not be transferred from one user to another

The secure sharing scheme consists of five steps and are as follows:

1. User A encrypts the data and uploads it to the storage provider
2. User B sends a request to A
3. User A sends a credential to the cloud provider to re-encrypt the data and
4. another credential to B to decrypt the re-encrypted data from the cloud provider using his private key
5. User B acquires the re-encrypted data from the cloud provider and decrypts it.

The re-encryption and the decryption on the server are based on progressive elliptic curve encryption(PECE). The main advantage of this encryption scheme is that a file can be encrypted multiple times but can be decrypted in a single step. It should also be noted that the authors warn that PECE is computationally expensive due to the many multiplication operations that are necessary and that it is less efficient than ECC [28] in that regard. It is also a rather easy protocol in that it only takes five steps to storing data on the cloud, granting access to it, fetching and decrypting it by a third party. Weaknesses are the assumption that we need private keys from the server, although alternatives are hinted at by using Diffie-Hellman key distribution [22], for instance.

Another area of research that is of interest is attribute-based encryption [54, 56] The idea is to define and enforce access policies based on data attributes, while at the same time allowing the data owner to delegate most of the computation tasks to cloud servers without disclosing the underlying data contents.

1.4 Structure of the Work

Chapter 2 gives a detailed view on what cloud computing [5] is and what benefits and challenges [45] accompany it. It also discusses state-of-the-art research into possible solutions to many of the current limitations of cloud computing including the use of multi-clouds [4, 9].

Chapter 3 gives an introduction to secret sharing cryptography, including Shamir's secret sharing scheme and extensions like verified and pro-active secret sharing. Following that we look at social secret sharing, which tries to adopt secret sharing paradigms to cloud environments.

Chapter 4 proposes our models for share distribution in the cloud and Chapter 5 provides a prototype implementation of the new distribution algorithms we have modeled in Chapter 4.

Finally, Chapter 6 provides a critical reflection on the work done in the previous two chapters and highlights strengths and where improvements could be made.

CHAPTER 2

Cloud Computing

2.1 What is Cloud Computing?

Cloud computing comprises technologies and business practices to enable the use IT resources in a flexible, pay-to-go process. Its goal is to provide cheaper, faster and better IT services than regular in-house hosting. This is achieved by making full use of the economics of scale, often through virtualization, by utilizing big data centers around the globe in low-cost environments. It is faster because companies do not have to build their own IT infrastructure before bringing a product to market and it can be better, because the cloud providers can invest more into making their services more agile, providing high performance and scalability, but often at the cost of security. In this chapter we will discuss the benefits and challenges of using cloud services in detail and also give a brief overview of state-of-the-art research topics that aim to provide solutions for issues and challenges in the field of cloud computing.

The Definition

The National Institute of Standards and Technology(NIST) defines Cloud Computing as follows [38]:

> 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.

In [26] Foster et.al. study more than 20 definitions of Cloud Computing in order to get a complete definition of Cloud Computing.

By taking all the aspects of the different definitions into account they arrive at an all-encompassing concept of Cloud Computing:

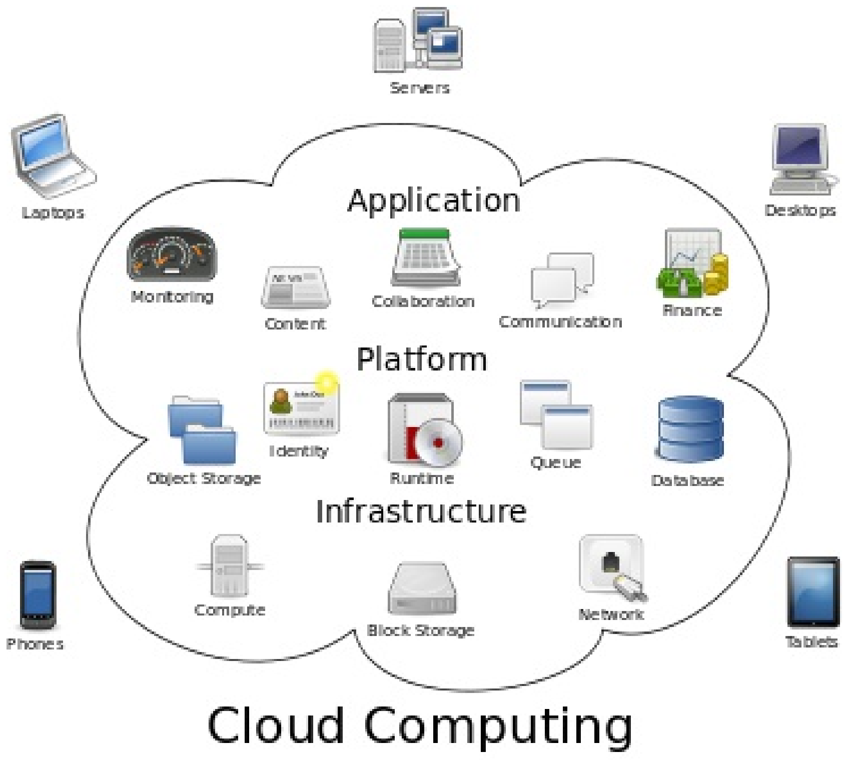


Figure 2.1: Cloud Computing environment. Image retrieved from `http://www.ekaru.com/blog/bid/92650/What-is-Cloud-Computing`

> Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs.

The Essential Characteristics

Cloud computing is defined by the following characteristics:

- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity

- Measured service

On-demand self-service means that a pre-packaged service is available instantaneously, where the user can “unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.” [38] This is in contrast to the old provisioning model, where physical servers have to be ordered and configured before they can be used.

Broad network access defines “capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms.” [38]

Resource pooling is integral to cloud computing, because it enables consumers and vendors to employ “a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand” [38]. Furthermore it allows a “sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources” [38], although the user may specify preferences such as countries or states for data centers in different locations.

Rapid elasticity means that 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.” [38]

A measured service is defined as cloud systems that “automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). [38] The usage of resources can then be monitored and therefore provide transparency for the provider as well as the consumer of the service.

The Service Models

In cloud computing the following three service models are generally recognized:

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

In addition to the three traditional service models, *Anything-as-a-Service (XaaS)* has been considered [35] a fourth service model in recent years.

Software as a Service (SaaS). SaaS applications are designed for end-users, delivered over the web and accessible via a web browser or a program interface. The user does not have to worry about the underlying infrastructure “including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.” [38]

Platform as a Service (PaaS). PaaS expands the benefits that SaaS in the cloud brought to the end-user to include the developers of the software as well. It can be defined as a “computing platform that allows the creation of web applications quickly and easily and without the complexity of buying and maintaining the software and infrastructure underneath it.” [33]

It enables the developer to “deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider”. [38]

Infrastructure as a Service (IaaS). IaaS extends the idea introduced in PaaS 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.” [38] Instead of acquiring hardware, software and the space required to host it space, businesses can instead buy these resources as a service on demand.

Anything as a Service (XaaS). The idea in this approach is to expand the basic service models with making virtually everything and anything available as a service in the cloud.

Aside from the three well-known service models listed above, a number of additional models have been established [35]. These include Strategy-as-a-Service, Collaboration-as-a-Service, Business Process-as-a-Service, Database-as-a-Service, Network-as-a-Service and Communication-as-a-Service.

The last two entries, *Network-as-a-Service* and *Communication-as-a-Service* have been recognized by the International Telecommunication Union (ITU) as part of the cloud computing service models.

The Deployment Models

The following four deployment models are considered an integral part of cloud computing:

- Public cloud
- Private cloud
- Community cloud
- Hybrid cloud

A public cloud is a deployment mode, where 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.” [38]

A private cloud is a deployment model where 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.” [38]

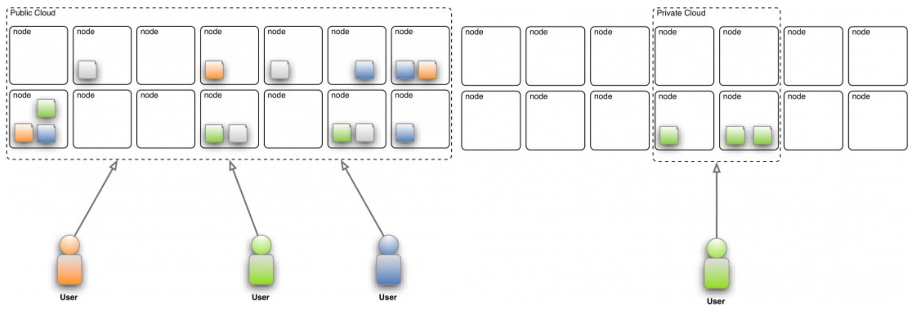


Figure 2.2: A user engaging with a public(left) and a private(right) cloud. Image taken from [17]

Organizations with similar requirements share a cloud infrastructure in a community cloud. The 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)". [38]

A hybrid cloud combines the services of different cloud systems, for example private and public cloud services. The 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)." [38]

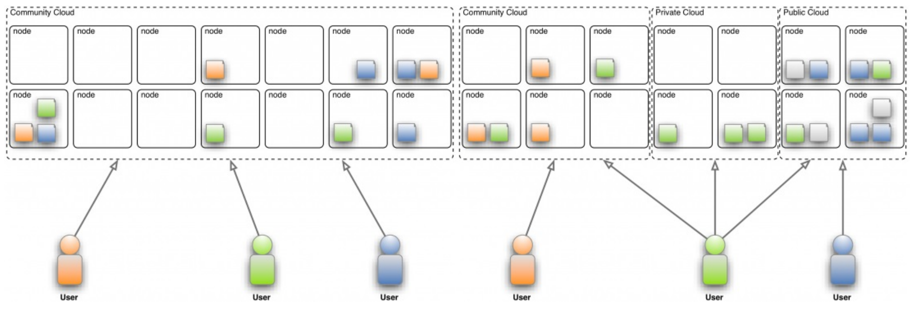


Figure 2.3: A user engaging with a community(left) and a hybrid(right) cloud. Image taken from [17]

2.2 Benefits And Challenges

Overall Benefits

Cloud computing offers many advantages over traditional in-house hosting or services compared to small-to-medium sized providers that enterprises can leverage to improve their businesses.

Armbrust et.al. [5] define the main benefits of Cloud Computing as follows:

1. Infinite computing resources available on demand

2. Lowering costs
3. Ability to pay for use on a short term basis
4. Elimination of up-front commitment by cloud users
5. Improved resiliency and redundancy
6. Economics of Scale
7. Transfer of risk

Infinite computing resources available on demand. The scalability of services is a very important aspect of cloud computing. It can help save setup costs that are often hard to anticipate correctly, because market demand for a new service or product may be unknown or hard to forecast. It is sometimes unavoidable that new IT infrastructure is purchased in higher quantity than is ultimately needed, because businesses do not want to lose future customers by not having enough resources to provide services in case of success.

Lowering costs. With cloud computing businesses do not have to acquire all the potentially needed IT hardware in advance but can scale (downscale as well as upscale) the cloud services as needed later on depending on the success of the product. Additionally, less maintenance personnel and space for hardware is required, which allows a business to save costs as well. On the side of the cloud provider similar cost savings arise through economics of scale that can be given back to the user. The construction of very large-scale data centers at low cost locations allows for a decrease in costs the of electricity, network bandwidth, operations, software and hardware available at these very large economies of scale. These cost savings can be passed on to the customer.

Ability to pay for use on a short term basis. Businesses can become more agile and responsive to changing market situations and business needs. Investing in new markets and products can be an expensive and risky undertaking for a company. Utilizing cloud service providers can help businesses to save money by reducing the necessary capital investment in new lines of business by renting services and hardware instead of having to make costly purchases in advance.

Elimination of up-front commitment by cloud users. Utilizing cloud service providers can help businesses by reducing the necessary capital investment by renting services and hardware instead of having to make costly purchases in advance. Therefore they become more agile and responsive.

Improved resiliency and redundancy. Resiliency is a special form of fail-over where IT resources are spread across several different physical locations - either within the same cloud or across multiple clouds. That way reliability and availability can be improved as well through resilient cloud based solutions.

Economics of Scale. On the side of the cloud provider similar cost savings arise through economics of scale [5] that can be given back to the user. The construction of very large-scale data centers at low cost locations was the key enabler of cloud computing, for they covered the factors of 5 to 7 decrease in cost of electricity, network bandwidth, operations, software and hardware available at these very large economies of scale.

Transfer of risk. Another advantage is a transfer or risk from the business to the cloud provider by the use of Service Level Agreements (SLA) [5]. SLAs define specific parameters and minimum requirements that a service provider or a specific service has to uphold. Parameters that are usually covered in SLAs may include the availability of a service, performance requirements, security considerations, backup and disaster recovery measures, the location of the data, access and portability of data, change management processes , processes to identify and solve problems and a resolution expectation, escalation process and also ultimately a exit strategy.

Overall Challenges

We have identified the following challenges:

1. Migration
2. Vendor Lock-In
3. Interoperability Issues
4. Internet Access
5. Loss of IT Competence
6. Process/Software Changes
7. Data Privacy Laws
8. Security[1]

Migration A big problem for adapting cloud services for businesses is the question of what to migrate.

Based on a survey (Sample size = 244) conducted by IDC [23] in 2008, the seven IT systems/applications being migrated to the cloud are:

IT Management Applications (26.2%), Collaborative Applications (25.4%), Personal Applications (25%), Business Applications (23.4%), Applications Development and Deployment (16.8%), Server Capacity (15.6%), and Storage Capacity (15.5%).

These results show that companies still have reservations about moving their data to the cloud. The rate of adoption for front-end services is higher than that of infrastructure. According to the study, companies are also often outsourcing marginal applications and keep their core line of business applications in-house.

[1]Will be discussed in detail below.

Vendor Lock-In Vendor lock-in is a situation in which a customer using a product or service cannot easily transition to a competitor's product. Therefore a customer may stay with the current service provider even though he is no longer fulfilling the needs of the customer or alternative providers offer better services and/or conditions. Vendor lock-in is usually the result of proprietary technologies that are incompatible with those of competitors.

Interoperability Issues Proprietary cloud APIs make it hard to integrate services with an organization's own existing legacy systems as well as other cloud provider's systems.

Internet Access Transitioning from in-house services to cloud services may drastically change the dependence on Internet access. While many in-house tools will work fine without Internet access or at least provide reduced functionality, cloud services cannot function without Internet access by definition.

Loss of IT Competence Another problem companies may face is a decline in IT competence in-house. It is important to consider this when outsourcing IT infrastructure and personnel whether there could be any side effects to this that are not necessarily related to the immediate issue of moving services to the cloud.

Process/Software Changes Companies need to find out whether processes or other software is affected by a cloud migration.

Data Security Laws Some countries explicitly forbid to move certain data outside the countries' borders. Also, not all data needs the same security assurances. Confidential data may have to be hosted with extra security and the cloud vendor needs to meet the required standards. An example would be health care related documents that require stringent security and access policies and may not be allowed to leave the country of origin. Another potential risk is the loss of data through take-down notices that are very easy to obtain in certain countries through legal frameworks such as the Digital Millennium Copyright Act (DMCA).

Lack of Control Data owners only have a limited control over where their data is stored, how it is processed and what hardware it is stored and run on. Therefore it is important for the owner to "establish a mechanism to mandate the enforcement of their security policies to ensure data confidentiality and integrity." [57] These circumstances have to be considered by the data owner when outsourcing data or business processes to the cloud.

Security

Security is one of the largest concerns for the adoption of Cloud Computing. As shown in Figure 2.4 more than 87% of the respondents cited security as a main concern for adopting cloud computing in the companies and more than 80% of were concerned with the interoperability standards present in current solutions as well as many other factors were also cited in research as major challenges for the adoption of cloud services.

The authors of [5, 14, 18, 20, 21, 23, 26, 37, 45, 46, 51, 52] cite the following issues with cloud security:

1. Privileged user access
2. Malicious Insiders
3. Malicious Outsiders
4. Regulatory compliance
5. Data location
6. Data segregation
7. Recovery
8. Investigative support
9. Long-term viability

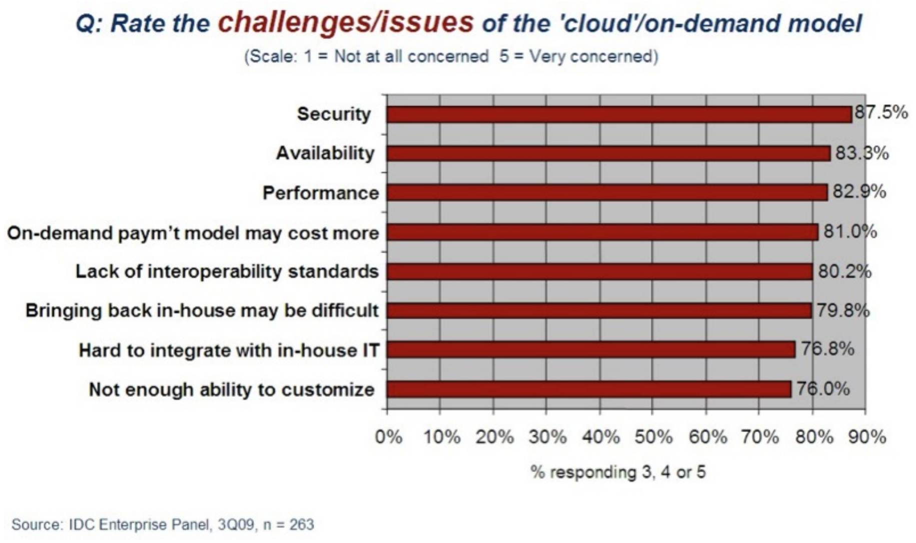


Figure 2.4: Survey on issues and challenges in cloud computing.

Privileged user access. Privileged user access deals with questions about the hiring and management of administrators as well as who has specialized or privileged access to the user's data. For instance, sensitive data processed outside the enterprise should only be accessible and propagated to privileged users.

Malicious Insiders. This problem is closely related to *Privileged user access*. Sometimes the assurances of a cloud provider may not be sufficient for the customer and he or she will have to protect the data from potential malicious insiders at the cloud provider.

Malicious Outsiders. Similarly to malicious insiders, customers may have to think about malicious outsiders as well. This is also related to the next issue that is introduced, *Regulatory Compliance*. Such attacks have to be tracked and the customer may need security measures, like encryption, to ensure the safety of the data.

Regulatory compliance. Is the cloud vendor willing to undergo external audits and/or security certifications? A customer needs to verify if a Cloud provider has external audits and security certifications and if their infrastructure complies with some regulatory security requirements.

Data location. Does the cloud vendor allow for any control over the location of data? Since a customer will not know where her data will be stored, it is important that the Cloud provider commit to storing and processing data in specific jurisdictions and to obey local privacy requirements on behalf of the customer.

Data segregation. The cloud provider needs to ensure that the data of different customers is segregated from one another's. It is important to question whether encryption may be necessary and if that is the case, whether it is available during all stages and that they are state-of-the-art.

Recovery. Recovery plans need to be set in place. It has to be decided what happens to data in the case of a disaster, and whether the vendor offers restoration and how long it takes to restore a service or database. Tt is important that the Cloud provider has an efficient replication and recovery mechanism to restore data in case of disasters or failure.

Investigative support. Another aspect to consider is investigative support. Does the vendor even have the ability to investigate any inappropriate or illegal activity? If this is important, then the customer should ensure to include a contractual commitment to support this.

Long-term viability. When moving services to the cloud the customer needs to think about exit strategies. What would happen if the cloud provider goes out of business or taken over by a competing company. The customer has to ensure that the data is returned and also define the format in which this is happening.

Characteristics of the service models

SaaS. The benefits of SaaS include web access to commercial software, which is managed from a central location, that is delivered in a one to many model, where users are not required to handle software upgrades and patches on the client machines. The use of Application Programming Interfaces (APIs) enable the integration of different software solutions to work together.

Potential security issues with SaaS include focusing on preserving or enhancing the security provided by the legacy software, the lack of knowledge where the provider hosts the data and the resulting discomfort with the lack of control over the data. Multi-tenancy and the fact that companies view their data and business processes as strategic and will want to guard them with access control and access policies. [20, 51]

SaaS is well-suited for situations where there is a requirement for communicating with the outside world. An example would be an email newsletter management and release tool. Another use-case scenario would be software that is only required for short-term use. An example would be a collaboration tool that is only used for a single project, where it would add unnecessary cost to purchase client software and install and maintain it in the business infrastructure. SaaS also performs well in situations where the demand for the software spikes, like tax or billing software, that is not used on a daily basis, but may have increased usage at the end of a month or quarter.

SaaS is not well-suited for use-cases that depend on fast processing of real time data due to the latency of cloud services and the fact that the data may not be suited to be offloaded to the cloud. Another reason that would prevent the use of SaaS is where the legislation or other legal regulations do not permit the data to be hosted externally outside the company's premises or out of country, which may not be guaranteed in a SaaS setting. It should also be evaluated whether by introducing SaaS a real advantage can be leveraged. The adoption may introduce considerable change in the organization and strucuture of precesses. If the SaaS solution does not increase the value or the existing solution fulfills all the requirements of the company then the existing system should not be changed.

PaaS. The benefits of PaaS include the availability of services to develop, test, deploy, host and maintain applications in the same environment. Furthermore, PaaS supports a multi-tenant architecture as well as built-in scalability of deployed software including load balancing and fail-over as well as easy integration of web services and databases via the use of common standards. Many PaaS solutions also support team collaboration by providing project planning and communication tools and provide tools to take care of billing and subscription management.

Potential security issues include vulnerability to host and network intrusions and the fact that often built-in security features are less complete and more flexibility is required to secure the system. Another problem is securing the Enterprise Service Bus(ESB), which would have to be done directly and that it cannot be segmented in a PaaS environment. [51]

PaaS is well-suited to environments where multiple developers work on a single project and external parties need to interact with the development process. PaaS also eases automated testing and deployment services, making it easier to support continuous integration in the development and software release cycle.

PaaS may not be a good choice in situations where the application needs to be portable in terms of where it should be hosted or proprietary languages and approaches would have a negative impact on the development process. Vendor lock-in is another aspect that has to be thought of when considering to move to a PaaS service model. If the use of proprietary tools or languages may prevent later migrations to competing providers. It may also not be the best solution where application performance requires customization of the underlying infrastructure or software, because that is usually not accessible to the user of a platform service.

IaaS. IaaS offers resources distributed as a service and allows for dynamic scaling. It has a variable cost, utility pricing model and generally includes multiple users on a single piece of hardware.

Potential security issues with IaaS include only basic security as well as perimeter firefalls and the need for higher levels of security provided at the host compared to in-house infrastructure hardware. [51]

IaaS is well suited for use cases with volatile resource needs and the applications see spikes in usage that require varying amounts of hardware. It can also be helpful for companies that do not have vast amounts of capital to invest in infrastructure and prefer using operating expenditure in a pay-for-use scenario. This is also the case when the market for an application is growing rapidly and it may not be feasible to purchase the hardware in time to keep up with the demands on the in-house infrastructure. IaaS can usually be deployed faster in such use-cases unless the hardware is set up in advance to be prepared for any kind of market success. IaaS is also an interesting option for trials or temporary infrastructure needs when purchasing hardware is not practical.

On the other hard, IaaS may not be suited in cases where outsourcing data storage and the processing is difficult or when very high levels of performance are required, in which case dedicated in-house hardware may be better suited. Similar to SaaS and PaaS, IaaS should only be considered when the existing capacity of the in-house infrastructure no longer meets the company's requirements or advantages to moving towards IaaS brings can be achieved.

2.3 Interoperability

In section 2.2 several challenges and issues of cloud computing have been highlighted. In recent years, research in the concepts of Multi-Clouds, Inter-Clouds and Federated Clouds have tried to identify potential solutions to problems like *vendor lock-in*, *malicious outsiders* etc.

One of the big concerns regarding cloud computing is the fact that cloud solutions from different providers do not always work well together. In the absence of standardized interfaces to connect the services of different providers, these considerations are often offloaded to the customer.

Differentiation between Multi, Inter and Federated Clouds

There are several aspects that differentiate multi-clouds, inter-clouds, and federated clouds:

- In [43] the authors differentiate between two delivery modes: federated clouds and multi-clouds.
- In [25, 27] the degree of collaborations between the clouds and by the way the user interacts with the clouds is the deciding factor between a federated cloud and multi-clouds:

  - In federated clouds there is an agreement between the different cloud providers to share their resources. The user is aware of the different Clouds and is responsible, or a third party is responsible, to deal with the provisioning of the services or resources.
  - In the second model there is no such agreement. The user interacts with one cloud and is not aware that the resources or services that are consumed are from another cloud.

- An inter-cloud is a federated cloud or a multi-cloud that includes at least one broker and offers dynamic service provisioning. [43]
- According to [27] multi-clouds can be distinguished by being either service or library based.

Multi-Clouds

The idea of multi-clouds is to use two or more cloud service providers to minimize the risk of downtimes or data loss due to failures in hardware, software or infrastructure at the cloud service provider. Multi-clouds are related and sometimes even interchangeable with the terms “Interclouds” or “Cloud-of-clouds”, which have been introduced in [53].

The authors of [43] define ten reasons why multi-clouds are needed:

1. deal with peaks in service and resource requests using external ones, on demand basis
2. optimize costs or improve quality of services
3. react to changes of the offers by the providers
4. follow the constraints, like new locations or laws
5. replicate the applications or services consuming resources or services from different Cloud providers to ensure their high availability
6. avoid the dependence on only one external provider
7. ensure backup-ups to deal with disasters or scheduled inactivity
8. act as intermediary
9. enhance own Cloud resource and service offers, based on agreements with other providers
10. consume different services for their particularities not provided elsewhere

In [3] Zain et. al surveyed the research done on cloud security and found that two thirds of the research was done on single clouds and only one third on multi-cloud systems. The research focuses on data integrity, data intrusion as well as service availability with a focus on storage solutions.

Designing cloud applications in such a way that multiple clouds are used can also be beneficial in preventing *vendor lock-in* and *long-term viability*, two of the challenges we have identified in the previous section on challenges [1, 2, 32, 48] in cloud computing.

By strategically distributing data and processes among multiple cloud providers can also be a security measure to prevent *malicious insiders and outsiders* from accessing or giving them only a limited view the customer's data or when dealing with *Data Security Laws* that require certain data files to be located in specific regions or have to be physically separate from certain other files in the system [9, 31].

Aside from easing security concerns using multi-clouds can also improve the feature set of cloud applications since not all cloud providers offer the same features and have different strengths and weaknesses. For instance, one provider may offer faster response times, while others may have a higher bandwidth or support faster disk operations.

Current research includes multi-cloud databases [4], which uses multiple cloud service providers instead of using a single cloud service provider such as in Amazon cloud service. The authors of this paper focus on the issues related to the data security and privacy aspects in cloud computing, such as data integrity, data intrusion and service availability.

This multi-cloud database permits customers, with different types of databases, queries such as aggregation, exact match and range queries with the ability to store any different types of data such as video, pictures or documents. It preserves security and privacy of user's data by replicating data among several clouds and by using the secret sharing approach.

Federated Clouds

Federated clouds or a cloud federation aims to aggregate the services of multiple providers in a single pool. This pool should support three core interoperability features [34]:

- resource migration
- resource redundancy
- combination of complementary resources

Resource migration allows the relocation of resources, including virtual machine images, data bases, code files, etc. from one service domain to another. Resource redundancy allows the use of similar service features from different cloud providers and the combination of complementary resources allows combining different types to aggregated services.

In [16] the authors propose a *three-phase cross-cloud federation model*, where the establishment of a cloud federation passes through three phases: discovery, match-making and authentication.

A cloud, that needs external resources, first looks for other available clouds (discovery), after which it chooses the one that best fits its needs (match-making), and finally establishes a trust relationship (authentication) with the cloud it chose in step 2 to provide the resources that it requires.

Inter-Clouds

Inter-clouds are either federated or multi-clouds that include at least one broker and offers dynamic service provisioning according to [43].

Current research [7,13,15] focuses on the fact that “cloud proliferation has not lived up to expectations in the enterprise segment. Often cited issues include confidentiality and integrity, but also reliability and consistency.” and sees the inter-cloud as the “second layer in the dependable cloud computing stack for the next-generation cloud”. [15]

2.4 Summary

In this chapter we have introduced the key characteristics of cloud computing as well as the service models and deployment models. Following that we have highlighted the main benefits and also the key challenges with a focus on security, availability, vendor lock-in and lack of control.

Current research topics in cloud computing including multi-clouds, inter-clouds and federated clouds highlight potential solutions for many of the challenges in cloud computing and may enable many use-cases for personal and business computing. Some of these solutions rely on secret sharing cryptography, which will be covered in the next chapter in detail and be the basis for the future work in this thesis.

CHAPTER 3

Secret Sharing Cryptography

3.1 Introduction

The idea of Secret Sharing is to distribute a secret among several participants, where each of them receives a part or share of the secret. Therefore it is well-suited for secure key management systems. In case of storing important data in an untrusted location key-based encryption is often employed. The problem with this kind of encryption is how to store the key securely, because if it is lost then the data cannot be retrieved any longer. Secret sharing allows to distribute the key in multiple parts, so that the secret can be retrieved using a subset of the shares.

Secret Sharing was first introduced independently by Blakley [8] and Shamir [47] in 1979. Shamir's version of Secret Sharing is more space-efficient with the shares being only as large as the original secret whereas in Blakley's version the shares are t times bigger than the original secret, where t is the threshold number of players in the scheme.

Figure 3.1 visualizes Blakley's secret sharing scheme, where 3 non-parallel hyperplanes intersect to reveal the secret.

Properties and Use-Cases

We consider two attributes that are suitable for using Secret Sharing in the cloud and in general:

- Achieve Fine-Grained Access Policy
- Availability

Fine-grained access policies can be set up by using a secret sharing threshold scheme. Let us assume an use-case scenario in a bank, where access to the vault is limited to several employees.

It may be undesirable for a single person to have access to the vault. For instance, the bank manager may have a key to open the vault with one other subordinate present. In his or her absence any three subordinates could be authorized to open the vault together. To prevent a single subordinate to open the vault this would require at least two separate locks, one for the manager

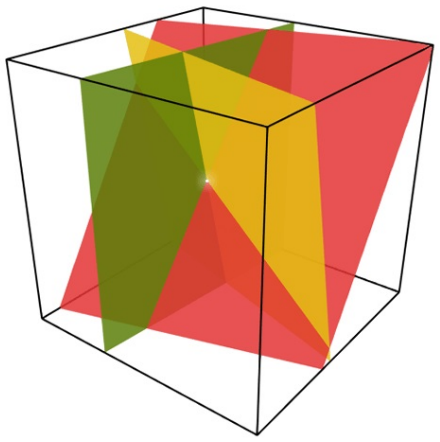


Figure 3.1: Blakley's scheme using three dimensions: each share is a plane, and the secret is the point at which three shares intersect. Image taken from Wikipedia.

and subordinate and one for the three authorized subordinates without the presence of the manager. In case of any three out of t subordinates the number of combinations would increase to 3^t. It is easy to see that this scenario could be extended easily to make it impractical to use if there are more emergency protocols or more persons authorized under different circumstances.

Secret sharing offers a simple solution to that problem by introducing a threshold secret sharing scheme. In this scheme, often written as a $(t, n) - thresholdscheme$ any t shares out of n total shares can be used to decrypt the secret, where t is larger than n.

In Shamir's secret sharing theme it is guaranteed that with less than n shares the secret can not be inferred, because every possible solution is as likely as any other. The details of this scheme and a proof as well as other implementations of Secret sharing that may not have these attributes will be given in the next section.

In the bank vault scenario the manager could therefore be given t shares in order to unlock the vault by his- or herself. Any subordinate could be given $t/2$ shares in order to unlock the secret access code to the vault together with the shares of any other subordinate in the manager's place.

The secret sharing scheme not only gives access control, but as can be seen in the above example also increases the availability. It does not matter which subordinate is present, any three will do to unlock the secret. In the sense of resource availability in computing the secret's availability is increased, because only a subset is required to retrieve the secret.

3.2 Shamir's Secret Sharing

Shamir's Secret Sharing is a form of secret sharing based on polynomial interpolation. The goal is to divide some data D into n pieces $D_1, ..., D_n$ in such a way that:

- knowledge of any t or more D_i pieces makes D easily computable

- knowledge of any $t-1$ or fewer D_i pieces leaves D completely undetermined in the sense that all its possible values are equally likely.

A scheme following these rules is called a (t, n)-threshold scheme. Figure 3.2 visualizes how one can draw infinitely many polynomials of degree 2 through 2 points, while 3 points are required to define a unique polynomial of degree 2. It should be noted that Shamir's version of secret sharing works in finite fields, which are not representable on a 2-dimensional plane.

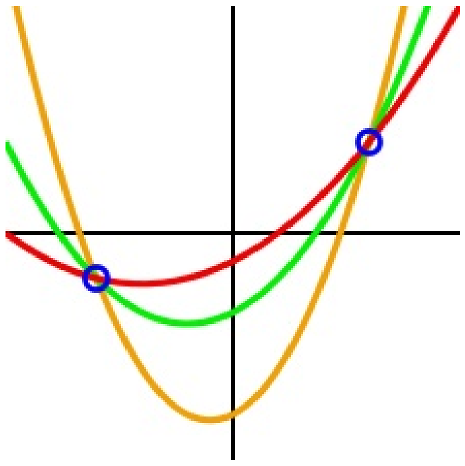


Figure 3.2: Shamir's secret sharing scheme represented in a 2-dimensional plane. Image taken from Wikipedia.

Construction by example

The construction of Shamir's Secret Sharing scheme will be explained by way of an example: Let the secret be $S = 10$ with parameters $n = 5$ and $k = 3$.

Since the threshold is set to $k = 3$, we need a polynomial of degree $k - 1$ and define it as $f(x) = S + a_1x + a_2x^2$ and set $a_1 = 7$ and $a_2 = 5$ at random in $Z/_{11Z}$. The polynomial therefore has the form $f(x) = 10 + 7x + 5x^2$.

The trusted party (also called the dealer in the Secret Sharing scheme) can now generate as many shares as are necessary:

$$\begin{aligned}(1, f(1)) &= (1, (10 + 7 \cdot 1 + 5 \cdot 1^2) \bmod 11) = (1, 22 \bmod 11) = (1, 0)\\(2, f(2)) &= (1, (10 + 7 \cdot 2 + 5 \cdot 2^2) \bmod 11) = (2, 44 \bmod 11) = (2, 0)\\(3, f(3)) &= (1, (10 + 7 \cdot 3 + 5 \cdot 3^2) \bmod 11) = (3, 76 \bmod 11) = (3, 10)\\(4, f(4)) &= (1, (10 + 7 \cdot 4 + 5 \cdot 4^2) \bmod 11) = (4, 138 \bmod 11) = (4, 6)\\(5, f(5)) &= (1, (10 + 7 \cdot 5 + 5 \cdot 5^2) \bmod 11) = (5, 170 \bmod 11) = (5, 5)\end{aligned}$$

After the shares are distributed among the participants, the secret polynomial can be discarded.

In order to recover the secret we need any 3 out of 5 shares in order to compute the secret S using interpolation and the following formula:

$$S = \sum_{i \in A} (Ii. \prod_{j \in A \setminus \{i\}} \cdot \frac{j}{j-i})$$

We choose the shares (2, 0), (3, 10) and (5, 5) arbitrarily to recover the secret:

$$f(0) = (0 \cdot \frac{3}{3-2} \cdot \frac{5}{5-2}) + (10 \cdot \frac{2}{2-3} \cdot \frac{5}{5-3}) + (5 \cdot \frac{2}{2-5} \cdot \frac{3}{3-5}) = -1 \bmod 11 = 10 \bmod 11.$$

As we can see the reconstructed secret S = 10 is correct.

3.3 Adversary models

Adversary models are important in the design of a secure system. It explains how the system is handling threats, what assumptions have been made, whether the model is unconditionally secure or only computationally. It also defines the intentions of the adversary and sets the scope of their potential actions.

Stinson et.al. categorize the following adversary models applicable to Secret Sharing schemes in Cloud Computing [40,41]:

- passive and active
- static and mobile
- computational and unconditionally secure

The passive adversary model assumes that the participants are curios and want to find out what the secret is. They will, however, not derivate from the given protocol in order to do so. This model is also called the “honest but curios” model. In the active adversary mode on the other hand, the participants may deviate from the protocol if this will enable them to learn the secret.

Furthermore, active and passive adversaries can be extended to include a static or mobile setting. The former setting implies that the adversary is able to corrupt other participants in advance, while in the latter setting the participants can only be corrupted during the different stages of the protocol.

As has already been indicated in the introduction, the security model can be unconditionally secure or computational. The former indicates that the adversary has unlimited computation power, while in the latter the security of the protocol also relies on assumptions regarding the hardness of factoring or discrete logarithms.

3.4 Verifiable Secret Sharing

Verifiable secret sharing schemes(VSS) [19] are designed to verify the consistency of shares during both the sharing and recovery phases. The first unconditionally secure versions of VSS were introduced in [6]. This scheme assumes the existence of secure private channels and is unconditionally secure only when $t \leq n/3$. The authors in [44] follow a similar path by assuming the existence of a broadcast channel on top of the secure private channels and only requires $t \leq n/2$. The proposed scheme, however, has a negligible probability of error, while the former scheme has zero probability of error.

Additional VSS schemes were introduced by Feldman [24] and Pedersen [42]. The schemes use homomorphic functions that are hard to invert and also make use of hardness of computing discrete logarithms over Z_p for prime p.

VSS, contrary to Shamir's secret sharing, assumes that the dealer cannot be trusted and could, in principle, transmit corrupted shares with which the participants would not be able to recover the secret. This also means that VSS is not only important against adversaries, but can also be critical in detecting errors during transmissions of shares or corruption of files without the presence of an adversary.

VSS also assumes that only authorized groups of participants are able to learn the secret and that honest participants may want to learn the secret even if the adversary corrupts the dealer and/or shares.

Furthermore, VSS schemes can be separated in the following categories:

- Interactive VSS: Involves the participants exchanging messages to determining the correctness of the shares.
- Non-interactive VSS: In this type of VSS scheme the share proves its own validity and an exchange of messages between participants is not necessary.
- Publicly VSS: The goal of PVSS is that anybody can verify that the participants received correct shares and not only the participants themselves.

In the following section the VSS protocol of Feldman's [24] scheme will be explained.

VSS scheme by Feldman

The version introduced by Feldman is a non-interactive VSS scheme. The protocol consists of the following stages:

Sharing Phase

Each participant P_i gets a different non-zero element $\alpha_i \in Z_p$, where Z_p is a finite field and g is a primitive element in Z_p. Furthermore, let S be the secret in Z_p.

The sharing phase consists of three steps that are described as follows [39]:

1. The dealer D chooses a random polynomial $f(x)$ of degree k over the finite field Z_p, where $f(0) = s$.

2. Each share s_i is computed by the dealer D as $s_i = f(\alpha_i)$ and then transmitted in secret to the participant P_i.

3. The dealer D broadcasts the values $A_j = g^{a_j}$ for $j = 0, 1, \ldots, k$ with $f(x) = \sum_{j=0}^{k} a_j x^j$.

Detection Phase

The detection phase consists of two steps that are described as follows [39]:

1. Each participant P_i verifies the validity of their shares by checking the following equation: $g^{s_i} = \prod_{j=0}^{k} A_j^{\alpha_i^j}$. If the equation holds, then the share is valid, otherwise P_i broadcasts an accusation to the dealer.

2. If more than k accusations to the dealer are broadcast, then the dealer D is considered corrupted and the protocol is stopped.

Reconstruction Phase

The reconstruction phase consists of three steps that are described as follows [39]:

1. Each participant P_i broadcasts $f(\alpha_i)$.

2. Take $k + 1$ broadcast values for which $g^{f(\alpha_i)} = \prod_{j=0}^{k} A_j^{\alpha_i^j}$ holds.

3. Determine $\tilde{f}(x)$ of degree at most k that passes through these points and output $\tilde{f}(0)$.

3.5 Proactive Secret Sharing

Proactive secret sharing (PSS) was introduced in [29]. The problem with threshold secret sharing is that the threshold may not be reached by an adversary through the entire lifetime of the secret. Security is only guaranteed if the adversary cannot recover more than k shares in up to n locations indefinitely. It would be possible for the adversary then to gradually break into different locations over a long period of time. One solution to this problem is to change the secret regularly and generate new shares, but this may not always be possible. For instance, cryptographic keys may be hard to change, because then all the information encrypted with those keys would have to be re-encrypted as well. Another case would be legal documents or large amounts of data that does not naturally change very often. That data is prone to such attacks as described above.

The idea of PSS is that the shares of participants can be updated without changing the secret in such a way, that once the update is finished, only new shares can be used to recover the secret, but having $k + 1$ or more shares split between old and new shares won't reveal the secret. If this protocol is enacted in a certain time interval t, then an adversary will have to gather $k + 1$

shares in less time than $k + 1$, otherwise his or her shares will be useless to recover the secret in the future. Another requirement is to prevent the loss of data by corruption of shares. In order to achieve this we need to periodically recover lost or corrupted shares without corrupting the original secret.

Periodic Share Renewal Scheme

The scheme presented in [29] not only guarantees the secrecy of the shared secret, but also the integrity and availability of the secret in the presence of up to k misbehaving servers.

Assumptions

Private and authenticated communication in distributed environments usually relies on public key cryptosystems. This poses a problem when an adversary breaks into one of the servers and is able to retrieve the public or private key to this server, changing them to prevent further communication and it to impersonate that server in the future. The scheme will assume that the private key cannot be learned or modified by an adversary and that the server's view of the public keys of other servers cannot be modified, either. Therefore the adversary will only be able to decrypt messages using the private key in the sense of a black box and generate signatures.

Initialization

The scheme assumes an initial stage, where a (k, n)-threshold Shamir's secret sharing scheme is started with a secret $S \in Z_q$ with q being a prime, is split into n pieces $s_1, \ldots, s_n \in Z_q$. Every participant $P_i \in \{1 \ldots n\}$ holds its share s_i for some polynomial f($\cdot$). At the start of each time period after initialization, the servers initiate an update phase in which a share renewal is done. Shares computed in time period t are denoted as $x_i^{(t)}$ and the polynomial $f^{(t)}(\cdot)$.

Share Renewal

Passive Attacker A share is updated by adding a random k-degree polynomial $\delta(\cdot)$, where $\delta(0) = 0$. Then $f^{(t)}(0) = f^{(t-1)}(0) + \delta(0) = x + 0 = x$. The renewal protocol for each server can be described in three steps:

1. P_i picks k random numbers from Z_q and defines a polynomial δ_i with a free coefficient $c = 0$, so that $\delta_i(0) = 0$.

2. For all servers P_j, where $j \neq i$, P_i sends $u_{ij} = \delta_i(j)(\mathrm{mod} q)$ to P_j.

3. After decrypting all u_{ij}, P_i computes its new share $x_i^{(t)}$ as $x_i^{(t-1)} +)u_{1i}, \ldots, u_{1n})(\bmod q)$ and erases all the data except the new share $x_i^{(t)}$.

It is important to remember that during step 2 it is assumed that the keys are transfered with perfect security. The problem with this protocol is that an active adversary controlling a server can cause the destruction of the secret by choosing a polynomial δ_i with $\delta_i(0) \neq 0$. It is therefore necessary to incorporate verified secret sharing in the protocol to ensure the integrity of the shares in case of an active adversary.

Active Attacker In case of an active attacker, the model has to be extended to incorporate VSS.

1. P_i picks k random numbers δ^{im}, where m $\in 1 \dots k$, to define the polynomial g_i and generates k values $\epsilon_{im} = g^{\delta^{im}}(\text{mod} q)$.
2. P_i computes $u_{ij} = \delta_i(j)(\text{mod} q)$ with $j \in \{1 \dots n\}$ and $e_{ij} = ENC_j(u_i j), \forall j \neq i$.
3. P_i sends the message $VSS_i^{(t)}$ and the signature $SIG_i(VSS_t^{(t)})$.
4. For all received messages in step 3 from the other servers, P_i decrypts and verifies its shares by using the following formula:

$$g^{u_{ji}} = (\epsilon_{j1}^{i})(\epsilon_{j2}^{i^2}) \cdots (\epsilon_{jk}^{i^k})(\text{mod} p)$$

5. If P_i detects no anomalies, then it broadcasts a message to the other participants informing them that all checks were successful.
6. If all the other servers acknowledged the message P_i proceeds to update its own share and erases all the data except the new share.
7. If anomalies in the behavior of other servers are found in step 5 during step 4 it then sends an accusation message to the other servers.

Resolving Accusations In order to correctly detect misbehaving servers it is important to verify the accusation to prevent a corrupted server from accusing other servers of being corrupted themselves. First, the server has to reject the polynomial $\delta_i(\cdot)$ sent by the corrupted server and second, the server has to inform all the other servers of the corrupted server. In order to verify that the accused server is indeed corrupted, all the other servers have to agree on which of the two servers is misbehaving. In order to establish this, the servers verify that the messages are free any irregularities, which includes:

1. whether a message is formally incorrect, that is sending messages in with the wrong time period, or numbers out of bounds etc.
2. two or more correct but distinct messages from the same server or no message at all
3. a mismatch in the verification of the message in step 4

The first two irregularities can be found using public information and mark the accused server as bad, that is it is removed from the list of trusted servers A and put in the list B. The third irregularity can only be computed locally by the receiving server. Therefore the accused server has to defend itself by sending a correct $u_i j$. It can then prove that this value corresponds to the publicly available encryption value e_ji, which was broadcast in step 3. All the other servers can now check whether this information is correct. Once all the accusations are resolved, the computation of new shares can resume by replacing step 6 in the renewal protocol by:

$$x_i^{(t)} \leftarrow x_i^{(t-1)} + \sum_{j \ni B_i} u_{ij} (\bmod q)$$

Detection of Corrupted Shares

The system has to be able to deal with missing or corrupted shares. This can be the case when a server missed or misbehaved during an update phase.

In order to verify the shares in a distributed environment, an invariant is added in each time period t and each server saves a set of exponents $y_j^{(t)} = g^{x_j^{(t)}} (\bmod p)$ of the current shares of all the servers in A.

Basic Share Recovery Protocol

The idea of the recovery protocol for passive adversaries is to take a group $D \subset A$ of d shares with $(k+1 \leq d \leq n-1)$ and think of it as a $k+1, d$-threshold scheme of any of the remaining shares $x_r, r \notin D = A \setminus B$.

1. Each$P_i \in D$ picks a random polynomial δ_i of degree k such that $\delta_i(r) = 0$.
2. And broadcasts $ENC_j(\delta_i(j))$ with $j \in D$.
3. Then P_i creates a new share of x_r with $x_i' = x_i + \sum_{j \in D} \delta_j(i)$ and sends it to P_r by broadcasting $ENC_r(x_i')$.
4. Finally, P_r can decrypt the shares and recover x_r.

Full Share Recovery Protocol

In order to extend the basic protocol to add protection against a mobile adversary, we have to add verifiability to steps 2 for dealing with $\delta_i(\cdot$ and for reconstruction in steps 3 and 4.

3.6 Social Secret Sharing

The papers [40, 41, 50] introduce "the notion of social secret sharing, in which shares are allocated based on a player's reputation and the way he/she interacts with other participants".

The motivation behind a social secret sharing theme is that, "in real-world applications, components of a secure scheme may have different levels of importance [...] as well as reputation".

Social secret sharing consists of three stages: sharing, tuning and recovery. The first and third stages are identical to Shamir's secret sharing, the only difference is the tuning phase, in which the participant's weight is adjusted according to his or her reputation. [40]

Assumptions

In order to construct a social secret sharing theme there are several steps that need to be taken. The weight of the participants $P_i \in \triangle$ has to be greater than the threshold:

$$\sum_{P_i \in \triangle} w_i \geq t$$

Furthermore, the weight of the colluders $P_i \in \bigtriangledown$ has to be less than the threshold:

$$\sum_{P_i \in \bigtriangledown} w_i \leq t$$

Finally, the weight of each participant, which is denounced by w_i should be limited by a parameter m, which is much smaller than the threshold t:

$$w_i \leq m \ll t$$

Social Tuning

The social tuning phase consists of three steps, which readjust the weight of the participants.

Adjustment: On the basis of the player's availability or response time, the reputation and therefore the weights of the participants are adjusted.

Enrollment: Participants should be allowed to join during the protocol, or when their weight was adjusted, they should be allowed to receive more shares. In order to do this, the other participants have to be able to generate new shares for this participant without the involvement of a dealer.

Disenrollment: Similar to the enrollment, a participant's weight may be decreased. Therefore the protocol needs a way to disable existing shares of a player, if their reputation or weight no longer represents their number of shares accordingly. A new polynomial $\tilde{f}$ is used for those shares that are still valid, such that participants can no longer use shares generated with the old polynomial f, that are therefore invalidated.

Trust Function

Let $\mathcal{T}_i^j(p)$ be the trust value assigned to P_i by P_j and $\mathcal{T}_i$ be the social trust function representing the reputation P_i.

$$\mathcal{T}_i(p) = \frac{1}{n-1} \sum_{j \neq i} \mathcal{T}_i^j(p)$$

Example. Let the trust values of P_1, P_2, P_3 with respect to P_4 be $\mathcal{T}_4^1(p) = 0.4$, $\mathcal{T}_4^2(p) = 0.5$ and $\mathcal{T}_4^3(p) = 0.6$ respectively. Hence the reputation of P_4 will be $\mathcal{T}_4(p) = 0.5$, because only a public trust value is assigned to each player and $\mathcal{T}_i(p) = \mathcal{T}_i^j(p)$ for all j.

As noted in Table 3.1, the protocol considers three kinds of participants with six defined outcomes, where α and β determine the boundaries on the trust values for the different sets of participants.

Trust value	Cooperation: $P_i(C)$	Defection:$P_i(D)$
$P \in B \Rightarrow T_i(p) \in [-1, \beta)$	Encourage	Penalize
$P \in N \Rightarrow T_i(p) \in [\beta, \alpha)$	Give A Chance	Take A Chance
$P \in G \Rightarrow T_i(p) \in [\alpha, 1)$	Reward	Discourage

Table 3.1: Six Possible Actions For The Trust Management

In order to update the reputation of the participants they use the functions $\mu(x)$ and $\mu'(x)$ as can be seen in Figure 3.3. To increment or decrement the trust value of a participant the parameters η, θ and κ are used and in the intervals $[1-\varepsilon, 1]$ and $[-1, \varepsilon - 1]$ both functions $\mu(x)$ and $\mu'(x)$ converge to 0 due to the assumption in the definition of the trust function.

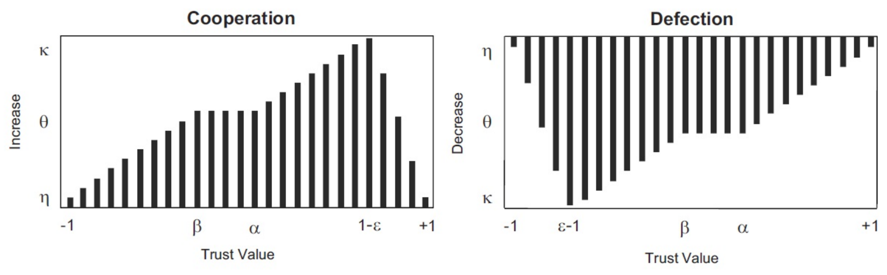


Figure 3.3: Trust adjustments by applying $\mu(x)$ and $\mu'(x)$. Figure taken from [40].

It is important to note that these functions do not only reflect a single round trip but also the history of the actions of the participants. It rewards good and penalizes bad participants more than average as can be seen in Figure 3.3 depending on the action from Table 3.1 and the described intervals: Cooperation is rewarded most in the interval $[\alpha, 1-\varepsilon]$ and Defection is penalized most in the interval $[\varepsilon - 1, \beta]$ and remains neutral for new participants that just entered the protocol in the interval $[\beta, \alpha]$.

The functions $\mu(x)$ and $\mu'(x)$ are defined as follows with $l_i \in \{0, 1\}$, where $l_i = 0$ denotes defection and $l_i = 1$ cooperation in time period i and $x = \mathcal{T}_i(p-1)$ being the previous trust value:

$$\mu(x) = \begin{cases} \frac{\theta-\eta}{\beta+1}(x+1) + \eta & P_i \in B \\ \theta & P_i \in N \\ \frac{\kappa-\theta}{1-\varepsilon-\alpha}(x-\alpha) + \theta & P_i \in G \\ \frac{\kappa}{\varepsilon}(1 - x - \varepsilon) + \kappa & \mathcal{T}_i(p) > 1 - \varepsilon \end{cases}$$

$$\mu'(x) = \begin{cases} \frac{\kappa}{\varepsilon}(x+1 & \mathcal{T}_i(p) < \varepsilon - 1 \\ \frac{\theta-\kappa}{\beta-\varepsilon+1}(x-\varepsilon+1)+\kappa & P_i \in B \\ \theta & P_i \in N \\ \frac{\eta-\theta}{1-\alpha}(x-\alpha)+\theta & P_i \in G \end{cases}$$

To ensure that $\mathcal{T}_i(p)$ never exceeds $+1$ or -1 the conditions $1-\varepsilon+\kappa \leq 1$ and $\varepsilon-1-\kappa \geq -1$ have to hold.

Application in Cloud Computing

The protocol including the trust function can be used in cloud computing to:

1. validate SLA with cloud service providers
2. achieve a higher customer satisfaction.

The proposed model for cloud computing consists of a deal, who initialtes a “weighted secret sharing scheme” with n cloud providers. Let $\overrightarrow{r} = (r_1, r_2, ..., r_n)$ and $\overrightarrow{w} = (w_1, w_2, ..., w_n)$ be the trust value and weight vectors of the participants and define the following actions $A_i \in \{\mathcal{C}, \mathcal{D}, \mathcal{X}\}$:

1. $\mathcal{C}$: for cooperative participants that are available at the required time and sends correct shares to the other participants
2. $\mathcal{D}$: for uncooperative participants that do not respond or respond with delay
3. $\mathcal{X}$: for corrupt participants that have been compromised and may send corrupted shares

Example. Let us assume a secret key ξ in an auction system. The system initialization is handled by the dealer, who distributed the initial shares according to the weights $\overrightarrow{w}$ of the different cloud providers, which can be seen in Figure 3.4. After this the dealer leaves the scheme.

Following the initial distribution the different cloud providers perform the actions that are required of them. The providers will answer requests from the servers of the auctioneers and eventually the secret will be recovered and the sealed-bid auction is accomplished.

During this process the different cloud providers rate each other based on their actions $A_i \in \{\mathcal{C}, \mathcal{D}\}$ and the trust function. Following that the weight of each cloud provider is adjusted as can be seen in Figure 3.5. In case of corruption, that is $A_i = \mathcal{X}$, the cloud provider is reset and rebooted and may join the protocol as a newcomer in the next round. The corruption can be detected by using a verified secret sharing scheme as we have introduced in Section 3.4.

In this example, w_1 has not performed as reliably as w_4. Subsequently, the weight of w_1 is adjusted downward accordingly and the weight of w_4 is incremented.

In the last phase, the tuning or self-configuration phase, the shares are redistributed by the cloud providers according to their weights in a pro-active secret sharing scheme, as was introduced in Section 3.5.

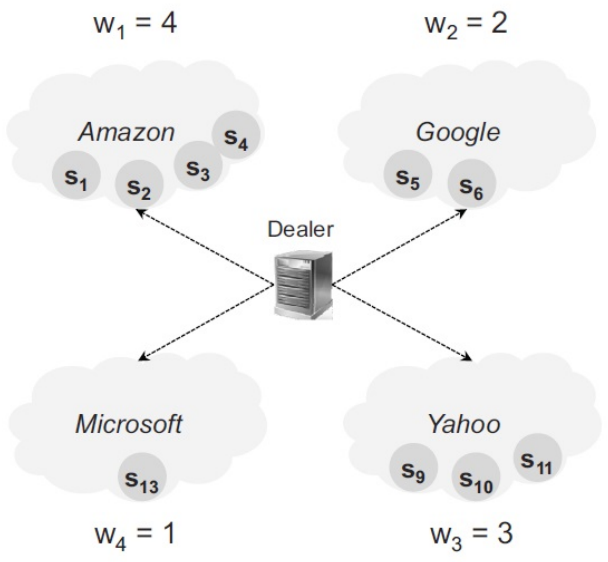


Figure 3.4: System Initialization. Figure taken from [40].

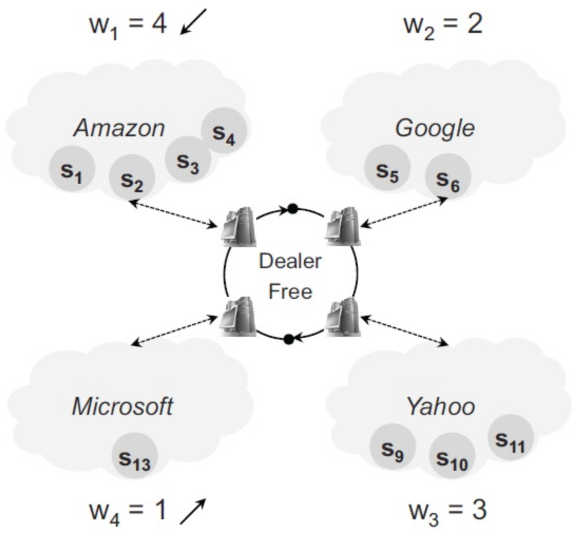


Figure 3.5: Weight Adjustment. Figure taken from [40].

The benefits of using this version of Shamir's secret sharing protocol is a fault-tolerant system that provides better availability, which is adjusted in time to consider changes in reliability in different cloud providers.

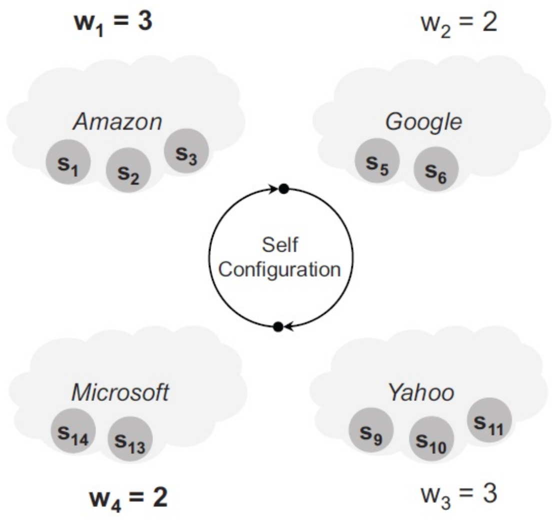


Figure 3.6: Self-Configuration. Figure taken from [40].

Adapted Social Trust Function For The Cloud

For the protocol that was introduced in Section 3.6 the trust function that was introduced in Section 3.6 is enhanced:

The modified trust function, called a "social trust function", retains $\mu(x)$ and $\mu'(x)$ that were previously introduced and is defined as follows:

$$\mathcal{T}_i(p) = \begin{cases} \mathcal{T}_i(p-1) + (1 - \frac{\delta}{n})\mu(x) & \text{if} l_i = 1 \\ \mathcal{T}_i(p-1) - (\frac{\delta}{n})\mu'(x) & \text{if } l_i = 0 \end{cases}$$

where $\delta = \sum_{i=1}^{n} l_i$.

An updated example of the actions can be seen in Table 3.2. Participants gain a part of their reward that is proportional to the number of non-cooperative participants, and vice-versa, their loss is also proportional to the number of loyal participants in case they do not respond in time. This means that the trust function is capable of rewarding those participants more when the rate non-cooperation is high, which is quite reasonable. This also means that the trust values should remain the same if all cloud providers cooperate or defect at the same time, no matter their history.

3.7 Unencrypted Secret Sharing Techniques

Aside from encrypted secret sharing schemes such as Shamir's secret sharing there are also versions that do not rely on encryption. These schemes can be used to split a secret in small parts and distribute the pieces among the participants such that a single defecting cloud provider cannot retrieve any valuable information from the single piece or pieces that he or she possesses.

$\delta = \sum_{i=1}^{n} l_i$	Cooperation	Defection
n	$\mathcal{T}_i(p-1)$	no defection
$\frac{3}{4}n$	$\mathcal{T}_i(p-1) + 0.25\mu(x)$	$\mathcal{T}_i(p-1) - 0.75\mu'(x)$
$\frac{1}{2}n$	$\mathcal{T}_i(p-1) + 0.5\mu(x)$	$\mathcal{T}_i(p-1) - 0.5\mu'(x)$
$\frac{1}{4}n$	$\mathcal{T}_i(p-1) + 0.75\mu(x)$	$\mathcal{T}_i(p-1) - 0.25\mu'(x)$
0	no cooperation	$\mathcal{T}_i(p-1)$

Table 3.2: Adapted trust management with the new trust function.

Example of a password being split. Let the secret s = “password”
order of the shares has to be known
Share 1: “pa——”
Share 2: “–ss—-”
Share 3: “—-wo–”
Share 4: “——rd”

While such a scheme is not as secure as Shamir’s Secret Sharing scheme, where no knowledge is leaked until the threshold is reached, use cases fitting this property can be argued for nevertheless. The authors of [30] propose such a scheme, where for example, an image is split into many small pieces, which could optionally be transformed prior to being distributed among the different storage locations to enhance security.

This scheme would have the advantage over Shamir’s secret sharing scheme in storage space overhead, because the pieces would not have to be as big as the original file, but of course, the security of the individual pieces is not as good as with Shamir’s secret sharing scheme.

3.8 Summary

In this chapter we have given an overview of some of the existing cryptographic secret sharing schemes, including how the basic secret sharing cryptography developed by Adi Shamir [47] in 1979 works, how it can be extended to support other desired features. These features include verifiability [24] via verified secret sharing and pro-active secret sharing [29], which can be used to update shares without changing the secret itself and therefore further increase the security by invalidating old shares that cannot be used alongside the newly generated shares to recover the secret.

We have also given an outlook of how these schemes can be used for cloud computing by using the social secret sharing scheme and the associated trust functions presented in [41].

In Chapter 4 we will present several models that are based on the concept of social secret sharing, which use a distribution mechanism to support different use-cases than the one presented here.

CHAPTER 4

Cloud Storage Models

4.1 Motivation

Today users have a myriad of Cloud Storage services to chose from like Dropbox [1], Google Drive [2], Microsoft SkyDrive [3] or Apple iCloud [4] to store their personal data in the cloud.

These services are often integrated in the operating system in the desktop and notebook realm as well as in the mobile space for smartphones or tablets. The services are easy to use, provide a good user experience and often offer a free tier that is adequate for storing personal documents and media files free of charge with the ability to upgrade to paid tiers for users who need more storage space.

None of the mentioned services provide adequate security and privacy. Data is transmitted unencrypted to these services and can be read by the cloud provider or anyone in between. With many of the mentioned providers it is even necessary for the user to agree to the data being used by the cloud provider to tailor specific advertisements according to the data that is stored and the privacy settings remain in question.

We provide a framework that can use multiple cloud storage providers and do not provide hosting in any way. Therefore the system can be seen as an additional software layer on top of existing cloud storage solutions that makes it easy to configure and use these services in a secure way even for non-expert users.

Security is guaranteed by using Shamir's Secret Sharing to encrypt all files that are transferred to the cloud and the ability to define custom rules on where and how files are to be stored. For example, a user may not want to upload files on servers in a specific country or region and in the case of Secret Sharing, only a certain number of the shares should be stored within a single storage provider or multiple storage providers within the same country or region/jurisdiction.

[1] `https://www.dropbox.com/`
[2] `https://drive.google.com/`
[3] `https://skydrive.live.com/`
[4] `https://www.icloud.com/`

Security is not the only aspect of configurable rules in the system. Availability is also a very important aspect that has to be kept in mind when designing outsourcing systems. A cloud provider may not be reachable in a critical time or performance may vary. It is therefore also important to consider failsafe scenarios for such cases. Since Secret Sharing requires multiple independent cloud providers in order to ensure the security of the uploaded files, the availability is already increased by the security requirements. Nevertheless, additional shares could be easily computed in order to increase the availability even further and decrease the chance of data loss or unavailability.

Redundancy works very similar to availability and could also be supported in theory. The key difference between availability and redundancy in our context is that by redundantly saving shares more than once at different cloud providers in a Secret Sharing scenario. In the case of availability more shares would be generated, whereas in the case of redundancy the same shares would be stored multiple times at different locations.

4.2 Model Overview

In this chapter we will introduce three models for using Secret Sharing Cryptography that allow us to securely store data in the cloud. In Chapter 2 we listed several current issues with cloud computing, including vendor lock-in, malicious outsiders and insiders, regulatory compliance and data location and these issues will be addressed in our model definitions.

The first model (Model 1) will introduce our basic idea to secure data storage in the cloud by using Threshold-Secret Sharing Cryptography. We will describe the benefits, challenges and potential use-cases as well as the decision procedure for the distribution of shares in this simple scenario.

Following that we will look at trust and collaboration in *Model 2* in a static model and how we can extend *Model 1* to describe trust and collaboration between providers and how this influences the decision procedure, which is used to distribute shares. This will allow us a more fine-grained selection of available storage providers to choose from.

In an extension to this model we will introduce the notion of *Collaboration Sets*, which will allow us to model legal bindings that may affect cloud providers and the user's data and model these bindings in terms of collaboration and trust.

Finally, *Model 3* will extend this static collaboration model to a dynamic setting similar to *Social Secret Sharing*, that was introduced in Chapter 3.

4.3 A Model Without Collaboration

The Model Definition

The first model, termed "Model 1(M1)", represents the basic application of Threshold-Secret Sharing Cryptography in the cloud with neither "collaboration" nor a "trust weighting" of the cloud providers taking place.

The properties of threshold-secret sharing in the context of secure cloud storage only demand that no more than $k-1$ shares may be uploaded to a single cloud storage provider in order to guarantee that the secret cannot be recovered with the shares present in one location alone.

Example. Figure 4.1 depicts a possible scenario for *Model 1*. Let us assume the user has access to three different, independent cloud storage providers *CSP #1, CSP #2 and CSP #3* and intends to upload the *File F1* in the cloud in such a way that the providers cannot recover the secret on their own. The user chooses a *(3,5)-threshold scheme* and the *decision procedure* will verify that the scheme can be applied with the number of available CSPs and consequently allow the procedure to continue, because the shares can be distributed in such a way that the threshold condition will not be broken.

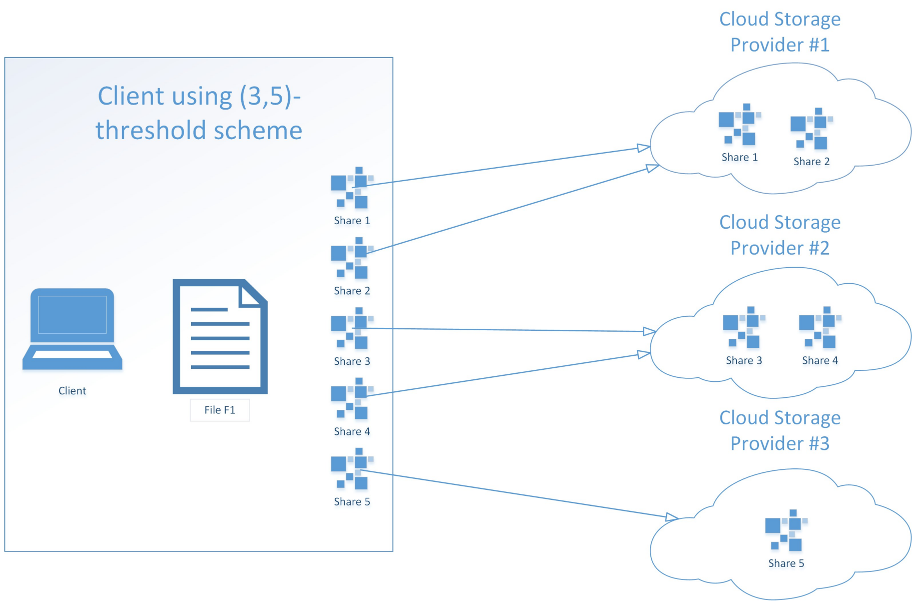

Cloud Storage Provider #1
Client using (3,5)-threshold scheme
Share 1
Share 2
Cloud Storage Provider #2
Share 3
Share 4
Client
File F1
Share 5
Cloud Storage Provider #3
Share 1
Share 2
Share 3
Share 4
Share 5

Figure 4.1: Model 1 without collaboration and trust weighting.

Subsequently, a secret sharing routine will be applied to the file *F1*, which will result in the creation of 5 shares, that can be stored in the cloud and used to recover the secret if at least three out of five shares are available.

In the Model 1 version the decision procedure will try to maximize *availability* and *security* by distributing the individual shares across as many different CPSs as possible. In our example this means putting two shares each on two CSPs and one share on a single CSP. There will be no particular decision involved in picking the one server that only gets one share versus the other two with two shares each.

In case of more providers, for instance 4 or 5 available providers in a $(3,5)$-threshold secret sharing scheme, the decision procedure would place one share on each server in the latter case and an additional share on the first server in the former case. Consequently, the scheme achieves a maximum of availability and security, because *a*) it is the most resistant against system failures compared to storing more shares on single nodes and *b*) an adversary will acquire the least amount of shares using this distribution.

Decision Procedure

The decision procedure for share distribution is rather simple in the model without collaboration as was already indicated in the previous section. Algorithm 4.1 checks whether a distribution is possible or not:

```
Input: Parameters $k, n, p$, the threshold, number of shares and number of providers
Output: Boolean: a distribution is possible or not
1 Algorithm check()
2     return $(k-1) \cdot n \geq p$
```

Algorithm 4.1: check() tests whether a distribution is possible or not.

If Algorithm 4.1 returns $false$, then the distribution cannot satisfy the threshold condition. That is, there are not enough providers to distribute at most $k-1$ shares per storage provider and the secret would not be secure.

The distribution algorithm of the decision procedure is simple as well, as can be seen in Algorithm 4.2:

```
Input: A set of share identifiers $S = (s_1, s_2, ..., s_n)$ and a set of provider identifiers
       $P = (p_1, p_2, ..., p_m)$
Output: A set of key-value pairs that represent the distribution:
        $D = ([s_1, p_1], [s_2, p_2], ..., [s_n, p_k], )$
1 Algorithm check()
2     for ($i = 1; i \leq S.length; i$++) do
3         $D[S_i] = P_{(i \bmod P.length+1)}$;
4     end
5     return $D$
```

Algorithm 4.2: distribute() returns a set of key-value pairs with the share-id as key and provider-id as value.

The algorithm simply iterates through the providers and assigns a share until all shares are distributed. It assumes that it is only run when the $check()$ procedure has returned successfully. The decision procedure aims for maximum availability, since as many providers as possible are used, but does not care, which provider gets more shares than the others, what is the expected behavior since we do not rate the providers in any way.

Characteristics, Evaluation And Use-Cases

Model 1 is a direct application of Threshold Secret Sharing Cryptography in a cloud storage environment. Therefore it is easy to implement and use, because the user does not have to worry about encryption keys and recovery of the files as long as $kshares$ can be recovered from the cloud providers that were used in hosting the file.

In Chapter 5 we will show how this model can be implemented and used to provide a similar user experience to existing, non-secure cloud storage solutions like *Dropbox, Google Drive* etc. This model is also a good choice when the user is not worried about collaboration between cloud providers in order to break the threshold scheme and simply wishes an easy method to distribute his or her shares on a small number of providers.

An example would be a $(2,3)$-*secret sharing scheme*, where the user stores the shares on three providers and needs two in order to recover the data, while one provider may be down at the cost of additional storage and bandwidth requirements for uploading and recovering the shares.

The primary advantages of the system compared to these solutions therefore are:

1. data is encrypted and cannot be read by the cloud storage providers
2. the user does not have to worry about key encryption management
3. increased availability and redundancy is inherent to this scheme
4. no vendor lock-in, multiple storage solutions can be used

The model does come with a few disadvantages as well. Being *information theoretically secure* requires the individual shares to be at least as large as the original data file, so that information about the secret is guaranteed not to be leaked with less than k shares. It follows that compared to unencrypted or key encrypted schemes this scheme requires more bandwidth and storage space when used and it may not be suitable for very large files.

Collaboration is another property that cannot be addressed in this model, because it only decides share distribution based on the threshold limitation.

Another limitation of this system is that it cannot take certain user preferences into account. For example, a user could not express the wish to prefer one provider over another, because the model does not tale any differences between the providers into account. This is one of the limitations that will be addressed in the next section when we introduce a model with static collaboration.

The primary disadvantages of the system compared to these solutions therefore are:

1. increased bandwidth and storage requirements
2. no means to consider collaboration or legal regulations
3. no means to consider user preferences with regards to provider selection

4.4 A Model With Static Collaboration

The Model Definition

In the second version, Model 2(M2), the cloud providers can be weighted by a trust value $\mathcal{T}(p)$ in the range from $[0, 1]$, where $\mathcal{T}(p) = 0$ means that the cloud provider cannot be trusted under any circumstance and therefore collaboration is guaranteed and $\mathcal{T}(p) = 1$, where the likelihood of collaboration with any other party is guaranteed to be impossible.

The decision procedure from *Model 1* gets extended as well to allow for the consideration of trust/collaboration values of the different cloud providers. This means that providers with a higher trust value $\mathcal{T}$ are preferred to such where the $\mathcal{T}$ is lower.

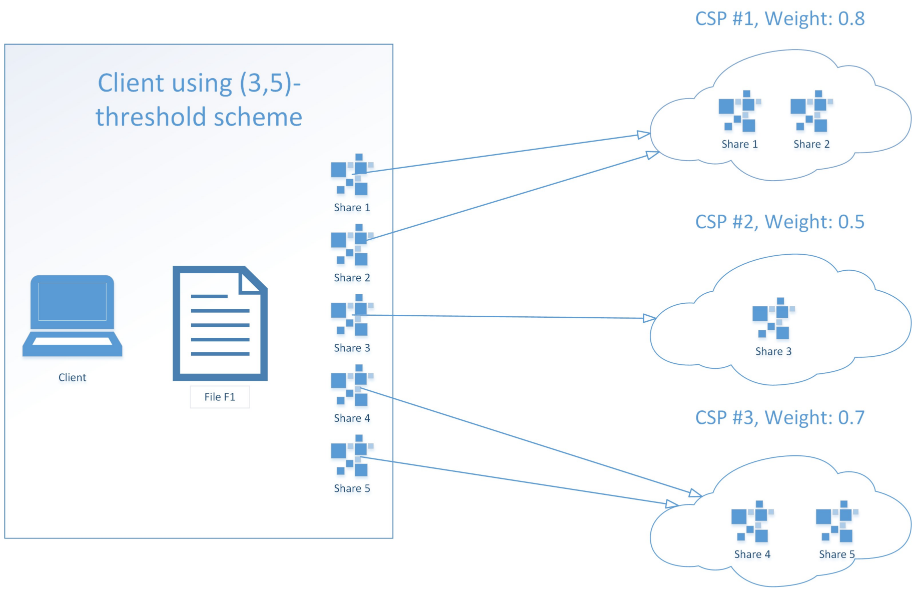


Figure 4.2: Model 2 with collaboration defined as trust weighting.

Example 1. Consider the example shown in Figure 4.2. The setting is similar to the example without collaboration with the user intending to distribute the shares of the threshold scheme on three different cloud providers. The distinction here is that the user *prefers* some cloud providers over others, or in other words, he may believe that some cloud providers are more likely to share their data with somebody else than others. Therefore the user assigns *trust values* to each cloud provider indicating the preference, which does not, however, exclude that provider from being used.

In the example, the *extended decision procedure* will check again whether it is feasible to use the defined *(3,5)-threshold scheme*, which it is, because three providers are available like in the previous example and then distribute the shares according to the threshold condition and the trust values.

Unlike the procedure in *Model 1*, the decision procedure in *Model 2* decides to place only one share on CSP #2 and place the other share on CSP #3 instead, because it has the higher trust value. Compared to the distribution of shares in Figure 4.1, this version allows a better placement of shares among the different cloud storage providers.

Example 2. There is one problem though with this decision procedure that is not apparent in the simple example given above. We also need to define the relationship between trust values and security and availability, and which of these properties takes precedence in case the threshold condition does not overrule them.

Consider the example in Figure 4.3: The problem becomes clearly visible when trying to place *Share 4*, even though we could have picked any share after the first one as well: When choosing the placement of Share 4 we have to decide whether to prefer CSP #4 with a low trust value to store a single share vs choosing CSP #1, which has a high trust value but already has one share? It becomes clear that the trust value alone is insufficient to determine the distribution of shares in this scenario and that we need to extend this model further:

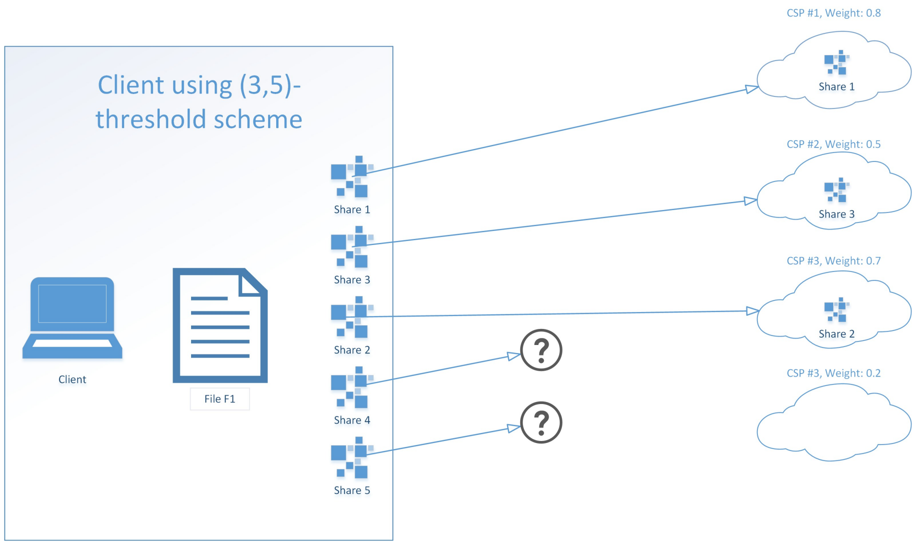


Figure 4.3: Model 2 showing decision problem: trust vs. number of shares and availability.

We will therefore introduce a second trust value, termed *relative trust value* that is calculated to decide where to put a share in the case described above:

Relative Trust Value In addition to the basic trust value $\mathcal{T}(p)$, we define $\mathcal{T}_i(p)^r$ as the *relative trust value* of provider p at time i during the distribution phase as follows:

$$\mathcal{T}_i^r(p) = \begin{cases} \frac{\mathcal{T}_i(p)}{\delta_i(p)} & \text{if } \delta_i(p) > 0 \\ \mathcal{T}_i(p) & \text{if } \delta_i(p) = 0 \end{cases}$$

where $\delta_i(p)$ is the number of shares that are currently stored on cloud provider p at time i.

In the next section we will show how the extended decision procedure will resolve the distribution problem depicted in Figure 4.3.

Decision Procedure

Definition. The decision procedure using threshold limit, basic trust value and relative trust value works as follows in distributing shares to the available providers:

1. Limit the number of available providers to those that would not break the threshold limit
2. Compute $\mathcal{T}_i^r(p)$ for all remaining providers if the share would be placed on them. If a single provider has the highest $\mathcal{T}_i^r(p)$, then choose p.
3. If there are more than one provider remaining with the same $\mathcal{T}_i^r(p)$, choose the one with the higher basic trust value $\mathcal{T}(p)$.
4. If there is still more than one provider, choose one of the remaining providers at random.

A more formal definition of the algorithm is shown in Algorithm 4.3:

```
Input: A set of share identifiers $S = (s_1, s_2, ..., s_n)$, a set of provider identifiers
       $P = (p_1, p_2, ..., p_m)$ and the threshold $t$
Output: A set of key-value pairs that represent the distribution:
        $D = ([s_1, p_1], [s_2, p_2], ..., [s_n, p_k], )$
Algorithm distribute()
    for p in P do
        providers = p.filter(f -> f.shares < t - 1)
            .sort(s -> s.getRelativeTrust)
            .sort(s -> s.getTrust)
        D.add(p, s);
    end
    return D;
```

Algorithm 4.3: distribute() returns a set of key-value pairs with the share-id as key and provider-id as value.

Example. We will resume the problem given in Figure 4.3 and show how the conflict presented there can be resolved by using the new *decision procedure*.

We have CSP #1 with a trust value of 0.8, CSP #2 with trust value 0.5, CSP #3 with 0.7 and CSP #4 with 0.2 and no shares have been distributed yet and assume that no shares have been distributed yet.

Based on the decision procedure, all providers are eligible for placement, since all providers have less than $k - 1$ shares stored - in fact none.

The relative trust value at time 1, which would be after placing one share, is identical to $\mathcal{T}(p)$. Therefore CSP #1 is the single provider with the highest $\mathcal{T}_1^r(p)$ and gets assigned share 1.

For the second share, we have the following relative trust values: $\mathcal{T}_2^r(CSP1) = 0.4$, $\mathcal{T}_2^r(CSP2) = 0.5$, $\mathcal{T}_2^r(CSP3) = 0.7$ and $\mathcal{T}_2^r(CSP4) = 0.2$. Therefore share 2 is assigned to CSP #3, share 3 to CSP #2, share 4 on CSP #1 again and share 5 on CSP #3 based on the relative trust values. The final configuration can be seen in Figure 4.4.

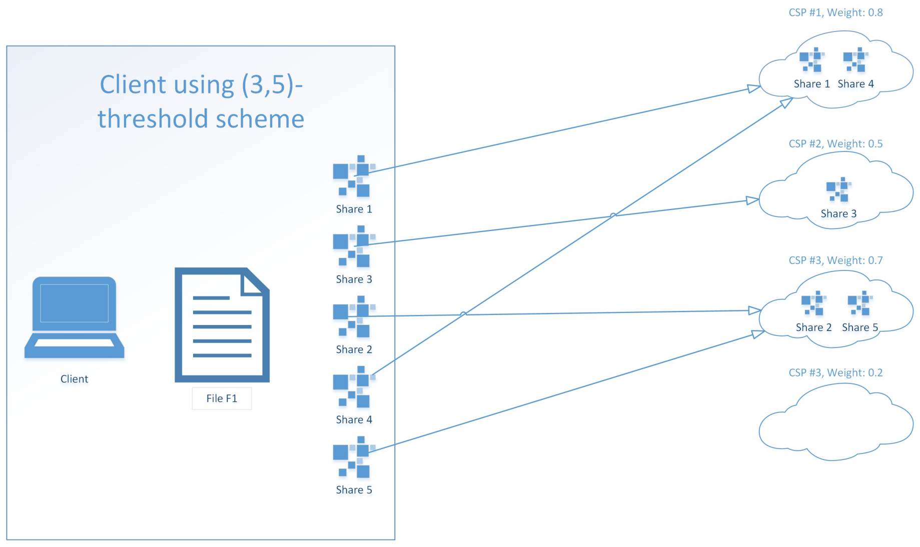


Figure 4.4: Model 2 using relative trust values for distribution.

Extension: Collaboration Sets

A challenge in cloud computing that we identified in Chapter 2 was the issue of *data security laws*, in particular laws granting certain entities the rights to extradite the information stored with cloud service providers. This creates a problem in our model, because we have no way to apply trust values based on regional or political borders to cloud providers. While the trust value of individual providers within a country may be high, existing laws that may make it easy to get access to all the files stored on these providers, would lower the trust value of all the providers within that country collectively.

We introduce the notion of *Collaboration Sets*, which extends the provider's basic trust value with a value of the set it belongs to, in order to describe the security of the data a user may store based on geographical or political factors as well as individual trust in storage providers.

This extension results in two more trust values, the *trust value of the collaboration set* $\mathcal{C}(p_1, p_2, ..., p_j)$, which is analogous to the trust value $\mathcal{T}(p)$ of the provider and a *relative trust value of the collaboration set* $\mathcal{C}_i^r(p_1, p_2, ..., p_j)$, which is analogous to the relative trust value $\mathcal{T}_i^r(p)$ of the of the provider:

$$\mathcal{C}_i^r(S) = \begin{cases} \frac{\mathcal{C}_i(S)}{\delta_i(S)} \mathcal{T}_i^r(p) & \text{if } \delta_i(S) \geq 2 \\ \mathcal{T}_i^r(p) & \text{if } \delta_i(S) < 2 \end{cases}$$

where $\delta_i(S)$ is the number of providers that currently have shares within the collaboration set. This means that the trust weight of the collaboration set does not have any impact unless more than one provider within this set has been used.

Extended Decision Procedure

A formal definition of the algorithm of the extended decision procedure is defined in Algorithm 4.4:

```
Input: A set of share identifiers $S = (s_1, s_2, ..., s_n)$, a set of provider identifiers
       $P = (p_1, p_2, ..., p_m)$ and the threshold $t$
Output: A set of key-value pairs that represent the distribution:
        $D = ([s_1, p_1], [s_2, p_2], ..., [s_n, p_k], )$
Algorithm distribute()
    for p in P do
        providers = p.filter(f -> f.shares < t - 1)
            .sort(s -> s.getCollabTrust)
            .sort(s -> s.getRelativeTrust)
            .sort(s -> s.getTrust)
        D.add(p, s);
    end
    return D;
```

Algorithm 4.4: distribute() returns a set of key-value pairs with the share-id as key and provider-id as value.

- The algorithm initially limits the number of available providers to those that would not break the threshold limit by adding another share to it.
- Then it computes the highest relative trust value $\mathcal{T}_i^r(p)$ for each collaboration set.
- If there is more than one provider remaining with the same $\mathcal{T}_i^r(p)$, then the one with the higher basic trust value $\mathcal{T}(p)$ is chosen.
- In case there is still more than one provider, then one of the remaining providers is chosen at random.
- Following that the algorithm multiplies $\mathcal{T}_i^r(p)$ by $\mathcal{C}$ of the collaboration set to receive the relative trust value of the collaboration set $\mathcal{C}_i^r(p_1, p_2, ..., p_j)$.
- If a single provider remains, then this provider is used.
- Otherwise the one with the highest $\mathcal{C}$ is used, then $\mathcal{T}_i^r(p)$, $\mathcal{T}(p)$ and finally a random provider in case all these properties are equal.

Example. To give an example let us take a look at Figure 4.5.

By the rules of the decision procedure the first share, share 1, gets assigned to CSP #4. Since the distribution is just initializing, it would be the single share on any of the storage providers and therefore $\delta_i(S)$, the number of providers that currently have shares within the collaboration set is less than 2, which means that the relative trust value of the collaboration set does not have any

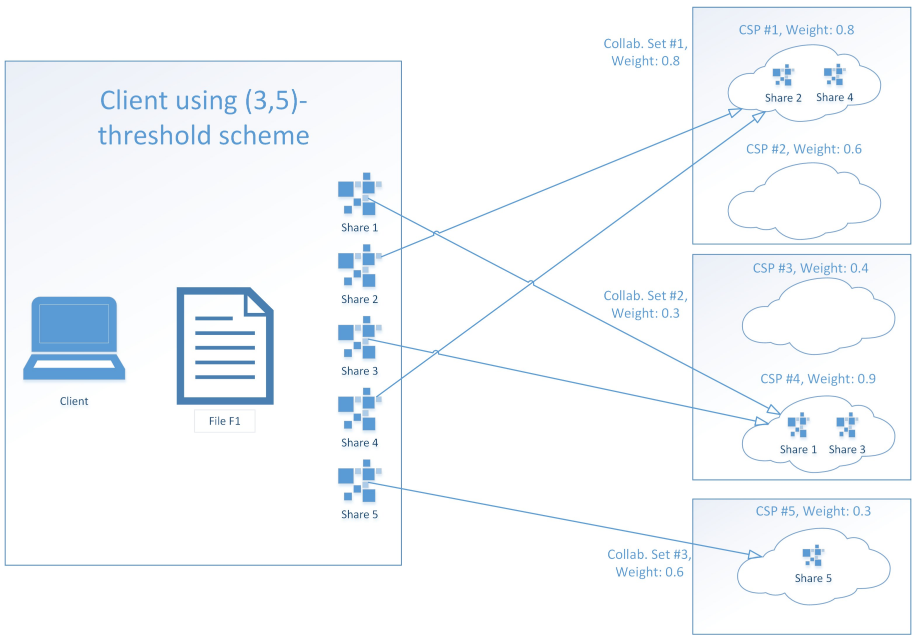


Figure 4.5: Model 2a with collaboration sets and trust values.

effect and the CSP with the highest relative trust value $\mathcal{T}_i^r(CSP4) = \mathcal{T}(CSP4) = \mathcal{C}_i^r = 0.9$ will be chosen. Share 2 will be placed on CSP #1 by the same reason that share 1 was placed on CSP #4, because $\mathcal{T}_i^r(CSP1) = \mathcal{T}(CSP1) = \mathcal{C}_i^r = 0.8$ is the best available provider.

Share 3 will be placed on CSP #4 and Share 4 on CSP #1. Finally share 5 will be placed on CSP #5, because the relative trust value of the collaboration set $\mathcal{C}_i^r = 0.3$ and higher than the $\mathcal{C}_i^r = 0.24$ of CSP # 2 for instance.

Without the collaboration sets, as seen in Figure 4.6 the distribution would have looked quite different. Shares 1 and 4 would have been placed on CSP #1 and shares 2 and 5 on CSP #1 and share 3 on CSP #2. This would have resulted in k shares being present in collaboration set #1 and the secret may have been revealed in that case. In case of share 3 it would have made no difference had the $\mathcal{T}_i^r$ for share 3 chosen CSP #3 instead of CSP #1 since they had the same value. In that case collaboration set #2 would have had more than k shares.

Characteristics, Evaluation And Use-Cases

This model solves two of the three main issues of *Model 1*, that is, the capability to let the user give a preference to choose one provider over the other by way of the trust value and to define collaboration both between individual providers as well as the modeling of collaboration between multiple providers through the use of collaboration sets.

Even though this model is an improvement in the features over the previous model, it does not come with any additional challenges in terms of bandwidth or storage space requirements. This model is therefore usable in any scenario in which *Model 1* was considered for and more complex situations, where collaboration between providers has to be taken into account as well.

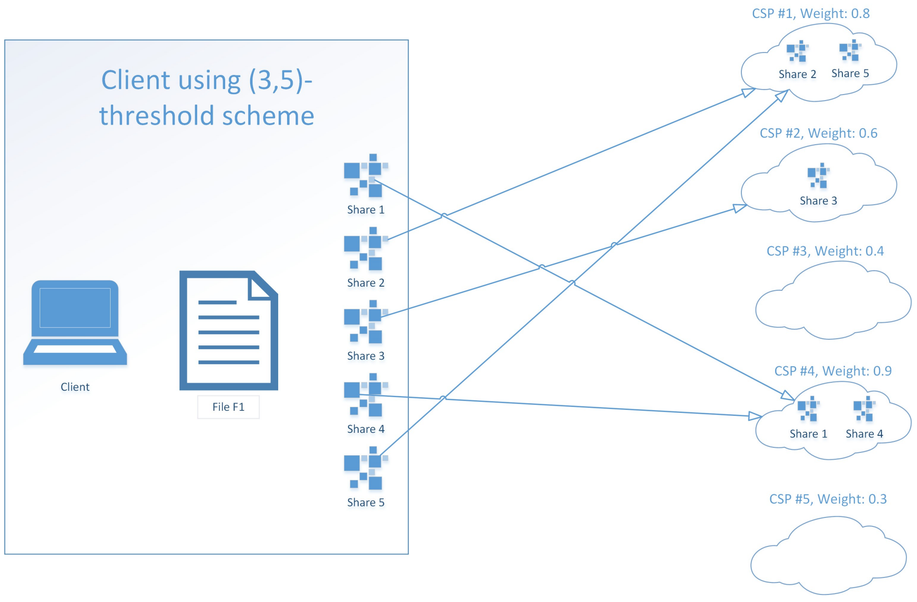


Figure 4.6: Model 2b without collaboration sets, trust values identical to Figure 4.5

4.5 A Model With Dynamic Collaboration

In the final model, *Model 3(M3)*,we will look at potential solutions for dealing with a dynamic environment, in which the trust values of single cloud service providers as well as collaboration sets may change with time.

In Chapter 3 we have introduced the work that was done on *social secret sharing(SSS)*, where the trust values and the reputation of participants was periodically updated based on their behavior during the execution of the protocol. The protocol that was used in this research including the assumptions that were made deviate significantly from our domain:

1. It is assumed that there exist many willing participants that can be enrolled and disenrolled at any time, when the need arises.

2. In SSS, the weight of the shares and the sum of weights that is supposed to be stored in one location is always assumed to be *much less* than the threshold.

These two assumptions are problematic for our use-cases. In general, we do not intend to enroll and disenroll cloud storage providers frequently, but rather only adjust their trust value based on their behavior during the protocol or on facts that are outside the scope of the protocol, for instance, a change in the legal status or company mergers with other storage providers that could have an effect on the security of the data we have stored in the cloud.

The second problem is the assumption that the weight of the shares or the number of shares in general stored on a single provider is much less than the threshold. We find this assumption not very practical for our model, because of the increased bandwith and storage requirements by increasing the number of available providers and the availability of providers in general. We do not consider it likely to be able to use more than a dozen truly independent cloud storage providers to choose from. Therefore our options in this regard are limited and we have to take this into consideration when designing our distribution and trust functions.

Another problem we face with the dynamic environment is the question of whether the user will be able to adapt the trust values in time to counter changes in the real world and whether anything can be done to successfully adjust the distribution of files that are already stored in the cloud.

Example Let us assume we have distributed a file according to Figure 4.5. At time period i that distribution scheme was considered secure. But then the trust values of one or more of the providers we have used change or a single provider switches from one collaboration set to a different set due to a relocation to another country in time period $i + 1$ as seen in Figure 4.7.

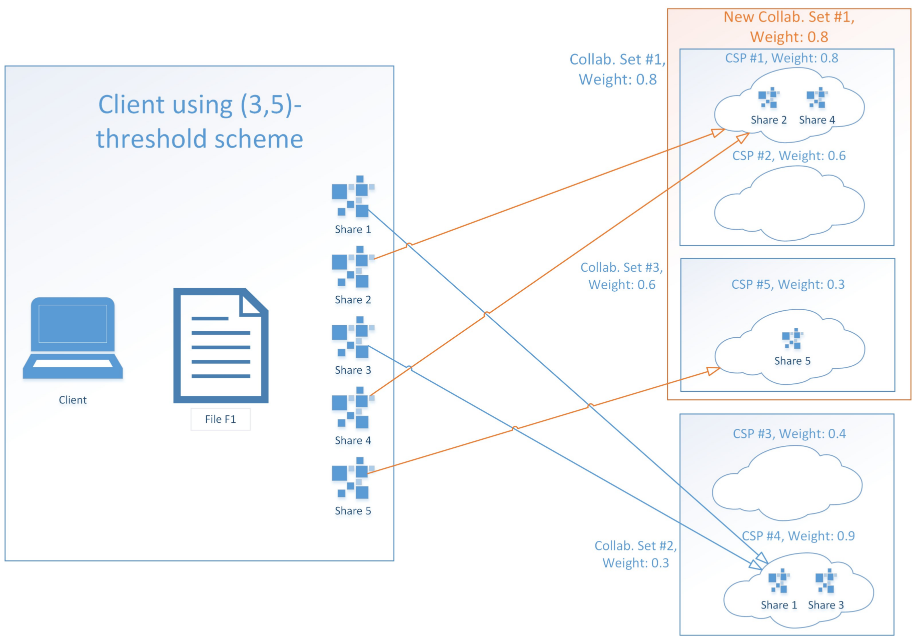


Figure 4.7: Model 3: new collaboration set #1 in red replaces the previous collaboration sets #1 and #3 in time period $i + 1$

When we assume that cloud providers store the data even after we delete the file or change the secret sharing files by using pro-active secret sharing, we still have to assume that all the providers in question may have backed up the previous version from time period i, no matter what we do, we cannot change the fact that the files are now potentially recoverable by the newly collaborating providers.

The Adjustment Phase

In the adjustment phase, which is similar to the adjustment/tuning phase in *social secret sharing*, we have to options to pro-actively update the shares on the cloud storage providers:

1. Adopt the classic pro-active scheme introduced in Section 3.5 and used in Social Secret Sharing

2. Use a simpler redistribution mechanism

The first version has already been covered in detail in Chapter 3. Adopting this variant would change the architecture and requirements of our model significantly. The models introduced in this chapter so far only required the user to acquire cloud storage space, using the pro-active secret sharing scheme to support a dynamic collaboration scenario would then require to purchase computation power as well in order to update the shares on the servers directly. It would also require all the other necessary features that are assumed to be available in such a protocol including a secure broadcast channel and bi-directional communication channels between all the providers.

This second requirement poses another problem we have not had to deal with before and that is that so far the providers were not aware of the fact that the user was using them to distribute the shares and this introduces another security issue.

The second option would be to adopt Model 2 with Collaboration sets and simply re-distribute the files using new polynomials from the client directly. The advantage would be that the user would not have to pay for a more expensive cloud architecture, but would regularly have to upload the updated shares to the cloud again.

4.6 Summary

The models we have proposed here try to solve many currently existing problems in cloud computing:

By using secret sharing cryptography we achieve better availability and redundancy and at the same time also get encryption, so that a single or even multiple cloud providers cannot recover the secret even when collaborating.

The proposed models also try to give the user the freedom to express preferences regarding the choice of service providers by assigning trust values to individual providers and to providers as part of collaboration sets.

CHAPTER 5

A Secure Cloud Storage Framework

5.1 Overview

In Chapter 4 we have proposed three models that show the *applicability of Secret Sharing Cryptography* in a cloud storage setting. In this chapter we want to transition the static models with and without collaboration from theory into a practical proposal for implementation.

In Chapter 5.1 we will introduce two possible *modes of operation* that cover secure cloud storage on an individual, singular user level and secondly, expand it to cover access for multiple users in a simple setting using a trusted server to act as a gateway to the cloud.

Subsequently, in Section 5.2 we will cover the protocols required for reading from and writing files to the cloud in detail. This includes sequential diagrams that show every single step that is necessary to encrypt the file, generate a strategy for share distribution and finally upload the shares to the cloud as well as the reverse task of retrieving shares from the cloud and recovering the original file.

Section 5.3 will cover critical aspects of the implementation of the framework, including file transfer to and from the cloud service providers as well as distribution algorithms to achieve secure share distribution in the cloud and how to access multiple cloud providers easily with multi-cloud libraries.

Modes of Operation.

The framework supports two modes of operation:

- stand-alone operation
- server-enhanced operation

The stand-alone version is meant for users who want to store or backup their files securely in the cloud. The server-enhanced version is geared towards small-to-medium businesses which require more options such as file or folder sharing, administrating access rights and has its core business within its own premises or hosted at a traditional provider. In order to leverage cloud service providers without giving away its secrets this company employs our proposed system where a trusted server is used as a gateway to cloud providers.

Stand-alone client. The client version, depicted in Figure 5.1, connects directly to the cloud providers that the user has configured to use with the system by providing the credentials to log in to the storage service.

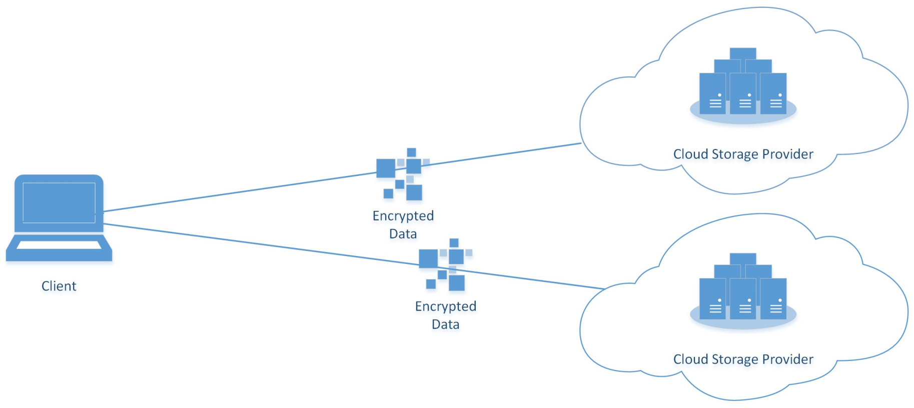


Figure 5.1: The Client-based model. High level overview.

The user can then apply trust values to the providers and define secret sharing rules by providing a (t, n) secret sharing scheme and start uploading files. The shares are uploaded in a convention-over-configuration approach, where the shares are bundled in a folder named after the original file. This enables the client to reconstruct a file list based on the existing folders across the configured storage providers. In order to retrieve shares from the cloud the user can then select the respective file and provide the necessary number of shares that need to be retrieved unless this information is already available locally through meta-data.

Server-enhanced version. The server-enhanced version, depicted in Figure 5.2, provides additional features through the use of a trusted server/gateway server that channels the communication of multiple clients to the cloud storage provider.

This version provides all features of the stand-alone version and additionally, the trusted server can be used to provide additional user and group policies to handle access privileges or to act as a cache for files stored in the cloud.

Meta-data Database Both versions of the framework implementation feature a light-weight database to store meta-data locally, either on the client directly or on the trusted server. This includes the distribution parameters, destination of shares and file attributes such as name, size and last accessed or modified.

This has the advantage that the program does not have to query the cloud providers every time the user wants a listing of the available files. It is important not to confuse this as another

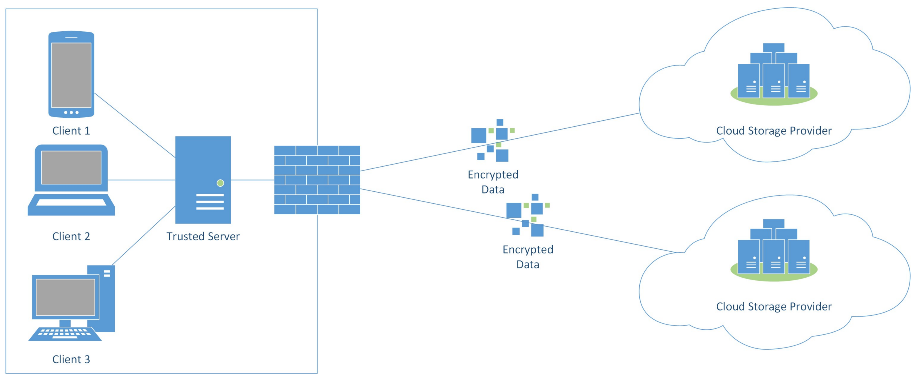


Figure 5.2: The Server-based model. High level overview.

way to introduce the disadvantages of a key encryption system: the framework has to function without the presence of the meta-data as well or to be able to re-create the meta-data in case of a system migration or setup of a new computer to make use of the cloud storage.

This will be achieved by employing a convention-over-configuration approach, where the way the data is stored on the cloud provider will infer the necessary information through a look-up of the file system tree hierarchy. The only aspect that cannot be inferred at this point from the data is the threshold parameter k, which the user would have to provide in the case of meta-data loss, even though that could also be stored in the cloud separately.

5.2 Transfer of Shares

Uploading shares to the cloud

Figure 5.3 shows the steps that are required to upload a file securely to the cloud:

Step 1

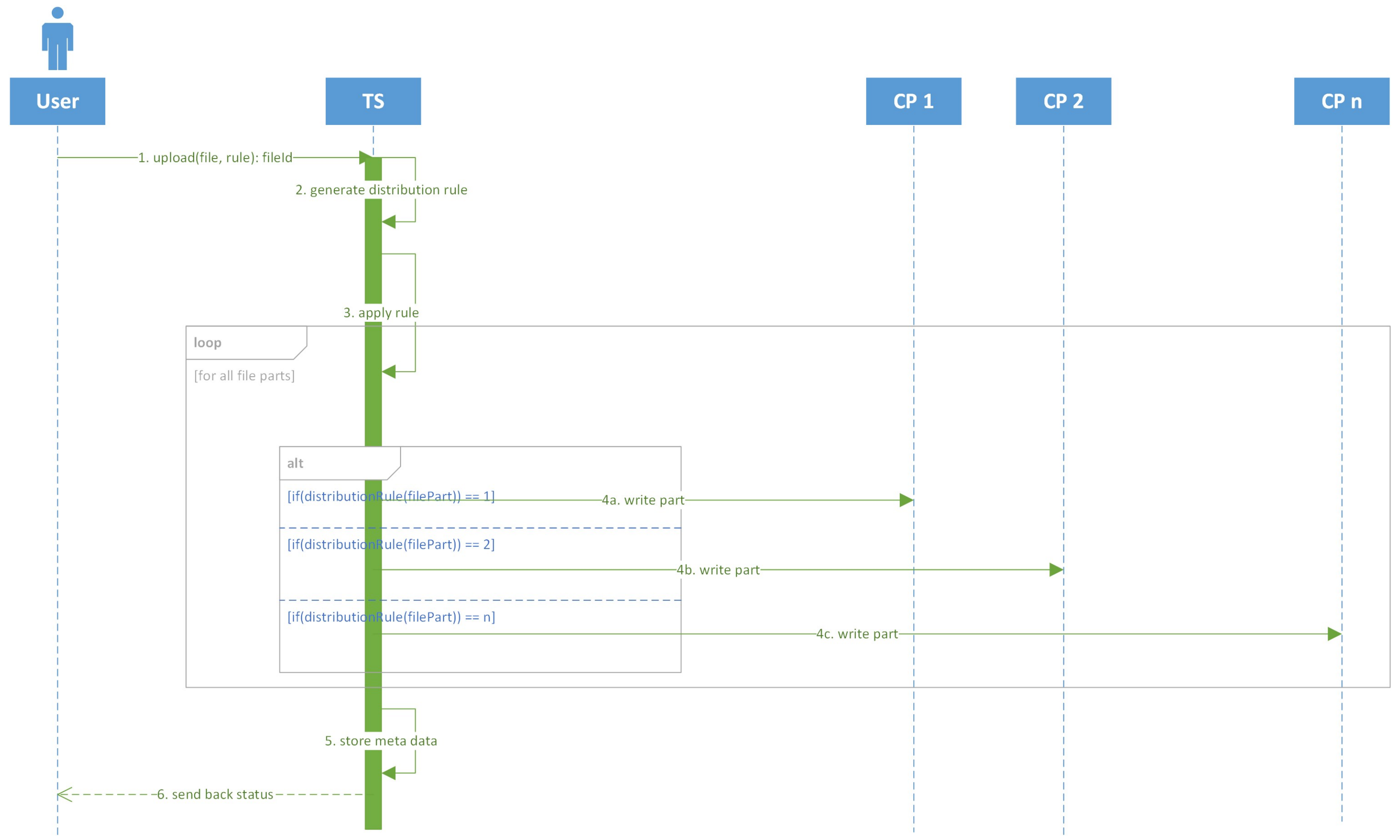
The user uploads a file and the rule settings to the trusted server. This can either be a pre-defined and already existing rule, in which case it is sufficient to provide the *ruleId*, or otherwise a new rule to use, which should consist of the threshold scheme that is used, i.e. the t and n parameters of secret sharing scheme and the trust values of the providers.

Step 2

The client or the trusted server now generates a distribution strategy of the shares among the available cloud storage providers. The details of this procedure are explained formally in Chapter 4 and a working Java implementations can be found in Listing B.1 without and with collaboration sets. The client/trusted server writes the meta-data files to the local database including the sharing scheme, location of shares and additional file properties.

Step 3

The program initializes the gfShare libary and generates the shares. Then the distribution



User
TS
CP 1
CP 2
CP n
1. upload(file, rule): fileId
2. generate distribution rule
3. apply rule
loop
[for all file parts]
alt
[if(distributionRule(filePart)) == 1]
4a. write part
[if(distributionRule(filePart)) == 2]
4b. write part
[if(distributionRule(filePart)) == n]
4c. write part
5. store meta data
6. send back status

Figure 5.3: High-level Sequence Diagram for writing a file.

of the actual files is started by looping through all the shares. It is important to take fallback scenarios into account here. For instance what to do in case a provider is unreachable, recovery after connection loss, etc.

Step 4
Every share is uploaded to a specific provider according to the distribution strategy. In Listing B.3 we provide exemplary code to upload files to various cloud provider by the use of the jClouds library, which enables uploading of files to many existing cloud storage providers.

Step 5
After the upload is completed, the files are marked as uploaded in the meta-data record. In case of failure in any of the steps above the system has to recover successfully, for instance by deleting files that were uploading in case the distribution as a whole failed.

Step 6
The user is finally notified of the successful storage of the shares in the cloud or receives a failure notification in case something went wrong.

Recovering shares from the cloud

Figure 5.4 shows the principal work flow of retrieving the individual shares and the recombination to recover the original file using a model that uses a trusted server as a gateway to the cloud. The protocol and the steps necessary are identical in a model where the client machine of the user takes the role of the trusted server and will therefore not be shown separately.

The protocol consists of the following six steps when retrieving files from the cloud:

Step 1
The user requests a file from the service by invoking a method that takes the *file_id* as input.

Step 2
The trusted server (TS)/client makes a lookup in the local database to see if the requested file exists. Every file that was transmitted to the TS will have an associated *metadata file* attached to it. The *metadata file* contains information about the file characteristics, such as file name, file owner, access rights, revisions and the *distribution rule*, with which the file was stored in the cloud and location-specific data how to re-assemble the file again.

Step 3
The client/TS now knows where all the shares are placed in the cloud and ranks them according to their trust value. That is, it assumes that the providers with higher trust values are also the most reliable and the chances are better that these servers will be available.

Step 4
It iterates through at least t servers and fetches the shares from the providers in order to reconstruct the original file. In case a server does not respond or the transmission times out, it will try another provider if available.

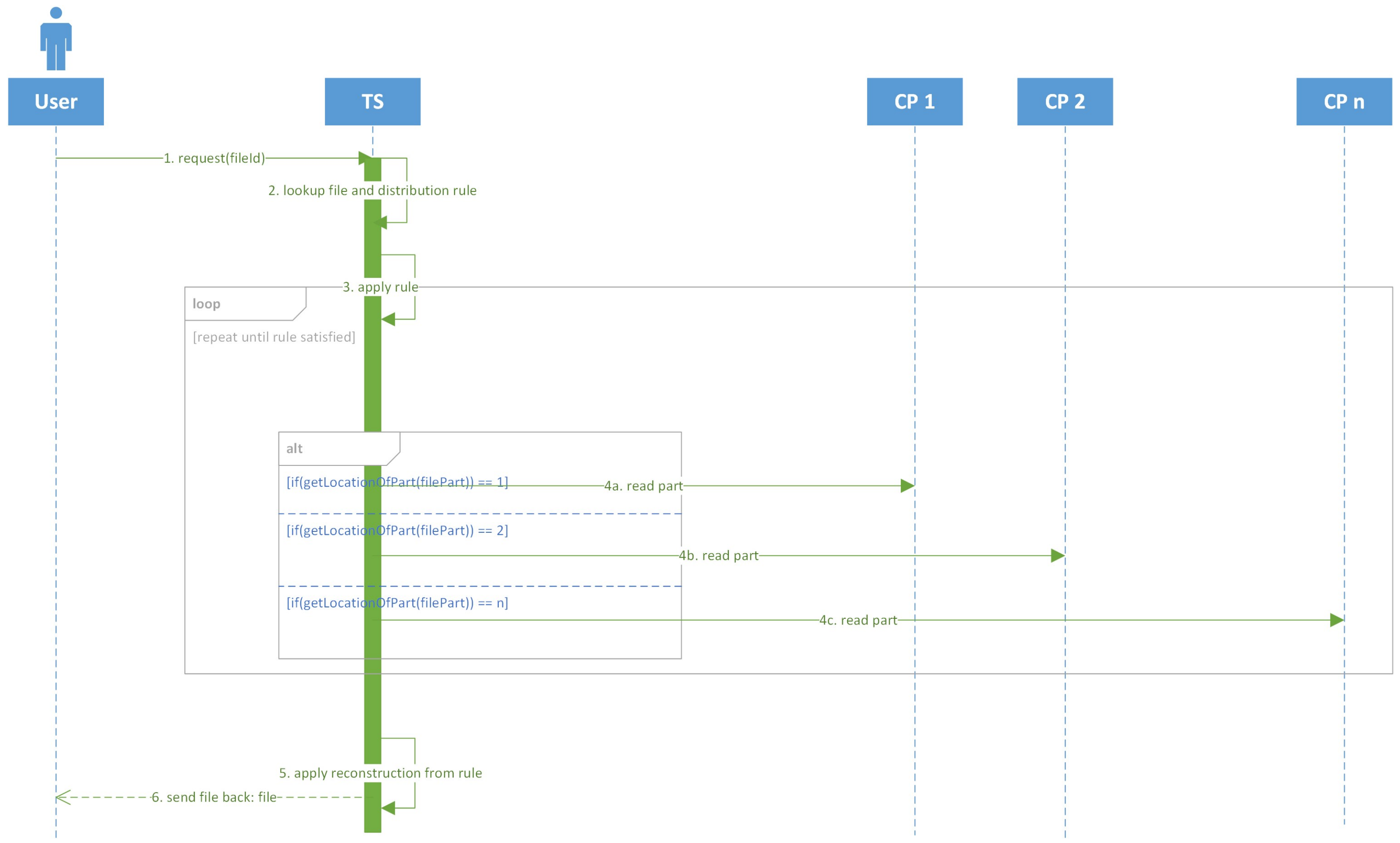

User
TS
CP 1
CP 2
CP n
1. request(fileId)
2. lookup file and distribution rule
3. apply rule
loop
[repeat until rule satisfied]
alt
[if(getLocationOfPart(filePart)) == 1]
4a. read part
[if(getLocationOfPart(filePart)) == 2]
4b. read part
[if(getLocationOfPart(filePart)) == n]
4c. read part
5. apply reconstruction from rule
6. send file back: file

Figure 5.4: High-level Sequence Diagram for reading a file.

Step 5

In case sufficient shares have been fetched the reconstruction is started by combining the shares using the secret sharing library GFShare.

Step 6

After completion the original file is available on the client or can be transmitted to the user by the trusted server.

5.3 Critical Implementation Aspects

We have identified three critical aspects for the successful implementation of the proposed models in Chapter 4:

1. Secret Sharing Libraries
2. Distribution algorithms
3. Multi-Cloud Support

Secret Sharing: GFShare In order to achieve the goals described in Chapters 4 and 5 we need secret sharing implementations that enable us to encrypt files in the range of a few kilobytes to several megabytes in such a time, that the encryption time does not add significant overhead to the overall transfer time. In Appendix A.1 we have evaluated several publicly available secret sharing libraries and finally chose GFShare, because of its ability to encrypt very large files (in fact, there is no known limit to the size of the files), it's performance, as can be seen in Section A.2, which is more than sufficient for our needs and its ability to run on multiple platforms as it is written in C and does not contain any platform-specific code. GFShare can handle files in the range of several megabytes in less than 1 second encryption/decryption time, which is sufficient compared to the upload times of the files with a broadband internet connection.

Distribution Algorithms The distribution algorithms are at the heart of the framework. In Listing 5.1 we provide a Java implementation of the static distribution without collaboration, introduced as *Model 2* in Chapter 4 provide a distribution suitable for use cases such as personal cloud storage, where the user wants to securely store data on multiple cloud providers such that a single cloud provider or depending on the scheme even multiple cloud providers collaborating together cannot recover the original files. A more complete example of the distribution algorithms, including an implementation of all Model 1 and Model 2 examples in Chapter 4 can be found in Appendix B.1.

Listing 5.1: Code for the distribution.

```java
public Map<String, String> distribute(final int t, List<
    Provider> ps, List<String> sr) {
  Map<String, String> map = new HashMap<String, String>();
  for (String s : sr) {
    Iterable<Provider> threshold = Iterables.filter(ps, new Predicate<Provider>() {
      @Override
      public boolean apply(Provider p) {
        return p.getNumberofShares() < t - 1;
      }
    });
    Ordering<Provider> primary = Ordering.natural().reverse().onResultOf(new Function<Provider, Double>() {
      @Override
      public Double apply(Provider p) {
        return p.getNextRelativeTrust();
      }
    });
    Ordering<Provider> secondary = Ordering.natural().reverse().onResultOf(new Function<Provider, Double>() {
      @Override
      public Double apply(Provider p) {
        return p.getTrust();
      }
    });
    List<Provider> compound = primary.compound(secondary).immutableSortedCopy(threshold);
    Provider p = compound.get(0);
    p.shares.add(s);
    map.put(s, p.name);
  }
  return map;
}
```

Listing 5.1: Code for the distribution.

Multi-Cloud Support It is important for the user to be able to choose from as many cloud providers as possible. The Models in Chapter 4 require the use of multiple, independent cloud storage providers. Libraries such as *jClouds*[1] allow the use of different providers, such as Amazon, Rackspace, Azure, etc. with a standardized interface, making the development of a multi-cloud-supporting framework very easy. Listing 5.2 shows an example of how to transfer a file to a cloud provider, in this example, Amazon through the use of their API. As can be seen in Line 3 of the code, the storage provider information, including the type and credentials, can be provided as parameters allowing code reuse.

Listing 5.2: Code for uploading a file to the cloud using jClouds.

```java
1 public void SaveFileToBucket(File f) {
```

[1] `http://jclouds.apache.org`. Retrieved on 05/01/2014

```
  try {
    context = ContextBuilder.newBuilder("aws-s3")
             .credentials(accessKeyId, secretKey)
             .buildView(AWSS3BlobStoreContext.class);


    BlobStore blobStore = context.getBlobStore();
    blobStore.createContainerInLocation(null, container);
    Blob b = blobStore.blobBuilder(f.getName()).payload(f).
        build();
    blobStore.putBlob(container, b, PutOptions.Builder.
        multipart());

    context.close();
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

Listing 5.2: Code for uploading a file to the cloud using jClouds.

5.4 Summary

We have shown in this Chapter that Models 1 and 2 can be implemented using freely available tools, such as GFShare or jClouds.

The requirements for an implementation of Model 3 are much higher, however. In Chapter 3 we have shown the requirements for Verified Secret Sharing and Pro-active Secret Sharing that were mandatory in Social Secret Sharing. Those include bi-directional, secure communication channels between each participant and also another broadcast channel that would require the additional purchase of compute power as well as storage space in the cloud and the necessary infrastructure and services to guarantee the assumptions made to secure the protocol.

In Chapter 6 we will discuss these matters in more detail and propose solutions for Model 3 and also possible extensions to achieve better results in Models 1 and 2.

CHAPTER 6

A Critical Reflection

6.1 Comparison of Model 2 with Social Secret Sharing

Advantages	**Disadvantages**
client or trusted server manages storage distribution and recovery	no dynamic collaboration support (yet)
meta data storage is optional, only knowledge of threshold is required to recover secret in basic settings	size of required storage linear to number of shares
can be extended with traditional, of-the-shelf software components to support SMBs with additional features	
only requires cloud storage	
Improved distribution algorithm works close to threshold, important when there are few providers	

Table 6.1: Advantages and Disadvantages compared to Social Secret Sharing.

It is important to highlight the simplicity of the proposed models and framework in terms of implementation and usage. In a basic Model 1 or 2 setting, the system can be configured through the *convention-over-configuration* paradigm, where aside from the threshold parameter every other required information can be looked up in the cloud by the program or the user to recover shares to reconstruct the secret.

It is also feasible to extend the system through the use of traditional authorization and authentication systems in a small-to-medium business setting where a central server, called a *trusted gateway server* in the models in Chapter 4 can be tasked with user and group policy enforcement to restrict access to the shares in the cloud or to be used to keep a cache of frequently requested files for example in a LRU[1] queue or a similar construct to avoid unnecessary cloud lookups and fetches.

The distribution algorithms also function well with a limited number of providers. In social secret sharing the assumptions exlcuded this problem by specifying that the weight of the shares cumulated on a single provider would be significantly less than the threshold, effectively neutralizing that problem. In the context of our models this apporach was not practical and we therefore had to adapt our distribution scheme to support a number of shares on a single provider that is as close as possible to the threshold without reaching it.

Shortcomings of our proposed systems is a lack of dynamic collaboration support that exists in social secret sharing through the use of VSS and PSS schemes. We have discussed the problems with these technologies in Chapter 4 briefly and would like to reiterate the problems of the increasingly complex architecture these features require including computation power in the cloud, as opposed to only cloud storage in our models as well as pairwise bi-directional communication channels between all the storage/compute units and an additional secure broadcast channel. Added to that this system would require frequent updates of the shares resulting in increased bandwidth requirements to distribute share renewal data across all providers.

These features would also result in direct communication channels between cloud providers enabling a potentially easier collaboration between these parties.

6.2 Additional Extensions and Open Issues

Issue 1 *Required storage space can be traded with security by using computationally and not information theoretically secure secret sharing schemes.*

Issue 2 *Increased flexibility in terms of precedence of security, availability, collaboration to allow for better distribution algorithms.*

Issue 3 *The relative impact of (t,n) parameters could to be taken into account as well and set it in relation to the number of shares already allocated on a storage provider.*

Issue 4 *Extend the models to support process composition and to include other features aside from cloud storage and define intersections and security concerns.*

Issue 5 *The models can be extended in theory to support dynamic collaboration by the use of Verified and Pro-active Secret Sharing.*

One of the most limiting factors in our models is the fact that storage space and bandwidth requirements increase linearly with the number of shares, this limiting the flexibility in increasing the number of providers to spread the data across more servers.

A possible solution is a trade-off between security and increased number of shares without increasing the storage and bandwidth requirements through the use of computationally secure secret sharing schemes instead of information theoretically secure schemes. The difference is

[1]least-recently used: the file that was least recently used will be evicted when the queue is full

that the latter schemes guarantee that no information about the secret can be learned unless the adversary has k shares, where k is the threshold of the scheme. Computationally secure schemes compromise in this by generating shares that are smaller than the original file, but leak information once m shares have been collected by the adversary, where m is smaller than k.

Such a scheme is more flexible in adding more providers and more shares with smaller file size at the cost of security. Adapted distribution schemes could take this into account and try to find an optimal distribution where k is an additional factor in determining how best to distribute shares among providers and collaboration sets.

The distribution algorithms only consider the trust of a provider or the trust in a collaboration set in addition to the general secret sharing threshold and the number of shares already present on one location. It would be desirable to enable the user to express preferences regarding providers, security, availability and other properties in a more fine-grained manner than a single value.

Another aspect of the distribution algorithm with room for improvement is the relative impact of shares with respect to the size of the parameters of the threshold scheme. For example, it should make a difference in weighting when putting a second share on a provider in a (3,5) and a (100,10000) scheme, the decision should not necessarily be the same even for the initial shares being distributed.

Another interesting area of research is to extend the models from a pure storage distribution mechanism to a composition mechanism for business processes/modeling work flows and how these interact. For example consider a multi-step business process that involves storing and retrieving data, then modifying or mining and transforming the data and return new information to the user. this involves several cloud components and we could model collaboration sets between data providers and compute providers and which may work together to form such a process and should not work together in order to prevent them from learning important aspects of the users business or business strategies.

The final open issue briefly introduced in Model 3 in Chapter 4 is dynamic collaboration. We have already covered this feature in the comparison between our model and social secret sharing in Section 6.1 and will therefore not repeat the arguments for or against it here.

CHAPTER 7

Conclusion

In the introduction we have outlined the challenges with data outsourcing to the cloud. Privacy concerns, lack of control as well as quality of service and availability pose risks to individual users and enterprises when adopting cloud services.

The primary challenge we tried to solve was to store data in the cloud such that the cloud provider or multiple cloud providers working together cannot learn the contents of the data and at the same time to increase availability of the data in case of outages.

In order to solve this problem we evaluated existing research in this area, specifically Social Secret Sharing (see Chapter 3, and Secret Sharing libraries in regard to performance and features (see Appendix A).

With the promising results of this evaluation we were able to design models for our single user cloud data storage with features such as preference of providers based on trust. We further introduced the concept of collaboration sets to describe how multiple providers may be bound by law for instance to reveal data and how we can solve this issue by taking this into account in our distribution strategies.

The result is a framework that works well for single users, who wish to store important files in the cloud without the providers learning the contents of the files. For example, it is easy to use in scenarios with three to five providers and does not require any metadata storage to keep track of how files are stored. It is sufficient for the user to remember the threshold of the scheme in case of a fresh install of the system.

APPENDIX A

Evaluation of Secret Sharing Libraries

A.1 Evaluation of Secret Sharing Libraries

In order to evaluate the feasibility of using Secret Sharing to increase security and availability in a cloud storage setting, we have gathered a comprehensive collection of libraries that support secret sharing for comparison:

Name	Type	Language/Platform	Share Size	Other Features
Secret Sharing in Java	GUI	Java	384-bit prime allows 48 characters of secret string	only supports Text and Numeric secrets
Secret Sharing for Windows	GUI	Windows, Qt/C++	no restriction on file size	good performance, calculates shares in memory
ssss	console-based application	C++, multi-platform	limited to 1kb secrets	good performance
GFShare	console-based application	C, multi-platform	no restriction	uses files as input and output, weak random number generator by default(replaceable)
SecretSharp	GUI	C#, Windows	unknown	supports only text

Table A.1: Advantages and Disadvantages compared to Social Secret Sharing.

Libraries available on the web: The aforementioned libraries can be obtained on the Internet at the following locations:

- Secret Sharing in Java: `http://sourceforge.net/projects/secretsharejava/`
- Secret Sharing for Windows : `http://sourceforge.net/projects/secretsharing/`
- ssss: `http://point-at-infinity.org/ssss/`
- GFShare: `http://www.digital-scurf.org/software/libgfshare`
- SecretSharp: `http://www.digital-scurf.org/software/libgfshare`

Other Implementations & Summary

Most of the tools that are available are limited in their scope to sharing text or numbers. Two of the tools show great promise to be used in our approach, because they support files and are not severely limited in terms of secret size.

We chose GFShare because it seems a natural fit as it is console-based and can therefore be used easily in an automated fashion. It would also be possible to fork the other implementation and remove the GUI and automate the process, but this is more work and not necessary as the performance of gfshare is sufficient for our use-cases.

A.2 Detailed Performance Evaluation of GFShare

The performance numbers were recorded using an Ubuntu-based 64-bit Linux distribution with the following hardware specification:

- Intel i5-2500K, 3.3 Ghz
- 8gb RAM
- 7200rpm harddrive

The measurements were taken using the *time command* and represent the real elapsed time in seconds of the program execution, because the library does not by itself support tracking these metrics. That is the reason why some execution times are shown as 0.0s, because the execution time was below the threshold of precision of the time command. Since we are only really interested in the cases where the SS split and combine phases would limit the usability by slowing the process this fact can be neglected.

As can be seen in Figure A.1 the performance of GFShare is satisfactory not only for small text-document-sized files, where the split times remain well under 1 second, but also for larger binary files in the range of three to five megabytes with wider threshold schemes than were deemed practical in our models and use-cases due to the increased storage and bandwidth requirements.

This shows that Secret Sharing libraries do not pose limitations in encryption and decryption times, only storage and bandwidth limitations have to be considered for practical purposes in our models.

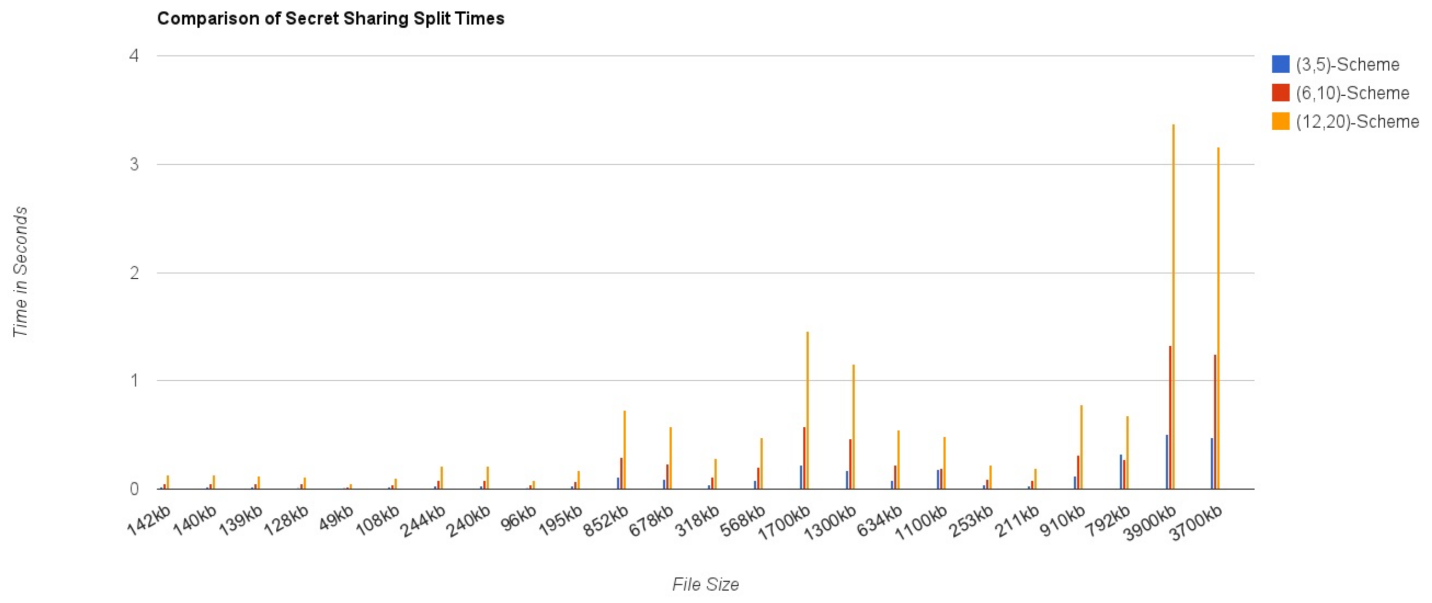


Figure A.1: Performance and Stuff

APPENDIX B

Code Samples

B.1 Decision Procedures and Test Runs

Listing B.1: Complete code samples for Figures 4.4,4.5 and 4.6

```java
package com.scs.core;

import com.google.common.base.Function;
import com.google.common.base.Predicate;
import com.google.common.collect.Iterables;
import com.google.common.collect.Lists;
import com.google.common.collect.Ordering;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 *
 * @author Martin Kirchner
 */
public class Distribution {

  public static void main(String args[]) {
    Distribution d = new Distribution();
    d.start(3, 5);
  }
  public void start(final int t, final int n) {
    List<Provider> providers = Arrays.asList(
            new Provider("P1", 0.8), new Provider("P2", 0.5),
            new Provider("P3", 0.7), new Provider("P3", 0.2))
               ;
    List<String> shares = Arrays.asList("Share 1", "Share 2",
        "Share 3", "Share 4", "Share 5");

    Map<String, String> map = distribute(t, providers, shares
       );

    System.out.println("####################################"
       );
    System.out.println("No Collaboration sets. Fig. 4.4");
    System.out.println("####################################"
       );
    for(String s : Ordering.natural().sortedCopy(map.keySet()
       )) {
      System.out.println(s + " has been added to Provider " +
          map.get(s));
    }


    CollaborationSet s1 = new CollaborationSet("CS1", 0.8);
    CollaborationSet s2 = new CollaborationSet("CS2", 0.3);
    CollaborationSet s3 = new CollaborationSet("CS3", 0.6);

    Provider p1 = new Provider("P1", 0.8);
    Provider p2 = new Provider("P2", 0.6);
    Provider p3 = new Provider("P3", 0.4);
    Provider p4 = new Provider("P4", 0.9);
    Provider p5 = new Provider("P5", 0.3);

    s1.providers.add(p1); p1.collabSet = s1;
    s1.providers.add(p2); p2.collabSet = s1;
    s2.providers.add(p3); p3.collabSet = s2;
    s2.providers.add(p4); p4.collabSet = s2;
    s3.providers.add(p5); p5.collabSet = s3;

    List<Provider> cproviders = Arrays.asList(p1, p2, p3, p4,
        p5);

    Map<String, String> cmap = distributeCollab(t, cproviders
       , shares);
```

```java

    System.out.println("####################################");
    System.out.println("With Collaboration sets. Fig. 4.5");
    System.out.println("####################################");
    for(String s : Ordering.natural().sortedCopy(cmap.keySet())) {
      System.out.println(s + " has been added to Provider " + cmap.get(s));
    }

    for(Provider p : cproviders) p.shares.clear();

    Map<String, String> ncmap = distribute(t, cproviders, shares);

    System.out.println("####################################");
    System.out.println("No Collaboration sets. Fig. 4.6");
    System.out.println("####################################");
    for(String s : Ordering.natural().sortedCopy(ncmap.keySet())) {
      System.out.println(s + " has been added to Provider " + ncmap.get(s));
    }
  }

  public Map<String, String> distribute(final int t, List<Provider> ps, List<String> sr) {
    Map<String, String> map = new HashMap<String, String>();
    for (String s : sr) {
      Iterable<Provider> threshold = Iterables.filter(ps, new Predicate<Provider>() {
        @Override
        public boolean apply(Provider p) {
          return p.getNumberofShares() < t - 1;
        }
      });
      Ordering<Provider> primary = Ordering.natural().reverse().onResultOf(new Function<Provider, Double>() {
        @Override
        public Double apply(Provider p) {
          return p.getNextRelativeTrust();
        }
      });
      Ordering<Provider> secondary = Ordering.natural().reverse().onResultOf(new Function<Provider, Double>() {
        @Override
        public Double apply(Provider p) {
          return p.getTrust();
        }
      });
      List<Provider> compound = primary.compound(secondary).immutableSortedCopy(threshold);
      Provider p = compound.get(0);
      p.shares.add(s);
      map.put(s, p.name);
    }
    return map;
  }
  public Map<String, String> distributeCollab(final int t, List<Provider> ps, List<String> sr) {
    Map<String, String> map = new HashMap<String, String>();
    for (String s : sr) {
      Iterable<Provider> threshold = Iterables.filter(ps, new Predicate<Provider>() {
        @Override
        public boolean apply(Provider p) {
          return p.getNumberofShares() < t - 1;
        }
      });
      Ordering<Provider> primary = Ordering.natural().reverse().onResultOf(new Function<Provider, Double>() {
        @Override
        public Double apply(Provider p) {
          return p.getNextCollabRelativeTrust();
        }
      });
      Ordering<Provider> secondary = Ordering.natural().reverse().onResultOf(new Function<Provider, Double>() {
        @Override
        public Double apply(Provider p) {
```

```java
        return p.getNextRelativeTrust();
      }
    });
    Ordering<Provider> tertiary = Ordering.natural().
       reverse().onResultOf(new Function<Provider, Double
       >() {
      @Override
      public Double apply(Provider p) {
        return p.getTrust();
      }
    });
    List<Provider> compound = primary.compound(secondary).
       compound(tertiary).immutableSortedCopy(threshold);
    Provider p = compound.get(0);
    p.shares.add(s);
    map.put(s, p.name);
  }
  return map;
}

public class Provider {
  public String name;
  public double trust;
  public List<String> shares = new ArrayList<String>();
  public CollaborationSet collabSet;

  public Provider(String name, double trust) {
    this.name = name;
    this.trust = trust;
  }
  public int getNumberofShares() {
    return shares.size();
  }
  public double getTrust() {
    return trust;
  }
  public double getNextRelativeTrust() {
    return getTrust() / (getNumberofShares() + 1);
  }
  public double getNextCollabRelativeTrust() {
    if (collabSet == null || (collabSet.
       getNumberofProvidersWithShares()) <= 0 || (collabSet
       .getProvidersWithShares().contains(this) &&
           collabSet.getNumberofProvidersWithShares() < 2))
          return getNextRelativeTrust();
        else
        {
          int numProviders = collabSet.getProvidersWithShares()
             .contains(this) ? collabSet.
             getNumberofProvidersWithShares(): collabSet.
             getNumberofProvidersWithShares()+ 1;
          return collabSet.trust / numProviders *
             getNextRelativeTrust();
        }
      }
    }

    public class CollaborationSet {
      public String name;
      public double trust;
      public List<Provider> providers = new ArrayList<Provider
         >();

      public CollaborationSet(String name, double trust) {
        this.name = name;
        this.trust = trust;
      }
      public int getNumberofProviders() {
        return providers.size();
      }
      public int getNumberofProvidersWithShares() {
        Iterable<Provider> withShares = Iterables.filter(
           providers, new Predicate<Provider>() {
          @Override
          public boolean apply(Provider p) {
            return p.getNumberofShares() >= 1;
          }
        });
        return Lists.newArrayList(withShares).size();
      }
      public List<Provider> getProvidersWithShares() {
        Iterable<Provider> withShares = Iterables.filter(
           providers, new Predicate<Provider>() {
          @Override
          public boolean apply(Provider p) {
            return p.getNumberofShares() >= 1;
            }
        });
        return Lists.newArrayList(withShares);
      }
    }
}
```

Listing B.1: Complete code samples for Figures 4.4,4.5 and 4.6

Results and comparison

As can be seen in the output of the test program that the results are identical to the expected results from Chapter 4.

Listing B.2: Output from the program

```
####################################
No Collaboration sets. Fig. 4.4
####################################
Share 1 has been added to Provider P1
Share 2 has been added to Provider P3
Share 3 has been added to Provider P2
Share 4 has been added to Provider P1
Share 5 has been added to Provider P3
####################################
With Collaboration sets. Fig. 4.5
####################################
Share 1 has been added to Provider P4
Share 2 has been added to Provider P1
Share 3 has been added to Provider P4
Share 4 has been added to Provider P1
Share 5 has been added to Provider P5
####################################
No Collaboration sets. Fig. 4.6
####################################
Share 1 has been added to Provider P4
Share 2 has been added to Provider P1
Share 3 has been added to Provider P2
Share 4 has been added to Provider P4
Share 5 has been added to Provider P1
```

Listing B.2: Output from the program

B.2 Additional Code Samples Related to Cloud Providers

Listing B.3: Code for uploading a file to the cloud using jClouds.

```java
public void SaveFileToBucket(File f) {
  try {
    context = ContextBuilder.newBuilder("aws-s3")
            .credentials(accessKeyId, secretKey)
            .buildView(AWSS3BlobStoreContext.class);


    BlobStore blobStore = context.getBlobStore();
    blobStore.createContainerInLocation(null, container);
    Blob b = blobStore.blobBuilder(f.getName()).payload(f).
       build();
    blobStore.putBlob(container, b, PutOptions.Builder.
       multipart());

    context.close();
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

Listing B.3: Code for uploading a file to the cloud using jClouds.

Bibliography

[1] Mohammed A AlZain, Ben Soh, and Eric Pardede. A new approach using redundancy technique to improve security in cloud computing. In *Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), 2012 International Conference on*, pages 230–235. IEEE, 2012.

[2] Mohammed Abdullatif ALzain and Eric Pardede. Using multi shares for ensuring privacy in database-as-a-service. In *System Sciences (HICSS), 2011 44th Hawaii International Conference on*, pages 1–9. IEEE, 2011.

[3] Mohammed Abdullatif AlZain, Eric Pardede, Ben Soh, and James A Thom. Cloud computing security: from single to multi-clouds. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 5490–5499. IEEE, 2012.

[4] Mohammed Abdullatif AlZain, Ben Soh, and Eric Pardede. Mcdb: Using multi-clouds to ensure security in cloud computing. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 784–791. IEEE, 2011.

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[6] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1988.

[7] David Bernstein and Deepak Vij. Intercloud security considerations. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 537–544. IEEE, 2010.

[8] GR Blakley. Safeguarding cryptographic keys. In *Proc. AFIPS 1979, National Computer Conference*, volume 48, pages 313–317, 1979.

[9] J Bohli, Nils Gruschka, Meiko Jensen, L Lo Iacono, and Ninja Marnau. Security and privacy enhancing multi-cloud architectures. 2013.

[10] Richard Brinkman, Jeroen Doumen, and Willem Jonker. *Using secret sharing for searching in encrypted data*. Springer, 2004.

[11] Richard Brinkman, Ling Feng, Jeroen Doumen, Pieter H Hartel, and Willem Jonker. Efficient tree search in encrypted data. 2004.

[12] Richard Brinkman, Berry Schoenmakers, Jeroen Doumen, and Willem Jonker. *Experiments with queries over encrypted data using secret sharing*. Springer, 2005.

[13] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Algorithms and architectures for parallel processing*, pages 13–31. Springer, 2010.

[14] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[15] Christian Cachin, Robert Haas, and Marko Vukolic. Dependable storage in the intercloud. *IBM Research*, 3783:1–6, 2010.

[16] Antonio Celesti, Francesco Tusa, Massimo Villari, and Antonio Puliafito. How to enhance cloud architectures to enable cross-federation. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 337–345. IEEE, 2010.

[17] Cloud Competence Center. What are deployment models in cloud computing? (http://www.cloud-competence-center.com/understanding/cloud-computing-deployment-models/).

[18] Yanpei Chen, Vern Paxson, and Randy H Katz. What's new about cloud computing security. *University of California, Berkeley Report No. UCB/EECS-2010-5 January*, 20(2010):2010–5, 2010.

[19] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Foundations of Computer Science, 1985., 26th Annual Symposium on*, pages 383–395. IEEE, 1985.

[20] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 85–90. ACM, 2009.

[21] Mache Creeger. Cloud computing: An overview. *ACM Queue*, 7(5):2, 2009.

[22] Whitfield Diffie and Martin Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[23] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: Issues and challenges. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 27–33. Ieee, 2010.

[24] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 427–438. IEEE, 1987.

[25] Ana Juan Ferrer, Francisco Hernández, Johan Tordsson, Erik Elmroth, Ahmed Ali-Eldin, Csilla Zsigri, Raül Sirvent, Jordi Guitart, Rosa M Badia, Karim Djemame, et al. Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66–77, 2012.

[26] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[27] Nikolay Grozev and Rajkumar Buyya. Inter-cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*, 2012.

[28] Darrel Hankerson, Scott Vanstone, and Alfred J Menezes. *Guide to elliptic curve cryptography*. Springer, 2004.

[29] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Advances in Cryptology CRYPTO 95*, pages 339–352. Springer, 1995.

[30] Martin Gilje Jaatun, Gansen Zhao, Athanasios V Vasilakos, Åsmund Ahlmann Nyre, Stian Alapnes, and Yong Tang. The design of a redundant array of independent net-storages for improved confidentiality in cloud computing. *Journal of Cloud Computing*, 1(1):1–19, 2012.

[31] Meiko Jensen, Jörg Schwenk, J Bohli, Nils Gruschka, and Luigi Lo Iacono. Security prospects through cloud computing by adopting multiple clouds. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 565–572. IEEE, 2011.

[32] Ari Juels and Alina Oprea. New approaches to security and availability for cloud data. *Communications of the ACM*, 56(2):64–73, 2013.

[33] Ben Kepes. Understanding the cloud computing stack: Saas, paas, iaas, 2011. `http://www.rackspace.com/knowledge_center/whitepaper/understanding-the-cloud-computing-stack-saas-paas-iaas`.

[34] Tobias Kurze, Markus Klems, David Bermbach, Alexander Lenk, Stefan Tai, and Marcel Kunze. Cloud federation. In *CLOUD COMPUTING 2011, The Second International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 32–38, 2011.

[35] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm. What's inside the cloud? an architectural map of the cloud landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 23–31. IEEE Computer Society, 2009.

[36] Ping Lin and K Selçuk Candan. Secure and privacy preserving outsourcing of tree structured data. In *Secure Data Management*, pages 1–17. Springer, 2004.

[37] Joseph McKendrick. Closing the security gap: 2012 ioug enterprise data security survey. Technical report, Unisphere Research, Sponsored by Oracle, 2012.

[38] Peter Mell and Timothy Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800(145):7, 2011.

[39] Ventzislav Nikov and Svetla Nikova. On proactive secret sharing schemes. In *Selected Areas in Cryptography*, pages 308–325. Springer, 2005.

[40] Mehrdad Nojoumian and Douglas R Stinson. Social secret sharing in cloud computing using a new trust function. In *Privacy, Security and Trust (PST), 2012 Tenth Annual International Conference on*, pages 161–167. IEEE, 2012.

[41] Mehrdad Nojoumian, Douglas R Stinson, and Morgan Grainger. Unconditionally secure social secret sharing scheme. *Information Security, IET*, 4(4):202–211, 2010.

[42] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology CRYPTO 91*, pages 129–140. Springer, 1992.

[43] Dana Petcu. Multi-cloud: Expectations and current approaches. In *Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds*, MultiCloud '13, pages 1–6, New York, NY, USA, 2013. ACM.

[44] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85. ACM, 1989.

[45] S Ramgovind, Mariki M Eloff, and E Smith. The management of security in cloud computing. In *Information Security for South Africa (ISSA), 2010*, pages 1–7. IEEE, 2010.

[46] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212. ACM, 2009.

[47] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[48] Yashaswi Singh, Farah Kandah, and Weiyi Zhang. A secured cost-effective multi-cloud storage in cloud computing. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 619–624. IEEE, 2011.

[49] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.

[50] Douglas R Stinson. *Cryptography: theory and practice*, volume 36. CRC press, 2006.

[51] S Subashini and V Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34(1):1–11, 2011.

[52] Toby Velte, Anthony Velte, and Robert Elsenpeter. *Cloud computing, a practical approach*. McGraw-Hill, Inc., 2009.

[53] Marko Vukolić. The byzantine empire in the intercloud. *ACM SIGACT News*, 41(3):105–111, 2010.

[54] Guojun Wang, Qin Liu, and Jie Wu. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 735–737. ACM, 2010.

[55] Weichao Wang, Zhiwei Li, Rodney Owens, and Bharat Bhargava. Secure and efficient access to outsourced data. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 55–66. ACM, 2009.

[56] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[57] Gansen Zhao, Chunming Rong, Jin Li, Feng Zhang, and Yong Tang. Trusted data sharing over untrusted cloud storage providers. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 97–103. IEEE, 2010.