João Miguel Cardia Melro Rodrigues

Licenciado em Engenharia Informática

TSKY: A Dependable Middleware Solution for Data Privacy using Public Storage Clouds

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador:	Doutor Henrique João Domingos, Prof. Auxiliar, Universidade Nova Lisboa

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2013

TSKY: A Dependable Middleware Solution for Data Privacy using Public Storage Clouds

Agradecimentos

Desde já agradeço à Faculdade de Ciências e Tecnologia de Universidade Nova de Lisboa, mais especificamente ao Departamento de Informática e a todo o corpo docente, não docente e a todos os colegas pelo apoio prestado no decorrer dos vários anos lectivos e em especial durante a fase de preparação e elaboração de dissertação período durante o qual me foi atribuída uma bolsa de investigação. Ainda mais, agradeço ao meu orientador, Henrique João Lopes Domingos, pelo apoio prestado relativo não só á dissertação e ao projeto TSKY, como agradeço todo o apoio prestado ao nível académico.

Finalmente, e não menos importante, agradeço a todos os meus amigos, familiares e conhecidos pelo apoio prestado ao longo dos últimos anos, em que fui aluno da FCT.

Resumo

A presente dissertação visa tirar partido das virtudes oferecidas pelos sistemas de armazenamento de dados em nuvem, disponibilizados por diferentes provedores na Internet. A solução proposta visa o incremento de garantias de segurança, disponibilidade e fiabilidade, através da utilização combinada de vários provedores, numa visão de armazenamento e computação em nuvem-de-nuvens. O sistema de *middleware*, designado por TSKY (*Trusted Sky*), pretende constituir uma solução confiável, para intermediação entre aplicações finais e nuvens de armazenamento de vários provedores. Para isso o sistema combina, na sua arquitetura, componentes e serviços para indexação, distribuição e replicação de dados em múltiplas nuvens de armazenamento, tendo em vista assegurar condições de resiliência perante falhas ou ataques que podem ocorrer de forma independente ao nível do provedor. Para garantir as condições de segurança são utilizados vários mecanismos criptográficos para cifra e autenticação de dados, com base em esquema de partilha de segredos (*secret sharing schemes*) e assinaturas de limiar (*threshold signatures*). Para concretizar e avaliar o sistema TSKY, como sistema genérico, foi criado um repositório confiável de correio electrónico, designando por TSKY-TMS (Trusted Mail System). Este sistema constitui um protótipo que utiliza os serviços base do *middleware* TSKY para armazenamento de mensagens numa tipologia de nuvem-de-nuvens.

Palavras-chave: Computação em Nuvem, Armazenamento em Nuvem, Esquemas e Algoritmos de Criptografia de Limiar, Esquemas de Encriptação Holomórfica, Privacidade, Fiabilidade e Disponibilidade dos Dados.

Abstract

This dissertation aims to take advantage of the virtues offered by data storage cloud based systems on the Internet, proposing a solution that avoids security issues by combining different providers' solutions in a vision of a cloud-of-clouds storage and computing. The solution, TSKY System (or Trusted Sky), is implemented as a middleware system, featuring a set of components designed to establish and to enhance conditions for security, privacy, reliability and availability of data, with these conditions being secured and verifiable by the end-user, independently of each provider. These components, implement cryptographic tools, including threshold and homomorphic cryptographic schemes, combined with encryption, replication, and dynamic indexing mechanisms. The solution allows data management and distribution functions over data kept in different storage clouds, not necessarily trusted, improving and ensuring resilience and security guarantees against Byzantine faults and attacks. The generic approach of the TSKY system model and its implemented services are evaluated in the context of a Trusted Email Repository System (TSKY-TMS System). The TSKY-TMS system is a prototype that uses the base TSKY middleware services to store mailboxes and email Messages in a cloud-of-clouds.

Keywords: Cloud Computing, Cloud Storage Services, Threshold Cryptographic Schemes and Algorithms, Homomorphic Encryption Schemes, Data Privacy, Reliability and Availability

Table of Contents

List of Figures

List of Tables

1

Introduction

1.1. Context and Motivation

In the current globalization context, it has become increasingly important to securely manage data regarding the internal operations of enterprises and their clients. Given the inevitable generation and exchange of new data, with an increasing tendency, and with the growing of the use of the Internet as a global business platform [1], the use of cloud based solutions has become even more appealing. This kind of solutions (cloud based solutions) offer multiple advantages compared with traditional solutions based on localized data centers and infrastructures maintained by own companies [2]:

- **Pay-per-use cost model**, in which the clients pay just for the resources in use, reducing the infrastructure allocation and update costs as the need for more resources increase;
- **Ubiquity**, enabled by the public cloud access, end-user applications and middleware systems can access computational resources from any location without relying on resources real location. Furthermore the use of public cloud services enables the combination of different data centers (in different locations) from the same or from different providers in order to achieve better reliability and availability for the target solutions;
- **Scalability and Elasticity**, the on-demand resources provided in multitenant systems, allow the adaptation of resources like memory, available storage or computational power, accordingly to system loads or necessities;

- **Maintenance costs**, while providing maximum control, local infrastructures carry total ownership costs [3] associated the cost of building the infrastructure itself along with maintenance costs. In multitenant public cloud solutions ownership and maintenance costs are omitted and the actual costs are easily managed as a pay-per-use cost model demands;
- **Security**, although not auditable cloud based solutions offer interesting reliability and availability conditions that could be explored by existing or new solutions.

1.2. Cloud as a Service

Recently, the number of cloud services providers has substantially increased [4] which has promoted the competitiveness in those markets and consequently lowering the prices. These providers, offering different cost models to different targets (private or corporative customers), divide the provided services in three different standard categories:

- **Infrastructures as a Service** or IaaS – consisting in providing pure or virtualized computational resources without outbox features;
- **Platform as a Service** or PaaS – in which resources, as databases, key-value storage services or content delivery networks, are provided as a platform to new or existing systems and applications;
- **Software as a Service** or SaaS – in which the provided services consist in out-of-the-box server software (e.g. email, messaging, file storage, CMS).

The proposed solution is introduced as a Middleware as a Service in which a key-value storage API is provided as a middleware service offering data confidentiality, integrity and availability services. This API is then explored by an email repository service used as validator to the proposed middleware solution. The multiple categories of services including the MaaS, are leveraged by a set of advantages, as stated before:

- **Elasticity** – provided by public cloud services, the resources are dynamically adjusted accordingly a set of well-defined needs (e.g. number of accesses per time period, data volume) or manually as customer demands;
- **Scalability and resources Relocation** – Normally application supported, scalability and relocation can be provided through data center virtualization

techniques. In the proposed solution this scalability can be provided through the use of multiple dynamic instances offering both scalability and relocation and consequently ubiquitous capabilities;

- **Costs** – The cost model, of the multiple cloud services are considered attractive due to the lack of ownership costs associated to traditional infrastructures that carry establishment and maintenance costs. Furthermore the use of private infrastructures carries constant energetic costs which can exceed the infrastructure effective profit.

In contrast to the proposed solution, the standard provided services are based on a set of non-auditable promises regarding durability, integrity, availability and confidentiality, defined in Service Level Agreements describing the maximum time impact of future unpredicted events. Furthermore the presented solution is intended to run in a trusted environment, namely in a local trustable proxy or locally in the users' computers.

1.3. Security Issues of data stored in public clouds

As previously described, the use of cloud storage repositories on the Internet leverages a set of operational advantages, the analysis of such advantages raises an interest in the use of such services as storage platform to existing applications, however the adoption of such providers reveals itself problematic when certain requirements are considered [19]. Requirements as maintenance and management with confidentiality and privacy guarantees or permanent user control and auditability are hard or even impossible to ensure. In fact, the use of such cloud based services as outsourced data management solutions are itself a still open problem [5]. These issues, widely discussed, are introduced by the following question: *"How to keep public data secure?"*, furthermore, various entities [20], [21] have analyzed these issues inspiring new research fields in computer systems security and cloud computing. For example in health field the adoption of cloud storage solutions requires a confidential and decentralized management of user's persistent medical data [22]. Moreover, the medical data must be kept available and uncorrupted at any time and if deleted, in must be in a permanent way.

Beyond users control and auditability over security agreements, the use of outsourced solutions raises issues related with the prices unpredictability and data migration costs. These issues occur when large data volumes are stored in a provider premises and the service associated costs are substantially raised, which introduces a dilemma: keep the data in providers' premises accepting the new cost model or do an expensive data migration to other provider. This sort of distress is called vendor lock-in.

Recent events have exposed multiple security vulnerabilities in cloud platforms, multiple reported cases [23], [24], [25], [26] shown that for multiple reasons the data stored on such platforms were disclosed, destroyed or corrupted accidentally or on purpose by identified or unidentified third parties exposing private data and so violating basic security properties as confidentiality, integrity and data availability leading to huge financial losses. These security related issues become more pronounced when critical data and services are considered. Almost all society sectors rely on critical applications and services. E.g.:

- Applications that manage and handle users' medical files and history involving patient-doctor confidentiality;
- Email messaging services, that handle critical messages from multiple users in corporative or personal contexts involving financial, legal, moral or even ethical matters;
- Applications that manage private banking data subject to strict terms and conditions in which the clients accounting data should not be disclosed under any circumstances;
- Any application that handles user history and that for legal or ethical reasons should guaranty that the stored data will only be accessible by trusted and authenticated parties;
- Applications that support resource management to armed forces in national security matters.

This thesis presents a middleware concept that act between final applications or services and the cloud storage providers offering reliability, availability integrity, confidentiality and indexing services to applications.

The above concerns have been extensively referred in the literature, as well as, in relevant reports published by different institutions [5] or results addressed in relevant research projects [6], [7].

The combination of mechanisms and services to deal with security, reliability and availability properties, conjugated in dependable and trusted solutions to manage outsourced data in public storage clouds, is a challenging research direction. The support for those properties with the addition of independent auditing control guarantees by the users is also a timely and innovative research contribution. Recently, the research community has been addressing these challenges, discussing the relevance and combination of different dimensions of the problem. Some approaches are particularly concerned in improving dependability services as requirements of cloud computing infrastructures and cloud data-centers [8], [9], [10]. Other approaches are more concerned in designing middleware services, implementing trust computer bases, to intermediate the use of untrusted public clouds, as they are provided today, but adding dependability guarantees under the user control [11], [12], [13], [14].

The approach of the present dissertation follows the latter approaches, considering the adoption of well-known and well-established cloud storage solution providers, such as: Amazon [15], Google [16], Nirvanix [17] or Rackspace [18] and the way to explore such solutions in a secure, reliable and high-available way. The main focus is the design, implementation and validation of a middleware solution, used as a trusted computing base under the user control, to provide a set of security and reliability services, independently of native dependability mechanisms supported or not by the cloud-service providers in their data-centers.

Regardless of the dissertation focus, the combination of security and reliability mechanisms and services, complementarily addressed with other approaches to improve native dependability services on cloud-based infrastructures, will be certainly a way to provide an integrated and global solution for security, privacy, reliability and availability, to support dependable *end-to-end* arguments in cloud computing systems and applications design.

1.4. Thesis problem statement

The dissertation problem is focused in finding possible answers and contributions to the following questions:

- Is it possible to build a middleware solution supporting a storage cloud-of-clouds approach to overcome the limitations of Internet public storage clouds in assuring reliability, availability and privacy guarantees for their users?
- How can we compose different mechanisms, in such middleware approach, combining reliability, availability and privacy-enhanced services by the use of appropriate cryptographic tools and replication strategies that can benefit from a cloud-of-clouds implicit diversity and heterogeneity?
- Is it possible to address this type of solutions, guaranteeing the secure deployment of critical applications while maintaining the user control and benefiting from the advantages of cloud-based outsourcing data management services?
- What kind of mechanisms and services are candidates to be conjugated in a generic middleware approach, to serve different types of applications?

In order to give response to the above questions, this dissertation addresses a system that provides multiple services attained by a set of components, algorithms and techniques combined in a middleware approach. In our approach the data-storage clouds are considered primarily as simple storage services, without the capacity of executing middleware's code. In the primarily approach (and in the addressed solution) we consider the clouds as not trusted, just being used to provide a data-integration support layer, accessed by their specific access interfaces or APIs, without modifications. The middleware itself was designed to execute as a local software library supporting applications in an end-user machine. However, the design concerns in the dissertation also address the possible evolution of the middleware abstraction, to run in remote trusted computing bases (as a proxy service implementing a transparent cloud-API abstraction to the remote clients). With this concern in mind, the current middleware solution addresses some important issues namely the minimization of the trusted computing base abstractions, in order to address, a future work vision, of possible support of some middleware components executing in a public computing cloud. From this generic problem statement, we identify the objectives and contributions introduced in the next sections.

1.5. Objectives

To draw all the advantages provided by cloud based services and to address the problems associated to their use, the current thesis aims to conceive, implement, test and evaluate a trustable middleware system that takes advantage of the benefits offered by multiple cloud storage providers in a cloud-of-clouds approach. The presented middleware can be explored in multiple application contexts by means of an API similar to the APIs offered by standard cloud storage provider solutions (used as key-value cloud storage services). The proposed middleware framework implements a TCB system that enables the composition of a set of components leveraging data privacy, reliability and availability properties while ensures auditability mechanisms under the user control.

The proposed middleware framework can be composed in a variety of ways, with multiple modules such as: fragmentation, encryption, replication and indexing modules. The solution indexes, authenticates, encrypts and replicates data on multiple storage clouds ensuring data integrity, data privacy and key-management in safety conditions. At the same time, the solution provides high-availability and fault-tolerance guarantees, supporting independent cloud-service failures than can occur in each storage cloud used.

By using threshold-based cryptographic techniques and tools the security support is mapped into a byzantine fault-tolerant replication model, using multiple storage clouds as a data-storage backend. Furthermore, homomorphic techniques have been included in a way to enhance data privacy purposes, ensuring the protection of middleware-side indexing and searching operations over encrypted data. The implemented middleware framework, in a proof-of-concept prototype (named TSKY), supports an object storage solution provided as a key-value storage service. TSKY provides a generic object storage service and can be used as a backend storage system for different applications.

As proof of concept, and to test and evaluate the proposed solution, an email repository was designed and developed over the TSKY object storage system. This email repository system, named TSKY-TMS (Trusted Email System), was designed and implemented to support a set of standard protocols, namely POP3, SMTP and REST protocols. Through these endpoints any mail user agent application can send, store or retrieve electronic mail messages. Additionally, the TSKY-TMS system provides search operations over e-mail message headers (according to the RFC822 standard) and e-mail body contents (including some types of attachments). The TSKY-TMS repository service has enabled the validation of different evaluation metrics, to test reliability and performance conditions regarding not only the TSKY-TMS integrated solution as regarding the base TSKY framework and the cloud providers' storage services themselves.

1.6. Main Contributions

As a solution to the described problem it was intended to design and develop a middleware solution that acts as a proxy between the client and the cloud storage providers in order to accomplish the proposed objectives. The proposed solution is composed by mechanisms based on a set of security services. The developed mechanisms integrated in the proposed framework combined with replication make up the following main contributions:

Controlled Data Management, by the use of the provided interface, applications will be able to list, add, remove and search data in a permanent way. Furthermore since that, the data is stored encrypted in data storage providers and since the key cryptographic materials are locally stored, the data retrievability and interpretation can only be made by the middleware, ensuring data privacy and giving total user control;

Data Confidentiality, delivered by the use of symmetric cryptography complemented with use of secret sharing schemes over encryption keys. Besides of minimizing the key management overhead, secret sharing schemes allow data deployment in byzantine context, i.e., it is possible to successfully recover data with at least $3f+1$ replicas, where f is the number of corrupted or unavailable replicas. The presented middleware was tested using AES encryption and three distinct secret sharing schemes: Shamir [27], Asmuth-Bloom [28] and Blackley [29] secret sharing scheme, detailed on related work section;

Data Integrity and Availability, delivered by the use of threshold signatures, providing unforgeable data integrity proofs that, as in other threshold schemes, threshold signatures can be mapped into a byzantine model, complemented with replication techniques, the use of threshold signatures guarantees corruption free data recoverability. Alternatively MACs were used as base to performance test comparisons;

Confidential Data Indexing and Searching provided by the application components the developed indexes with homomorphic properties, leverage the system security guarantees minimizing the middleware's TCB while providing search operations over encrypted stored data at API level. The components providing such operations were designed using LSS and Paillier homomorphic schemes enabling ciphered keyword search and encrypted scoring updates respectively as defined in [30];

Cloud Storage Providers Individual Independence, the presented middleware solution, as described before, explores a cloud-of-clouds topology in which the middleware itself can be considered a cloud exploring a set of other clouds. Each and all cloud providers are explored and supported by a set of heterogenic adapters. Furthermore, the presented solution assumes a fail model in which each cloud can fail independently without compromising data confidentiality, availability or integrity.

The validation of the proposed system, which comprises all the above contributions, includes not only the evaluation of each of the components individually as include performance tests of the overall solution including over-cloud operations latencies. Furthermore, the above contributions included in the final solution and deployed using TSKY framework guarantee, in whole, the desired properties with main focus in data confidentiality, integrity, availability and system reliability. The overall solution desired properties are provided not only by the above contributions as are enforced by the cloud storage providers guarantees and mechanisms, i.e., the system desired properties are enforced by availability and durability guarantees defined in each providers SLA.

1.7. Contributions and achieved results

As a corollary of the current dissertation three main results were produced:

- **The TSKY-TMS system prototype**. The prototype is available for use, for possible demonstration and validation purposes, downloadable from the TSKY project site (http://asc.di.fct.unl.pt/TSKY);
- The presentation and publication of two articles introducing and discussing the TSKY-TMS design model and prototype implementation criteria, presenting also different assessment metrics for the system implementation analysis and validation:
    - J. Rodrigues, B. Ferreira, H. Domingos, "A Secure Email Repository Service using Public Untrusted Storage Clouds," in INForum Informatics Symposium, 2013, pp. 395-406
    - J. Rodrigues, B. Ferreira, H. Domingos, "TMS – A Trusted Mail Repository Service using Public Storage Clouds," in Eighth Workshop on Middleware for Next Generation Internet Computing, 2013, article no. 2

2

Related Work

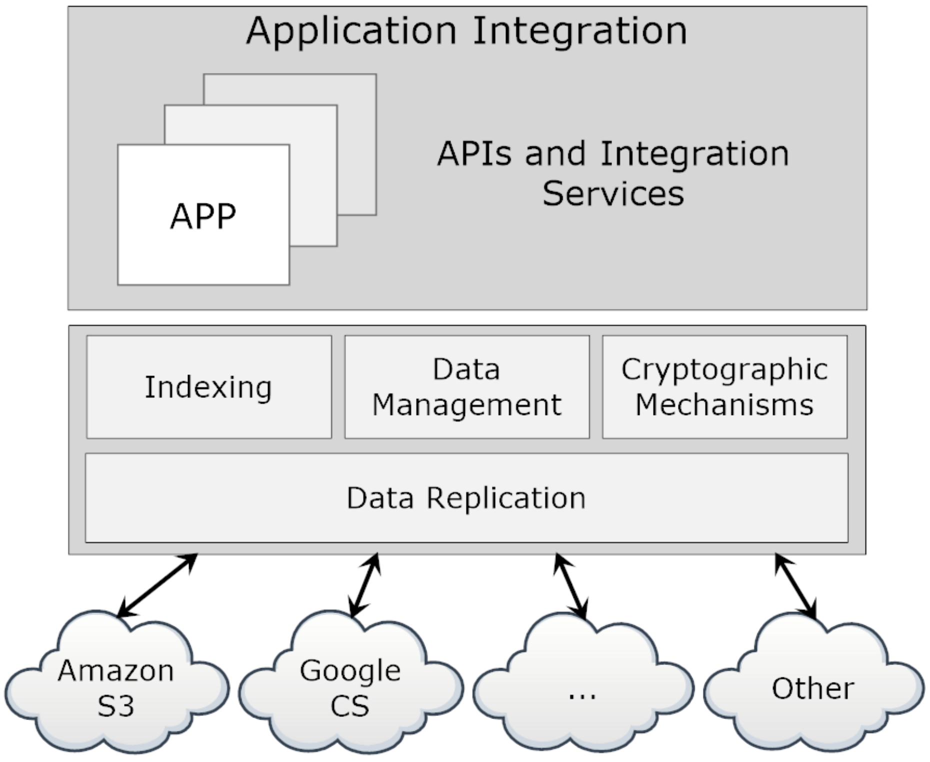


Figure 1: Main System Concerns

In Figure 1 we identify a set of important research directions related with components and services integrated in the architectural vision of the proposed system. The multiple components and the provided services motivate the study of relevant research work. Accordingly to Figure 1, in the next sections we will approach the studied related work references in the following areas:

Indexing – Although not as main focus, indexing techniques and services tend to be important to a set of applications we want to support. In the presence of large data sets indexing is key feature enabling the management of such data. One of the key issues, in this research direction, is how to extract actual information from large heterogeneous (different types/formats of data) datasets. Different techniques to address this issue, including multimedia information extraction, have been already addressed in [31]. In the context of development of this thesis, the indexing components are addressed using specific techniques for text-based information retrieval, namely simple linear searching schemes [32], [33] combined with ranking based indexing techniques, namely using the Okapi BM25 scoring scheme [34] for the purpose of validation of application scenario (email repository platform);

Data management services - In this direction, systems addressing the use of third party data-repositories not necessarily trustable for the applications using them are particularly interesting [13], [14]. More recently, approaches to dependable services in cloud-oriented middleware systems addressing security and reliability concerns, are important contributions inspiring the ideas for this thesis [11], [12].

Cryptographic Mechanisms and Tools – One of main issues addressed in this thesis regards data security, and so cryptographic mechanisms represent a big role in such objective. As research direction, we explored not only widely used encryption algorithms, as we explored some novel approaches using existing mechanisms, namely secret sharing schemes [27], [28], [29], threshold signatures [35] and cryptographic homomorphic schemes [30]. The solution we intended to create uses these mechanisms integrated in other research directions, namely integrated in indexing, data management and cloud replication;

Data Replication – Finally and also as a research direction, the use of replication mechanisms including Byzantine fault tolerant replication has been extensively studied in multiple contexts as [36] or [37]. Although could be addressed as future work, the use of fault tolerant replication algorithms is out of scope for the current thesis (and is addressed in the context of the TSKY project architecture in the context of other dissertations). Nevertheless, replication plays an important role in the designed solution regarding availability and reliability issues assumed as objectives of the solution. Furthermore, the use of replication using multiple clouds (in a cloud-of-cloud solution) enables new research directions concerning profiling and erasure codes techniques and also ubiquity and location awareness, all addressed as future work.

2.1. Cryptographic Mechanisms and Tools

2.1.1 Secret Sharing Schemes

To easily understand the problem behind the existence of such schemes, Shamir (in [27]) has enunciated the following problem:

"Eleven scientists are working on a secret project. They wish to lock up the documents in a cabinet so that the cabinet can be opened if and only if six or more of the scientists are present. What is the smallest number of locks needed? What is the smallest number of keys to the locks each scientist must carry?"

To solve such problem we could think in a combinatory solution where 252 keys per scientist and 462 locks are needed. This solution, not just is impractical for the above problem as, it is impractical and even infeasible in a computational solution where some data need to be encrypted and keep secure on threshold basis. So, we need a secret sharing scheme to solve such problem in a practical way.

A secret sharing scheme, also called secret splitting or *threshold(t,n)* scheme, refers to a method for distributing a secret amongst a group of participants, each whom is allocated a share of the secret. This secret can be any kind of numerical data, whereas any content can be converted to numerical data. The shared secret can be reconstructed only when a sufficient number, of possibly different types, of shares are combined together. This kind of schemes should held two essential properties:

- The knowledge of any t or more D_i pieces makes D easily computable;
- And the 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).

This kind of schemes held a huge advantage in the present dissertation, due to the replication (mapped into a Byzantine fault model) defined as an objective and due to the confidentiality that must be held. Although useful this kind of schemes are based on numerical operations and so are pretty heavy regarding CPU processing. Some tests regarding the various scheme performances were made and can be seen in evaluation section.

2.1.1.1. Shamir Secret Sharing Scheme

Published in 1979, the Shamir Secret Scheme [27] is based on polynomial interpolation, this is, a set of planar points are generated in such way that the result of their interpolation is a function that held the secret. The Lagrange Interpolation Theorem used to interpolate the points states that given $n + 1$ data points $(x_0, y_0), \dots, (x_n, y_n)$ in the plane, with $x_i \neq x_j$, for $i \neq j$, there is a unique polynomial p of degree at most n that interpolates the given data, i.e.: such that $p(x_i) = y_i$ for $i = 0, \dots, n$.

In other words using a polynomial of degree $t - 1$ as the secret we share a set of n points as a set of shares of that secret. In order to recover the secret we will need at least t points to be able to recover the secret, i.e. we need t points to be able to build the original $t - 1$ function that interpolates all the points.

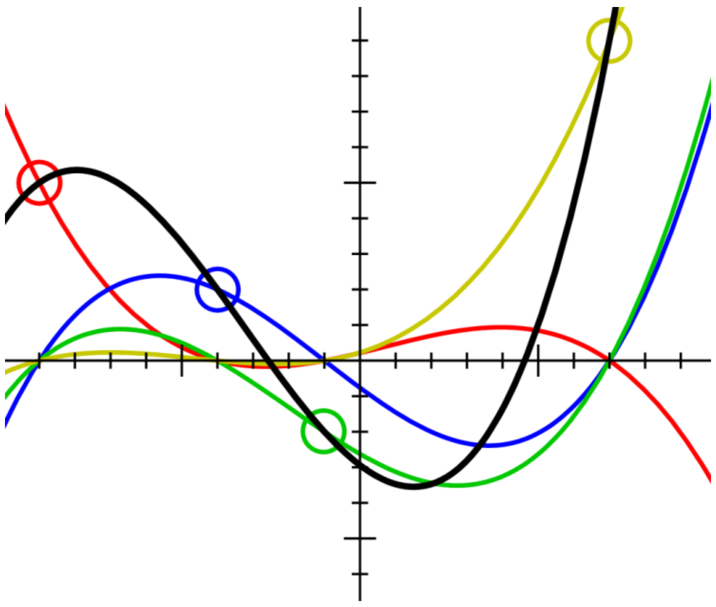


Figure 2: Interpolation Problem, 4 points, 3rd degree function

As seen in Figure 2, through Lagrange interpolation definition we verify that a set of 4 point (*n+1* points) determine uniquely the function of degree 5 (degree *n*) interpolating such points. This assumption is based on the existence of a solution and the uniqueness of such solution (found in [38]).

Algorithm

To share a secret D we start by randomly generate a polynomial of $k-1$ degree:

$$q(x) = a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}$$

Where:

$$q(0) = a_0 = D$$

We generate and distribute n distinct points $(x_1, y_1), \dots, (x_n, y_n) \in q$ modulus p such that $p > D$ and $p > n$. To recover the secret we need at least k points from q. For each of the k points we compute the Lagrange polynomials:

$$l_j = \sum_{i=1, j \neq i}^{k} \frac{x - x_i}{x_j - x_i}$$

Finally we recover the secret computing:

$$q(x) = \sum_{j=1}^{k} l_j(x) \cdot y_i$$

We get $q(0) = a_0 = D \bmod p$.

Practical Example

We want to share the secret $D = 4321$ amongst 6 people and we want at least 3 of them needed to recover the secret. We generate a function randomly with $a_0 = D$:

$$f(x) = 4321 + 192x + 105x^2$$

Next we select 6 distinct points from f modulus a prime $p = 4327$:

$$(1{,}291), (2{,}798), (3{,}1515), (4{,}2442), (5{,}3579), (6{,}599)$$

We distribute the shares among the players. To recover the secret we select randomly at least $k = 3$ points (shares):

$$(2{,}798), (4{,}2442), (5{,}3579)$$

Next, we compute Lagrange polynomials for each of the selected points:

$$l_2 = \frac{1}{6}x^2 - \frac{3}{2}x + \frac{10}{3}$$

$$l_4 = -\frac{1}{2}x^2 + \frac{7}{2}x - 5$$

$$l_5 = \frac{1}{3}x^2 - 2x + \frac{8}{3}$$

Finally, to recover the secret, we compute:

$$f(0) = 768 \cdot \frac{10}{3} + 2442 \cdot (-5) + 3579 \cdot \frac{8}{3} \bmod 4327 = 4321$$

In this thesis the people correspond to a set of heterogenic clouds and the secret we pretend to share correspond to the seed used to generate a cryptographic key or a set of keys which in turn are used to encrypt the stored data. Although considered perfect and the most efficient method to share a secret this method present an error as defined in Lagrange Theorem. This error doesn't occur frequently, even so, and to avoid secret corruption, the secret can be wrapped in some random data so the less significant data is discarded in the process.

2.1.1.2. Blakley Secret Sharing Scheme

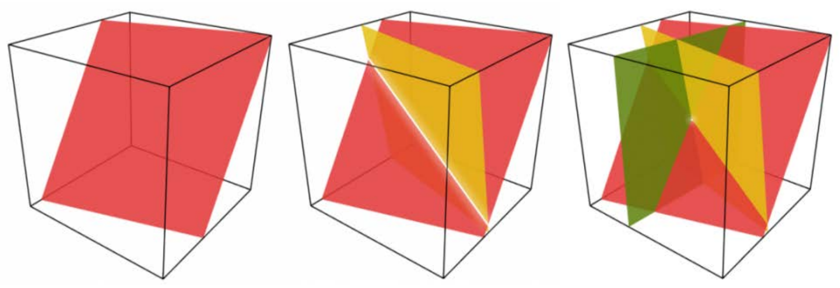


Figure 3: Three 3-Dimensional Plane Intersection

As with Shamir Scheme, this scheme was invented in 1979 but by George Blakley. The idea behind this scheme [29] is that we can create a hyperspace in such way that a subset t out of n planes can be intercepted in a point in that hyperspace. This point is the secret we want to share. In other words, each share corresponds to a plane and the secret held in the point of intersection of at least t planes as illustrated in Figure 3.

Algorithm

First we select a prime $p > D$ where D is the secret we want to share. We generate a random point $P = (P_1, \ \ldots, \ P_k)$ where $P_1 = \mathrm{D}$.

Now we need to generate n linearly independent ($det \neq 0$) *k-dimension* planes:

$$\begin{pmatrix} \alpha_{1,1} & \cdots & \alpha_{1,k-1} \\ \vdots & \ddots & \vdots \\ \alpha_{n,1} & \cdots & \alpha_{n,k-1} \end{pmatrix} : \forall_{i,j} \alpha_{ij} = \text{rnd mod } p$$

Finally we define the last monomial:

$$\alpha_{i,k} = P_k - \alpha_{i1}P_1 - \cdots - a_{ij}P_j (\text{mod } p), \ j \in [2, k], \ i \in [1, n]$$

Having at least k planes we can precede to the secret recovery. To do so we must solve the following congruences' system:

$$S = \begin{bmatrix} \alpha_{1,1} & \cdots & \alpha_{1,k-1} & (-1) \\ \vdots & \ddots & \vdots & \vdots \\ \alpha_{k,1} & \cdots & \alpha_{k,k-1} & (-1) \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_k \end{bmatrix} \equiv - \begin{bmatrix} \alpha_{1,k} \\ \vdots \\ \alpha_{k,k1} \end{bmatrix} (\text{mod } p) \rightarrow (P_1, \ldots, P_k) = (\beta_1, \ldots, \beta_k)$$

Note that if the randomly generated planes aren't linear independent they can null each other, this means that they doesn't intercept each other.

Example

Again, we want to share the secret $D = 4321$ amongst 6 people and we want at least 3 of them needed to recover the secret. Select a prime $p = 7333$.

Next we randomly generate a *k-dimensional* point $P = (4321,12,35)$ where the first coordinate is the secret we need to share. Generate random a and b values modulus p to form a set of n linearly independent planes:

$$A = (a_1, \dots, a_n) = (10,11,13,23,33,25)$$

$$B = (\mathrm{b}_1, \dots, \mathrm{b}_\mathrm{n}) = (14,25,18,21,43,6)$$

Finally we compute the last monomial for each plane, completing our shares set:

$$c_i \equiv z_0 - a_i x_0 - b_i y_0 \; (mod\, p) \rightarrow C = (655,3535,2310,3062,3586,1933)$$

To recover our secret we start by selecting k out of n planes and build a congruences system matrix. Now we can solve the congruences matrix by applying the Gaussian elimination process obtaining the secret:

$$\begin{bmatrix} 10 & 14 & -1 \\ 11 & 25 & -1 \\ 13 & 18 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \equiv \begin{bmatrix} -655 \\ -3535 \\ -2310 \end{bmatrix} \; (mod\; 7333)$$

$$\Leftrightarrow$$

$$(x, y, z) = (4321,12,35)$$

As in the Shamir algorithm people correspond to a set of heterogenic clouds and the secret we pretend to share correspond to the seed used to generate a key or a set of keys which in turn cipher the stored data. Unlike Shamir scheme, this scheme cannot be considered perfect because at each plane intersection the solution space is reduced [39]. For instance in Figure 3 we can check that without any plane our point of intersection is localized in a 3-Dimensional space. By adding one plane we know that the secret is localized is a plane. By adding other plane we get a line of intersection corresponding to a reduced secret space. Since we are working in Z_p this scheme can be considered ideal.

2.1.1.3. Asmuth-Bloom Secret Sharing Scheme

Invented by C. Asmuth and J. Bloom and published in 1983 the Asmuth-Bloom secret sharing algorithm [28] is based on the Chinese Remainder Theorem. The Chinese Remainder Theorem [40] states that a set of linear congruences with distinct co-prime moduli has a unique solution in moduli multiplicatory. This is:

Let $m_1, m_2, m_3, \dots, m_n$ be positive integers such that:

$$\gcd(m_i,\ m_j) = 1 \text{ for } i \neq j$$

Then the simultaneous linear congruences:

$$x \equiv a_1 \ (\text{mod}\, m_1)$$

$$x \equiv a_2 \ (\text{mod}\, m_2)$$

$$\vdots$$

$$x \equiv a_n \ (\text{mod}\, m_n)$$

Has a unique solution modulo $m_1 \dots m_n$ satisfying all these congruences.

Algorithm

The algorithm is divided in two phases. The first phase or dealing phase is described as follows. Let D be the secret we pretend to share and let
$m_0 < \cdots < m_n$ be a set of co-prime values, $\forall_{\text{i,j}} \gcd(m_i, m_j) = 1$

$$m_0 \cdot \prod_{i=0}^{k-2} m_{n-i} < \prod_{i=1}^{k} m_i \text{ with } m_0 > d$$

Let $M = \prod_{i=1}^{t} m_i$ the dealer select a random a for $y = D + a \cdot m_0$ such that $0 \leq y < M$. To each player a share of the secret (a congruence of the system) is delivered:

$$y_i \equiv y \, mod \, m_i$$

The recovery phase consists in selection at least k shares (k congruences) $y \equiv y_i \ (mod\, m_i)$ for any $i \in [1, n]$, in turn, we apply the Chinese Remainder Theorem in order to find y. Finally we apply the modulus to y to find the secret.

Example

We want to share the secret $D = 4321$ amongst 6 people and we want at least 3 of them needed to recover the secret. We select a m_0 and n co-prime numbers accordingly to the previously described algorithm restrictions:

$$m_0 = 25561$$

$$(m_1, \dots, m_n) = (38219{,}38231{,}38237{,}38239{,}38261{,}38273)$$

Then we generate a random value $a = 1104742420$ and compute:

$$y = 4321 + 1104742420 \cdot 25561$$

Now we obtain a set of congruences ($a_i = y \bmod m_i$):

$$x \equiv 10330 \bmod 38219$$

$$x \equiv 9405 \bmod 38231$$

$$x \equiv 12243 \bmod 38237$$

$$x \equiv 16349 \bmod 38239$$

$$x \equiv 4831 \bmod 38261$$

$$x \equiv 34276 \bmod 38273$$

To recover the secret we select a subset of congruences $t \geq 3$:

$$x \equiv 10330 \bmod 38219$$

$$x \equiv 12243 \bmod 38237$$

$$x \equiv 4831 \bmod 38261$$

Then applying the CRT algorithm we compute:

$$M = \prod_{i=1}^{n} m = 55913856468683$$

$$(1462985857 \cdot 34023 \cdot 10330) +$$

$$(1462297159 \cdot 34431 \cdot 12243) +$$

$$(1461379903 \cdot 8009 \cdot 4831) = 28238321001941 \bmod 25561 = 4321$$

This scheme should be considered perfect due to the fact that the modulus value is unknown for the attacker so with just part of the congruences it is impossible to know where de solution stands. As with others, this scheme is considered ideal because are working in Z_p. Though only Asmuth-Bloom has homomorphic properties documented in [41], Shamir and Asmuth-Bloom can be used in such way through the point or plane translation respectively. Although interesting to maintain key freshness, this kind of properties will be addressed as future work.

2.1.2 Threshold Signatures

With the paper work being replaced by email and electronic documents the use of traditional signatures has been voided. The use of digital signatures offer not only data authenticity as provide a message integrity proof, this is, the signer cannot be impersonated and the authenticated messages can be verified against corruption or tampering. Signatures as RSA or DSA are widely used and leverage message integrity and authenticity verifiable by any party, provided that a public key is available. The use of public/private key based signature schemes confines the authenticity to a unique key and so to a unique entity that detains the key, which can be seen as an unique point of failure or simply very limited when a set of parties represent an unique entity. Multisignature and Threshold Signatures Schemes has been designed to solve such problems. As defined in [42] , there are two major differences between threshold signature schemes and multisignature schemes. Multisignature schemes, unlike with threshold schemes, do not restrict a minimum number of parties to generate a valid signature, i.e., do not define a threshold. Finally in multisignatures schemes each entity signs the message individually while in threshold signatures schemes a well-defined group of entities sign the message using a splitted key.

The existing threshold signature schemes are based on existing signature schemes as ElGammal (discrete logarithm problem), RSA (factorization problem) or Elliptic Curve ElGammal (elliptic curve discrete logarithm problem). A threshold signature scheme is defined by two basic requirements [42]:

- At least t participants in the group can collaborate to generate a valid signature on behalf of the group signature;
- Anyone who plays the role of a verifier can use the group's public key to verify the group signature without exposing the identities of the signers.

Our solution approach, based on an algorithm defined at [35], explores a RSA approach mixed with Lagrange interpolation formula (as in Shamir secret sharing scheme) complemented by individual share verification process that enables fault detection avoiding process failure. The author describes the algorithm as unforgeable and robust in the random oracle model, this is, no attacker should be able to generate a valid threshold signature and corrupt players present signature process should not be able to prevent fair players from generating valid signatures. Furthermore it is assumed the hardness of the RSA problem.

The article defines a set of players, a trusted dealer and an adversary as part of the system model. Additionally he defines t as the number of corrupted players and k the minimum number of signature shares needed to obtain the signature. Being l the total number of players, k and t are restricted by $k \geq t + 1$ and $l - t \geq k$. Finally, it's defined a signature verification, share verification and share combining algorithms used in the various process phases. The algorithm defines a dealing phase in which the dealer generates a public key along with l private key shares and l verification keys. After the dealing phase each player including the corrupted ones signs a message of their choice (it is intended to be the same legit message) resulting in l signature shares. Once signed each player outputs their signature share. The use of a share verification algorithm before the combing phase allows the selection of valid signatures that are then used in the combining algorithm resulting in a signature.

As defined by the author the signature scheme assumes a "static corruption model" in which the adversary can choose which players corrupt. In TSKY-TMS system the use of multiple storage clouds in a Byzantine fault tolerant model rise issues regarding the adversary attacks nature and location, for this purpose the use of threshold signatures is suitable in a way that middleware acts as dealer and players signing the same data and distributing the verifiable shares by the multiple storage clouds, in this way, an attack can occur in any storage node as long as the middleware is trustable and the number of affected nodes do not exceed the threshold value.

2.1.3 Homomorphic encryption

Due to cloud based platforms characteristics and costs, the migration of data has become increasingly interesting. To keep data safe we can apply a set of widely used techniques, namely symmetric/asymmetric crypto techniques along with replication. Though, the secure data must be managed by applications running in controlled/trusted environments. Due to the hostile nature of cloud environments the migration of such applications can be considered a thread to data safety.

To allow the execution of operations over encrypted data, without exposing any critical information, homomorphic encryption techniques has emerged as techniques allowing such operations. In this way is possible to keep both applications and data in public cloud storage without threatening data privacy. There are two groups of homomorphic schemes: the most common ones, called partial homomorphic cryptosystems allow specific operations to be executed over encrypted data (addictions or multiplications), for instance Paillier allow addition over encrypted data:

$$E(x_1) * E(x_2) = (g^{x1}r_1^m)(g^{x2}r_2^m) = g^{x1+x2}(r_1r_2)^m = E(x1 + x2 \bmod m)$$

A homomorphic scheme supporting both addiction and multiplication is called a fully homomorphic scheme. Through the time, various authors have proposed these kind of schemes. Recently (in 2009) Gentry [43] proposed a fully homomorphic encryption scheme based using ideal lattices along with the formalization of theorems and definitions that can be used to create other homomorphic schemes. Subsequently, Dijk, Gentry, Halevi and Vaikuntanathan created a fully homomorphic scheme [44] based on modular operations over integers which does not require ideal lattices.

Due to poor performance of fully homomorphic schemes different approaches have been taken, namely the use of multiple partial homomorphisms in a context-aware approach, in which the used homomorphisms are chosen based on the target application and data.

2.1.3.1. CryptDB

As the name suggests CryptDB [45] is a middleware system that provides practical and provable confidentiality in the face of data disclosure attacks over database systems. The base idea of the system is enabling the execution of SQL queries over encrypted data, to do so homomorphic properties are explored using SQL-aware encryption schemes. Since the encryption schemes are SQL-aware some primitive operators must be considered, namely equality or inequality checks, order comparisons, aggregations and joins. Furthermore CryptDB implements an adjustable query-based encryption in which encryption schemes are adjusted accordingly to the runtime executed queries. The encryption schemes are arranged in onions. Each onion, define a set of layers and each layer allow the execution of a certain type of operation. The choice of such layers is dependent of the kind of operation and security level we want to guarantee. For instance, layers go from Random, with maximum security offering in distinguishability under CPA to deterministic with weaker guarantees.

Finally CryptDB chains encryption keys to user passwords and so in order to access data user's password is needed. If the target user is not logged and if the adversary has no access to the user's password, the adversary cannot decrypt the stored data even if DBMS and application server are completely compromised. Through annotations over application's SQL schema, the developer can define security and sharing policies, defining which users have access to each data item. The system architecture defines two parts, a database proxy and an unmodified DBMS. CryptDB defines two threat levels, one in which a curious database administrator (DBA) try to learn data by snooping into DBMS server and a second threat in which the attacker takes control of the application and DBMS server. The first threat is defined as occurring in the DBMS server while the second thread also includes the database proxy containing users' active sessions.

One of the key points of CryptDB system is the seamless integration with existing database backed up applications and so to evaluate such solution the authors have used real applications, namely they used phpBB revealing a 14.5% of throughput reduction. Although convincing the system presents some weaknesses namely regarding data managed by users that are currently logged.

2.1.3.2. Discussion

Due to the cloud nature of TSKY-TMS solution, the inclusion of homomorphic schemes leverage the privacy of the overall solution by reducing the exposed data stored locally in the middleware. This is, since the middleware is running locally or in a computational proxy we can take advantage of homomorphic schemes applied to a locally stored index, protecting the data from possible index disclosure attacks. It would be interesting to apply such homomorphic operations to the stored data itself, but since the data stored in cloud storage repositories cannot be directly manipulated the use of homomorphic schemes is confined to the locally stored structures, namely the index.

2.2. Data Management

2.2.1 Dependable Data Storage Systems

2.2.1.1. Farsite

Farsite [46] is a distributed file system, secure and scalable that works like a centralized file server. The system is designed to execute on client machines, offering data availability, reliability, confidentiality and integrity over the stored files and directories. It explores a local cache system with slow update propagation in order to achieve a good performance. The system protects the data against corruption or unavailability through the use of symmetric encryption and byzantine fault tolerant replication. To manage the permissions different types of certificates were defined: namespace, user and physical machine certificates assuring the authenticity of all engaged entities.

Each system machine can take three distinct roles: it can be a client, a member of a directory group or a files host. The client machine interacts directly with the user while a member of a directory group is a machine that manages the data using a fault tolerant byzantine protocol. Every member in the group stores a data replica and as new requests arrive the group handles them in a deterministic way, updating the replicas and sending confirmation to the client. Along with semantic verification in group accesses this mechanisms offers system reliability and data integrity.

By using cryptographic secure hashes over the content the directory groups can detect corruption in hosts stored data avoiding overall data corruption, this is, detecting replica corruption the system avoids actual data corruption. In order to achieve reliability and availability Farsite replicates the directory metadata by multiple directory groups and the files data across multiple hosts. To redundantly store data erasure codes are used instead of simple replication. If a machine stays unavailable for a substantial amount of time its functions are migrated into other machine using other data and metadata replicas in order to generate a new replica.

Data privacy is archived by the means of using symmetric encryption over files content and sensitive metadata. When the client creates a new file, a random symmetric key file is generated then the client calculates the file's hash and uses it as encryption key to the newly created file. Finally the created encryption file is used to encrypt the hashes instead of the data directly. This technique is called convergent encryption because a set of identical files converge into a single ciphered file independently of users key. Block level encryption instead of file level encryption allows the execution of operations over a subset of blocks avoiding reading or writing the entire block. Furthermore the use of blocks allows read on demand operations, e.g., streaming applications.

Farsite system, as other distributed file systems, is very interesting concerning this thesis since it addresses confidentiality, integrity and availability issues over a file system which enables a comparison regarding de data stored in the multiple hosts. The use of certificates, addressed as future work in the current thesis, can be used to leverage the authenticity of the multiple instances of the middleware and probably for users and/or clients' access control. Other fact that is considered interesting and that was explored in the presented middleware solution was the use of hashes in order to detect file corruption. In the current solution the objects' cloud identifiers are data hashes that allow integrity check on data retrieval. Although addressed as future work, the use of convergent encryption, as presented in Farsite system, could be used as means to reduce the storage costs which can be interesting regarding the cost model improvement objectives.

2.2.1.2. EHR

The system described in the article intend to ensure security and privacy of Electronic Health Records or EHRs [47], this is, it is required to guarantee that health records are kept confidential being only accessible by the health system personnel and their patients. To ensure such properties the system makes use of an Attribute Based Encryption or ABE. An ABE system is a public key cryptographic system in which each user's key is labeled with a set of attributes and the ciphered content is associated with an access policy. The user's secret key can only decipher the content if the key associated attributes satisfy the content access policy.

The solution assumes the existence of a trustable authority or TA that generates users' keys and publishes the public values. The system assumes that cloud storage providers provide a reliable service and that data can be manipulated and analyzed by the same providers.

Each patient record consist in an encrypted file along with a metadata table describing the encrypted file including the encrypted file location, a plain access policy and a search index or SI for search operations over the ciphered content. To search over the ciphered data the system uses a Secure Channel Free Public-Key Encryption with Keyword search or PEKS. This mechanism gives the users private search abilities over ciphered data without revealing the search keywords or the partial matches with the server.

This system is interesting, regarding this thesis, due to the homomorphic search operations it offers. Furthermore, as in TSKY-TMS, EHR has main focus in protecting data privacy using symmetric keys but complemented with a policy based public key system.

2.2.1.3. Silverline

Silverline [48] is a system designed to improve the security of data stored in public clouds. Silverline is based on a simplified model in which the data is encrypted by the client before storing it to the cloud. Since Silverline is a system designed to be used as a middleware between applications and databases, and since some database operations cannot execute over ciphered data (eg. inequality comparisons or field aggregations), a mechanism was required in order to determine whether the data could be encrypted or should be let plain. Once determined whether data should be encrypted it is necessary to assign symmetric key(s) to one or various data fields. One solution would consist in using one global key to all encrypted data. Although simple, this solution would raise many security issues regarding a single point of failure. On the other hand, a key per each data field could be used instead but although highly secure it introduces an overhead in key management and distribution. So the authors came with a hybrid solution in which are created data groups defining encryption domains with a single key per each group, this is, the loss or compromise of one key result in data loss or disclosure of only one of the groups data fields.

Silverline ensures a transparent user access, through the use of HTML5 based applications without the need of using modified browsers. Although not approaching integrity as an issue, this system presents an interesting hybrid solution regarding the use of multiple keys versus a single key. In TSKY-TMS, a multiple key scheme is used, where each object is encrypted with a different key. This approach can reconsidered towards latencies and costs optimization, and so a solution where multiple objects are merged into a single object creating larger encryption domains is desirable being addressed as future work.

Other interesting feature of Silverline is the reduced TCB model in which keys are stored in a client side browser using current technologies, and so the data security relies on client side stored crypto material. The current approach of TSKY-TMS has defined a REST API promoting the use of web based technologies as interface to the evaluated system. A Webmail system with a browser based key management system would be highly desirable and will be also addressed as future work.

2.2.1.4. Discussion

Although in a different scope, Farsite file system, as in TSKY-TMS, assures some dependability attributes and so can be considered a dependable system. Data confidentiality is achieved by combining symmetric convergent encryption combined with asymmetric keys that may be used in future as a means to archive data privacy while reducing storage costs.

The EHR system, unlike the other systems, presents a solution aimed at ensuring confidentiality using a policy based approach where user's access to data depends on a key and an associated policy. Regarding the current thesis it would be interesting explore such solution as a mean to control multiple middleware instances or multiple users. Since the current thesis do not addresses multiple users or multiple instances of TSKY-TMS this mechanism will be assigned as future work.

2.2.2 Cloud Oriented Dependable Solutions

As with the current system there were examined a set of dependable systems. The term dependability [49] is defined in three parts: the threads to the system dependability, the means by which dependability is attained and a set of attributes. The described systems cover some of combination of dependability attributes, this is, cover availability, reliability, safety, confidentiality, integrity and/or maintainability. Beyond dependability these systems are also characterized by functionality, performance and costs discussed in the end of each subsection, thus creating a comparison with the presented system.

2.2.2.1. Depsky

The DepSky system [11] is described by the authors as a cloud-of-clouds system that leverages availability, integrity and confidentiality of data stored in storage clouds and that deals with data integrity and availability through the use byzantine fault tolerant replication mechanisms.

DepSky cloud-of-cloud model mitigates vendor lock-in problem by the use of multiple storage providers disposing the dependency of single providers. Furthermore the use of multiple clouds combined with erasure code techniques assure not only data integrity and availability as allows the cost reduction of data storage costs up to 50%, as stated in DepSky storage evaluation. By exploring a set of storage clouds as simple storage nodes and by directing all processing needs to a client or to a middleware server, along with encryption and secret sharing techniques the system attains a TCB (Trustable Computing Base) reduction to the client itself while offering confidentiality (in the DepSky-CA version). The DepSky-A version supports all dependability attributes supported by CA version other than confidentiality.

The system defines each data unit or DU by a unique identifier, a version number (in order to support versioning and updates), application specific metadata, integrity codes and the data itself. The system was designed considering three entities, the writers, the readers and the cloud storage providers. The system assumes that both readers and writers can corrupt the stored data. As mean to solve corruption and as a concurrency control mechanism DepSky proposes a low contention locking mechanism that uses the storage clouds itself as lock platform. In this manner is possible to use authenticated locks, through the use of asymmetric cryptography, in order to assure authenticated concurrency concerning multiple writers and readers.

The use of secret sharing techniques, erasure codes and multiple storage providers provides the system support to a byzantine fault tolerant model where *3f+1* clouds support at most *f* faults.

The described system (TSKY-TMS system) is very similar with DepSky system, as in TSKY-TMS, DepSky-CA explores a cloud-of-clouds model in order to provide data availability, confidentiality and integrity making DepSky the most similar dependable system analyzed. Although the concurrency control is addressed as future work the TSKY-TMS supports integrity and authenticity through the use of signatures, more precisely threshold signatures that are suitable for the replication model that TSKY-TMS supports.

2.2.2.2. iDataGuard

The authors describe iDataGuard [12] as a system to combat cloud diversity and heterogeneity while enforcing data security with search over encrypted data functionality.

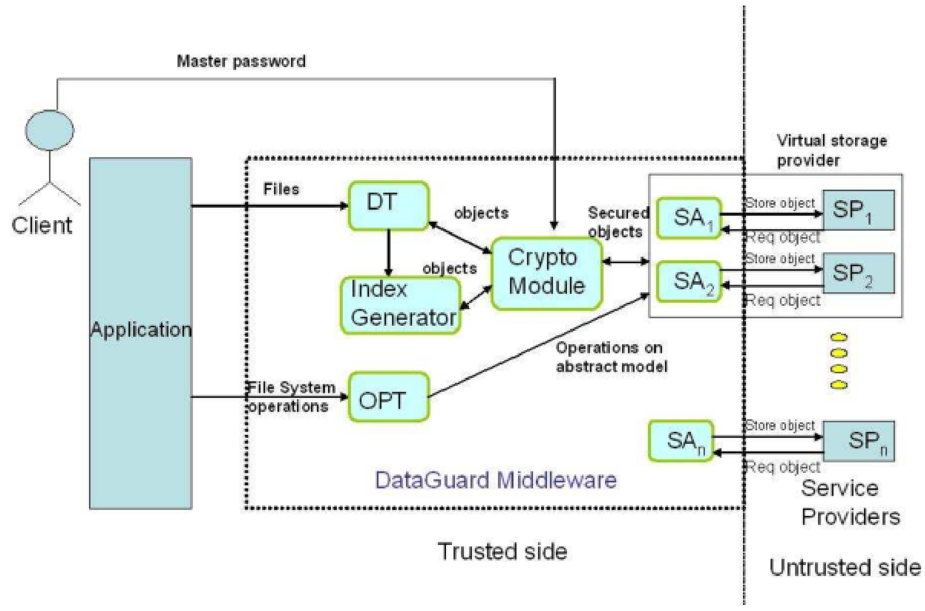


Figure 4: iDataGuard Data Model

As shown in Figure 4, the iDataGuard architecture is defined in two main components, a client and middleware components that execute in a client machine or alternatively in a secure proxy. The system provides two distinct interfaces, one based on a file system and other based on abstract objects in which the file system interface is based on. The system execution is initiated by the conversion of the outsourced data into the abstract data model though the data translator component or DT. The cryptographic module or Crypto Module uses a PBE scheme based on a master password in order to encrypt the contents. Finally an index generator creates a cryptographic index for all text based files.

Each system object is represented by a unique identifier (generated by concatenation of full path and users' password making it deterministic), a file name, content and metadata stored in a key-value approach. Since the identifier doesn't provide any information it is kept plain while the metadata is stored encrypted using an object encryption key or OEK. This key is generated on-the-fly using a key derivation function or KDF of PBE PKCS#5 specification using as salt an random string preventing from key pre-calculation to the most common passwords. The salt is stored in the plain section of objects metadata, in this way, iDataGuard can regenerate the key on demand. The data integrity is provided through the use of a Hash Based Message Authentication Code or HMAC applied to the identifier, name, content, metadata and version, which ensure the object most current version.

In a similar way, the proposed middleware system ensures confidentiality through the use of on-the-fly generated keys. As in iDataGuard, TSKY-TMS also faces cloud storage heterogeneity by the use of adapters that implement a standard cloud storage interface. Furthermore, the current TSKY-TMS architecture includes an index allowing search operations over encrypted data through the use of an index with homomorphic properties.

2.2.2.3. HAIL

HAIL [13] or High-Availability and Integrity Layer for Cloud Storage is a system that assures availability and reliability using RAID similar mechanism where storage clouds are used as base storage rather than traditional hard disks. The HAIL defines assurances or proofs in two dimensions, the PDP or *Proof of Data Possession* that assures the presence of data and POR or *Proof of Retrievability* in which is assured the existence of an effective data recovery process.

To assure a Byzantine fault tolerant model HAIL uses replication mechanisms that, i.e., *f* replicas out of *3f+1* can be corrupted without compromise data retrievability. To do an integrity check HAIL randomly extracts a data block from each replica checking its equality. In case some discrepancy is found the system rebuilds a non-corrupt replica using the remaining replicas.

The PDP could be addressed in future implementation, complemented by threshold signatures, in order to detect possible corruptions while reducing data transfer rates and consequently reducing the associated costs. However this technique would not be possible to implement taking into account the inexistent data correlation, i.e., in the current implementation each stored replica differs due to the use of different keys in different blocks.

2.2.2.4. RACS

RACS [14] or Redundant Array of Cloud Storage is a system that splits data in a similar manner to RAID systems, using as storage backend various storage clouds. The system, as in TSKY-TMS, offers a transparent and reliable storage service providing data availability and integrity and avoiding vendor lock-in issues by the use of replication mechanisms. RACS, as other cloud storage providers provides retrieval, insertion, deletion and listing operations over data indexed by a unique key.

When a data insertion request arrives, the actual data is divided in *m* similar size parts (*1/m* of the original size). Then RACS uses erasure codes in order to create *n-m* redundant parts totaling *n* parts, where redundant parts size is similar to the non-redundant ones. When a data retrieval request arrives, RACS gathers *m* parts and reconstruct the original data. The objects' metadata is replicated by multiple servers to enable efficient load distribution.

The use of a single RACS proxy to process all data insertion and retrieval operations was considered, by the authors, a system bottleneck. To mitigate such bottleneck related issues, RACS system was designed to run in a distributed system in which multiple RACS proxies coexist exploring a set of common cloud storage repositories. Each proxy stores a limited quantity of data, including users' authentication data, locations and credentials of each repository. Nevertheless, changes in data must be propagated and reflected in the remaining proxies, furthermore, the use of multiple proxies using the same repositories raises concurrent access issues in write operations. To solve this problem

RACS coordinates the access operations is such way that can only exist a writer with multiple readers at once for each key-value pair, in other words, RACS implements one-writer many-readers model.

The RACS authors discuss the possibility of using more complex client applications that can make use of configurable repository selection in order to explore geographical proximity and consequently reduce latency. Furthermore, the use of configurable repository sets enables the use of load distribution techniques between multiple RACS proxies. RACS implementation explores two different providers (Amazon S3 and Rackspace Cloud Files) complemented by a local disk replica.

In a brief analysis we can verify the existence of common features between RACS and the system we have designed, namely in the use of a common set of heterogenic cloud storage providers explored by multiple instances of RACS proxy (addressed as future work in the current thesis). The use of geographic adaptation to latency reduction is also addressed as feature work in the current thesis. Concluding, RACS is a dependable system ensuring data integrity and availability, present in this thesis main focus, taking into account the costs associated with the use of multiple providers, addressed as a future work issue in this thesis.

2.2.2.5. Discussion

The majority of the referred systems present specific mechanisms that were or will be possibly incorporated in TSKY-TMS middleware implementation. DepSky system ensures integrity, availability and confidentiality through the use of erasure codes and secret sharing schemes combined with Byzantine replication. The use of low contention locks, as defined by DepSky authors, is addressed as future work and will perform an important role in concurrency control mechanisms. Furthermore, garbage collection mechanisms, also included in DepSky's solution could be used, along with erasure codes, as a means to reduce storage costs.

HAIL and RACS system share a RAID philosophy, ensuring integrity and availability, where data is stored in multiple storage clouds viewed as local hard disks. HAIL, unlike in RACS, defines PDP and POR proofs that could be used as a means to ensure data storage reliability and will be addressed as future work. On the other hand, RACS models a distributed system by the means of using multiple RACS proxies, raising concurrency issues regarding write operations solved using a single-writer multiple-reader model achieved by using Apache Zookeeper library.

2.2.3 Final Discussion

	DepSky	iDataGuard	Farsite	HAIL	RACS	EHR	Silverline
Confidentiality	X	X	X			X	X
Integrity	X	X	X	X	X		
Availability	X	X	X	X	X		
Handle Heterogeneity	X	X			X		
Distributed			X		X		
Search Support		X				X	

Table 1: Comparison table between multiple dependable systems

Through a brief observation of Table 1 we verify that generally all systems provide data confidentiality, integrity and availability guarantees which are the main focus of this thesis and are supported by a set of components, some similar to the ones used in the presented systems. Some systems handle heterogeneity present in cloud storage providers. Since we pretend to use multiple cloud providers we have implemented a set of adapters to multiple providers, described in the next section, providing a common PUT/GET/REMOVE interface to the other system components. The search functionality is other feature supported by some systems. In the TSKY-TMS system search is provided as a context specific layer of the middleware, though this feature is not considered main focus in the presented work.

2.3. Cloud Storage Platforms and Services

2.3.1 Amazon S3

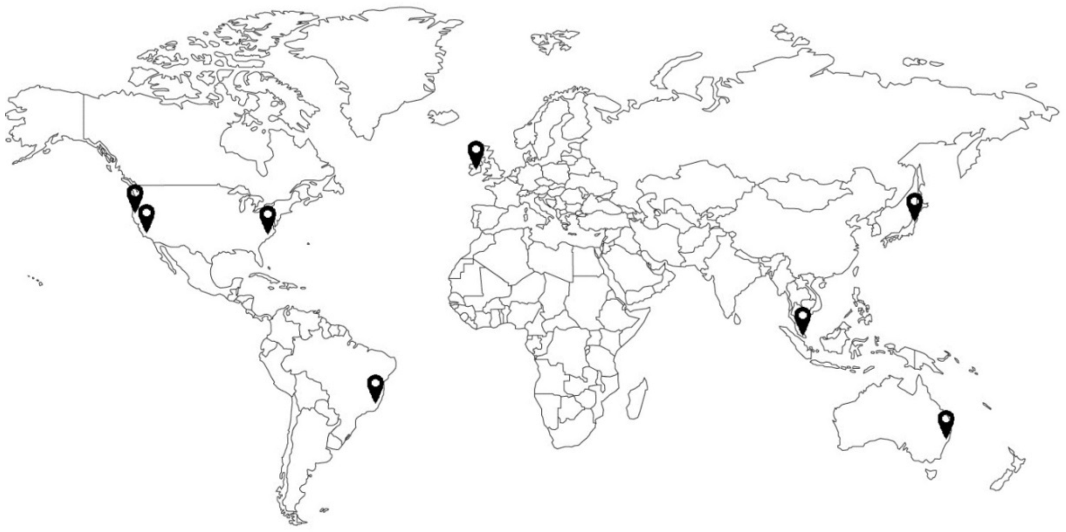


Figure 5: Various worldwide Amazon data center locations

The cloud storage service Amazon S3 [15], provided by the Amazon corporation allows the creation and destruction of objects with sizes understood between 1 byte and 5 terabytes. There are no limitations regarding the number of objects that can be inserted. The objects are stored under an unique identifier and are organized under buckets located in one of the eight word wide locations (Oregon, US; California, US; Virginia, US; Ireland, EU; Singapore, Asia; Tokyo, Japan; São Paulo, Brazil; Sydney, Australia) as shown in Figure 5. In each region the data is replicated across multiple data centers and/or inside each data center.

To protect data against attackers, by default the stored objects can only be accessed by the owner and all operations are register on each access so it can be checked later. Apart from access control mechanisms, Amazon ensures 99.999999999% data durability and 99.99% of yearly availability. This provider also offers an out-the-box cryptography solution named Amazon S3 Encryption Client or Amazon S3 Server Side Encryption to client side or server side encryption respectively. The security on communications is provided by the use of SSL channels for data transfer. The integrity after a remote transfer is ensured by the use of checksums.

The cost model is GB oriented and can be reduced with increasingly data volumes. Additionally PUT, COPY, POST, LIST and data transfer requests are charged, which could result in a vendor lock-in situation.

In Amazon SLA there are a section presenting some terms exclusions applicable in case of major occurrences as natural disasters or service termination, threatening service and data availability or integrity.

2.3.2 Google Cloud Storage

Similarly to Amazon S3, Google Cloud Storage service [16] provides an object oriented storage service and such objects are organized in buckers accessible under a unique identifier through a RESTful API. This provider offers data availability through an inter-datacenter and possibly intra-datacenter replication. Although the precise object location is not disclosure the objects may be stored in United States or Europe data centers. This provider uses an OAuth2.0 based authentication as access control mechanism. As consistency model Google defines a read-after-write consistency model in which data can only be read once all write operations are concluded.

Google defines an availability period of 99.9% per month with a credit based policy in case of lower availability values. As in Amazon, Google defines exclusion premises regarding durability and availability, so similarly with Amazon some issues, regarding data security, can be raised. As in Amazon the GB oriented cost model can introduce vendor lock-in situations.

2.3.3 Nirvanix Cloud Storage

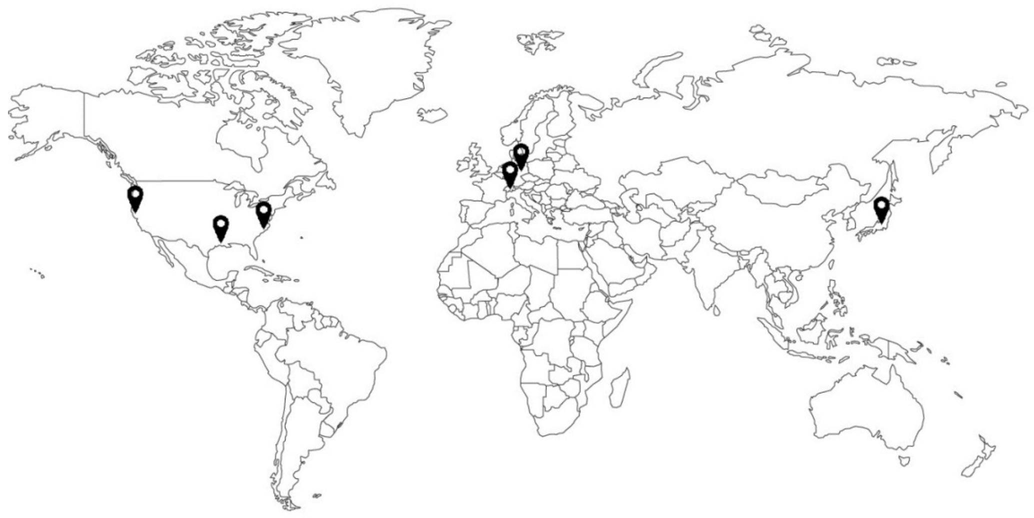


Figure 6: Various worldwide Nirvanix data center locations

The cloud storage service provided by Nirvanix [17] is supported by storage nodes globally spread (California, US; Texas, US; New Jersey, US; Switzerland, EU; Japan) as illustrated in Figure 6. Rather than provide an object based storage, Nirvanix provide a storage service similar to a file system in which directories and files are hierarchically organized. To file management this provider offers upload, copy, move, rename and file deletion primitives.

The service reliability lies on replication through multiple nodes and the use of RAID architecture locally in each node or data center. In addition, integrity is provided by the means of hash based verifications before and after data transfers. As reliability guarantee the service provided by Nirvanix is certified as a SAS 70 level 2 that certifies the entity as capable of managing the data in a controllable manner accordingly to data security goals.

The SLA defines an availability period of 99.9% per year for simple storage, 99.99% to policy based storage with two replicas and 99.999% with three replicas. As the other providers SLA dictates some term exclusions, namely client side failures or attacks, improper system usage or service termination.

The data confidentiality is ensured by the use of internal policies, logical node access mechanisms and by the use of SSL communication channels. Furthermore Nirvanix ensure no correlation between the data and the clients.

The cost model is GB oriented and the data transfers are charged, whereas download and upload costs are differentiated. Data volumes migration can result in a vendor lock-in situation due to the associated costs.

2.3.4 Rackspace Cloud Files

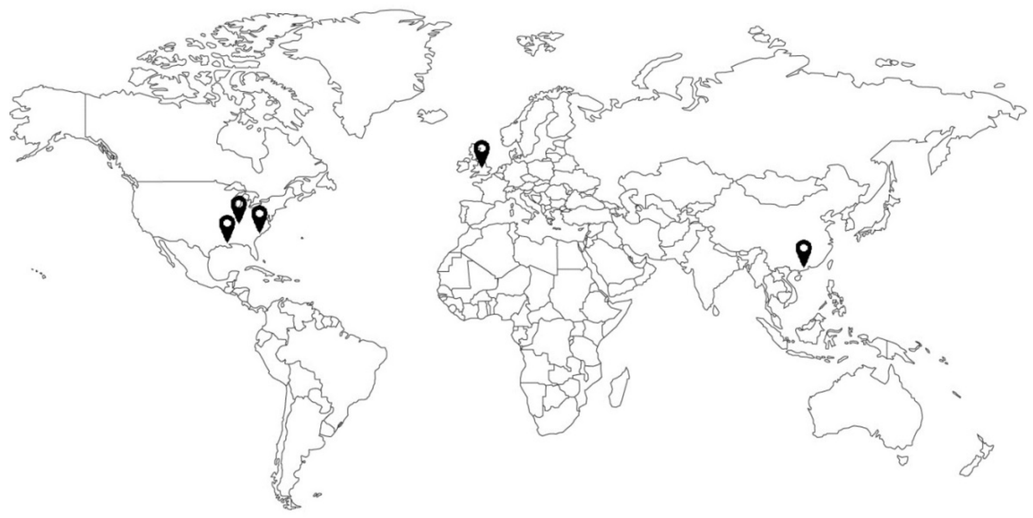


Figure 7: Various worldwide Rackspace data center locations

Rackspace [18] provides a worldwide cloud storage service with storage nodes distributed in five different locations (two data centers in Texas, US; Chicago, US; two in Virginia, US; two in United Kingdom, EU; two in Hong Kong) as illustrated in Figure 7. The security guarantees offered by this provider are mainly supported by datacenter internal characteristics. This is, the mechanisms to ensure data durability, confidentiality and availability are supported by a set of physical characteristics and internal policies associated with inner workings. As so, we cannot conclude about data safety by doing a brief external analysis.

The cost model is similar to other providers where each gigabyte stored is charged. In contrast with the other providers, Rackspace only charges download transfers and no HTTP requests are charged, nonetheless this cost model encourages data upload which can result in a vendor lock-in situation.

2.3.5 Microsoft Azure

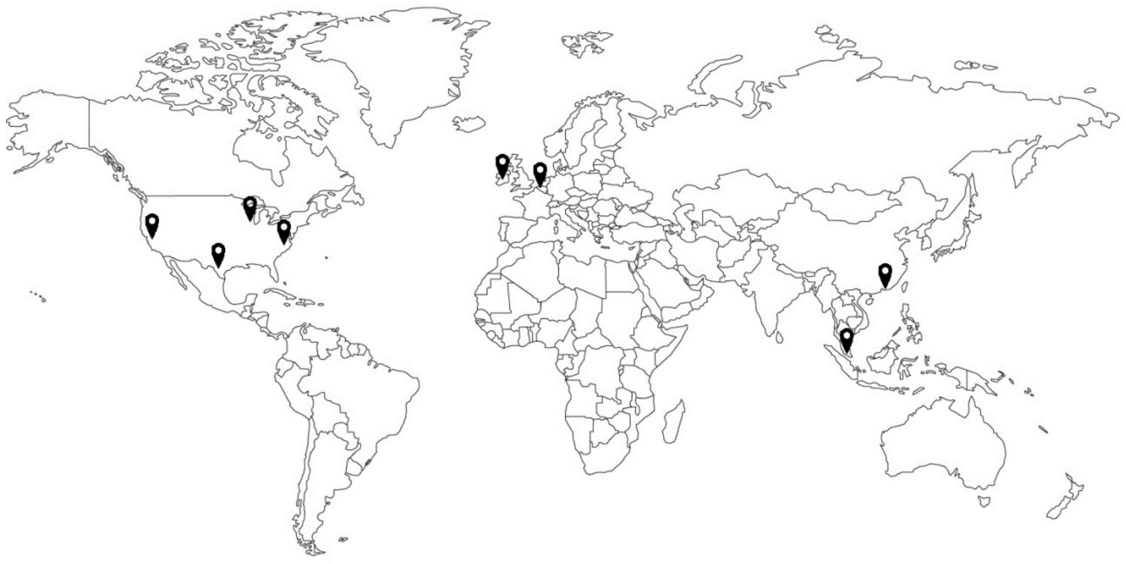


Figure 8: Various worldwide Microsoft Azure data center location

Microsoft [50] defines its data model as a set of containers containing a set of BLOBs or Binary Large Objects. BLOB can be defined in two categories: block BLOBs in which each block is transferred block by block and each BLOB can accommodate up to 200GB of data; page BLOBs designed to random access can be divided in pages and can carry up to 1TB of data. The use of BLOBs divided by blocks assists in rapid failure recovery due to the resume capacity on the most recently transferred block. Microsoft stores their clients' BLOBs in three major world spread regions namely Asia (Hong Kong and Singapore), Europe (Ireland and Netherlands) and United States (Illinois, Texas, Virginia and California), illustrated in Figure 8.

As in Amazon, Microsoft provides different data access mechanisms. It provides a RESTful API defining the common data manipulation operations, as defined in the most providers. Furthermore Microsoft provides libraries and facilitates the access to their storage service by using their cloud services or their operating systems. To improve data availability data is replicated inside each data center, more precisely, each time a BLOB is written it is replicated by three machines. Azure also provides tools to do cross-replication by multiple datacenters (geo-replication).

In contrast with the other providers, Microsoft defines in SLA, maximum latency times for the most operations. The uptimes are expected to be superior to 99.9% otherwise client will be credited accordingly. As with others Microsoft include some SLA exclusion terms in which, outside factors and client side factors are included.

2.3.6 Dropbox

With over 200 Million clients, Dropbox [51] is a file oriented and final client service providing a well-known file storage service defining fixed free and paid storage quotas, with up to 1TB of storage quota, without any file transfer limitations. The Dropbox's service terms is extended to third parties, in this case Amazon. The extension of the trustable domain could lead to undetected data leaks.

To business clients Dropbox defines security a set of security mechanisms and guarantees, as SSL connections, file encryption using AES-256, 99.999999999% durability and 99.99% availability offered by Amazon storage service, a set of certifications provided by Amazon and access control measures.

Although with similar guarantees offered by other storage providers, in 2011 Dropbox has been involved in a critical problem that endangers users' data confidentiality by the means of authentication exploit, as posted on Dropbox's blog [52], in which non-authorized authorities gained access to users' accounts.

2.3.7 Luna Cloud

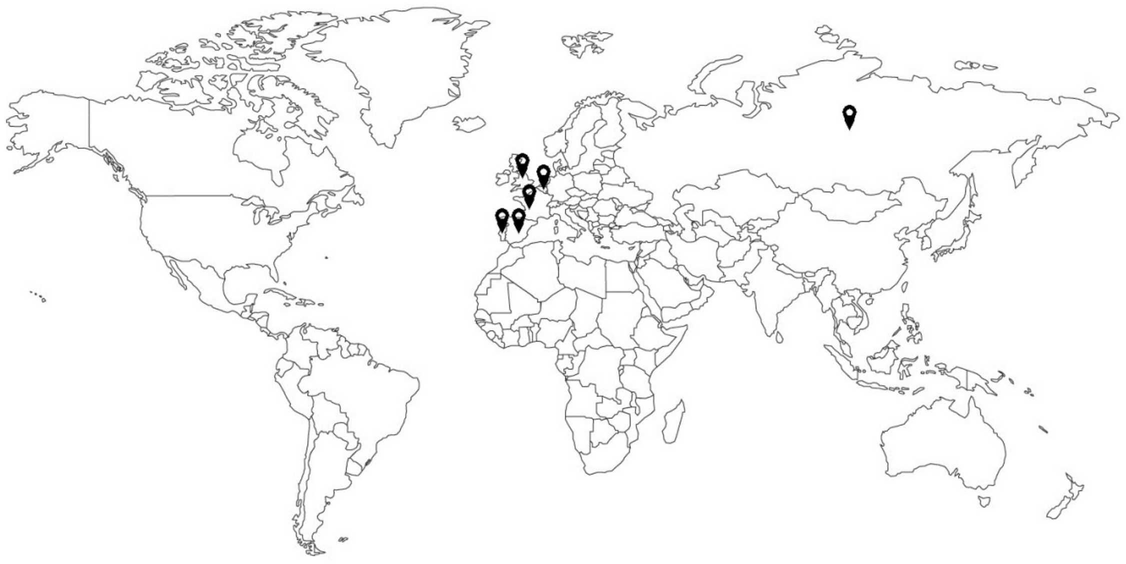


Figure 9: Various worldwide Luna Cloud data center locations

Luna Cloud [53] owns a cloud network mainly located in Europe with nodes present in Lisbon (Portugal), London (UK), Paris (France) and Frankfurt (Germany) as illustrated in Figure 9. The SLA defines a service availability of 99.99% without any client compensation. For inferior availability times the client is credited perceptually. Luna Cloud, unlike the other providers, also defines Web Panel availability time which means that at some time the service could be running without user control.

As with others, Luna defines some SLA exclusions related with system misuse by the customer, scheduled maintenance downtimes or downtimes due to "Force Majeure events", which we interpret as natural disasters or bankruptcy.

2.3.8 Comparison

Comparing the various services we can verify similar guarantees, though Amazon has the most detailed documentation and support, describing availability and durability guarantees. The Google doesn't disclosure the precise location of the stored data, defining only to zones (Europe and US). All services provide REST based APIs and a set of basic operations, as PUT, GET or REMOVE data blocks. Furthermore some providers, namely Amazon allow the user to define access policies over the data which sometimes can be translated into data breaches as some reports stated [25]. The majority of providers base their availability and durability on local or inter-region replication and on failover based machines (ex.: redundant storage).

In the current solution we have managed to explore all the described providers performing some benchmarks discussed in evaluation section. The use of multiple providers will provide us the ‘power of choice’, this is, if the system is capable of supporting multiple storage providers a subset may be chosen as storage support based on their latencies, physical location, cost model or even provided guarantees, to do so we will use profiling techniques and dynamic adaptation techniques (addressed as future work).

3 System Model and Architecture

As described before, TSKY-TMS system is a middleware service providing a dependable email repository service using a set of core components that cover most of the dependable attributes. The use of an applicational layer enables the evaluation and validation of the system in a real case scenario.

3.1. Attacker model

It is assumed that the most common attacker can, analyze and disclose any data present in individual storage clouds. These attackers can be inside employees or outside entities that have all access to the data itself and may perform all available operations over such data. The attacker can delete, alter or replace any data, nevertheless it is assumed that collusive attacks are unlikely to happen, this is, there is a remotely chance of multiple attackers with diverging storage cloud accesses collude in order to break the used threshold mechanisms. These collusive attacks are supported by TSKY-TMS in a threshold basis, the system defines a Byzantine model and so the system can handle collusion attacks of at most t out of $3t+1$ attackers, where $3t+1$ is the total number of replicas.

Although, is assumed that the middleware runs locally or in a secure proxy, eavesdropping attacks between operations or in idle times can happen without compromising data confidentiality, integrity or availability, as long as no unauthorized operations are executed.

3.2. System requirements

As system requirements, and so defined as objectives of this thesis, we have defined a set of functionalities that we intended to support by the meaning of a set of dependable components placed in the middleware lower layers. So the designed middleware includes the following requirements:

- As main end user requirement, the system must ensure data confidentiality in a way that data storage and cryptographic mechanisms can be auditable by the end user without any time-consuming bureaucracy;
- As other end user requirement, the system must ensure that the stored data is intact (not corrupted) and available at any time;
- The system should tolerate any cloud storage provider unavailability, and so ensuring data availability to the end user;
- The system must ensure that corrupt data is not delivered to the applicational layer and so data integrity must be eventually verified in at least one data processing stage;
- The system must tolerate attacks (as specified in attacker's model) in at most f out of $3f+1$ storage clouds, as defined in the Byzantine model complemented by the use of threshold schemes (secret sharing schemes and threshold signatures);
- Acceptable service time loss versus out-of-the-box solutions. This is, due to the use of multiple cryptographic and replication mechanisms it is expected to have a worse service time and so the system is intended to mitigate such negative impact;
- Support for multiple diverse storage clouds or local storage solutions as a mean to ensure data availability and integrity;
- Explore multiple cloud storage providers in order to guarantee data availability and confidentiality while reducing the data access times in the presence of an ubiquitous, and so dynamic, system access;
- Explore multiple cloud storage providers in order to mitigate vendor lock-in issues and to leverage optimized and future-proof cost model;

- Implementation of the system as a framework based system, ready to accommodate new components, defined in future work;

The email service repository was designed in order to explore the requirements fulfilled by the bellow layers complemented by a set of functional requirements:

- As MUAs (Mail User Agents) requirement, the system must provide standard POP3 and SMTP (RFCs 1939, 5321) interfaces or the secure variants POP3S and SMTP *over* TLS (RFC 2595, 2847) supporting the most common over-mailbox operations, i.e., the applicational layer must provide mailbox fetching and email message send operations;
- As additional operation (not supported in POP3 and SMTP standards), the system must provide search operations over standard email messages headers and body (RFC 2822). Since search operations are not supported by today's email standards a RESTful API must be provided instead. This interface must support HTTP requests as defined in RFC 2616. Furthermore, this interface must support all operations as POP3 and SMTP endpoints enabling the use of TSKY-TMS as a backend storage service to existing webmail solutions;
- The index data should be stored locally or remotely without revealing any information about the message contents in the occurrence of an attack. So the system should enable searching operations at any time without revealing any message data. This issue described later is addressed by the use of homomorphic schemes.

3.3. System Architecture

The system overall model was defined in three possible variants:

- A **local variant**, in which the middleware operates in a local trustable machine and so, private data could be stored in primary and secondary memories without need of further cryptographic mechanisms;
- A **proxy mode**, in which the middleware operates in a trustable machine in the domain of a user or a group of users, for instance, a server running in a corporative office;

- A **cloud variant**, in which the middleware runs on a computational public cloud outside the management domain of users and so, can be considered untrusted. Although not considered for the current system design, this approach is considered as future work due to the need of further improvements in system TCB minimization.

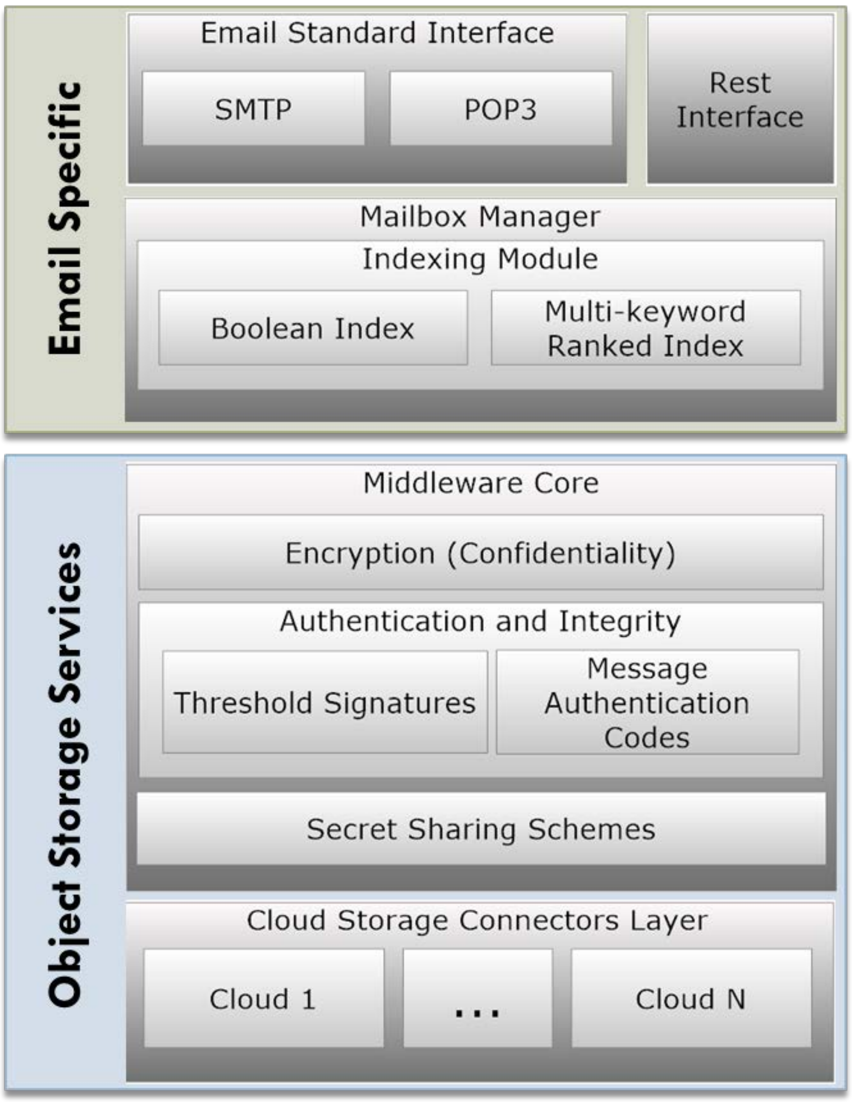


Figure 10: TSKY-TMS Architecture

The system architecture is divided in two main focus (layers), the application support and the generic object storage service that corresponds to TMS and TSKY service respectively. The overall TSKY-TMS is constituted by four main layers that can be independently explored in different contexts (as shown in Figure 10). The first layer provides a way of evaluate the overall solution by providing standard POP3 (RFC 1725) and SMTP (RFC 821) interfaces. Additionally and in order to support search operations or to provide TSKY-TMS as a service a REST API was implemented (using HTTP standard protocol).

The second layer provides an email applicational platform and was developed as a proof of concept to the middleware core. This layer enables the evaluation and validation of the underneath layers. Beyond email repository service layer other layers could be used, as for instance, distributed file system service or any other service that uses storage as service foundation. This layer provides basic email operations over users' mailboxes including search operations over headers information and over messages' contents. Although along, indexing and mailbox components could be divided in distinct layers leveraging the use of generic indexing and search operations over non-specific data. Despite this, the use of generic indexing and searching techniques would clear any semantic trace from the application itself, for instance it would disable the capacity of do searches over specific header fields. This issue/feature will be addressed as future work.

As seen in Figure 10, the first (interface layer) and second layers are application specific and use the beneath layers as storage base for email data in such way that the aggregation of the different set of layers define the TSKY-TMS system/middleware. The lower layers are used as a storage service and can be used by other applications defining the TSKY framework. The dependable service provided by TSKY framework, similar to the one provided by existing cloud storage services, defines simple PUT/GET/REMOVE and LIST operations over the stored data.

The third layer, or the core layer, belonging to TSKY framework is the layer responsible for all cryptographic operations including data encryption, using standard symmetric encryption algorithms, secret sharing over data or over encryption keys, generation of Message Authentication Codes and Threshold Signatures over the message content. Altogether, the components improve data confidentiality and integrity reducing the trustable computing base to the key management process itself, this is, to the secret sharing and threshold signatures processes.

Finally, the forth layer gives support to distributed storage over multiple heterogenic storage clouds, along with threshold mechanisms described earlier a Byzantine fault tolerant model is supported in which *f* faults, data corruptions or disclosures out of *3f+1* can without compromising data availability

Since the key management is made by the middleware itself and since the shares of the signatures and of the keys are stored in threshold basis in multiple clouds the trust base is reduced to its maximum disallowing successful disclosure attacks on middleware idle times.

3.4. Data Model

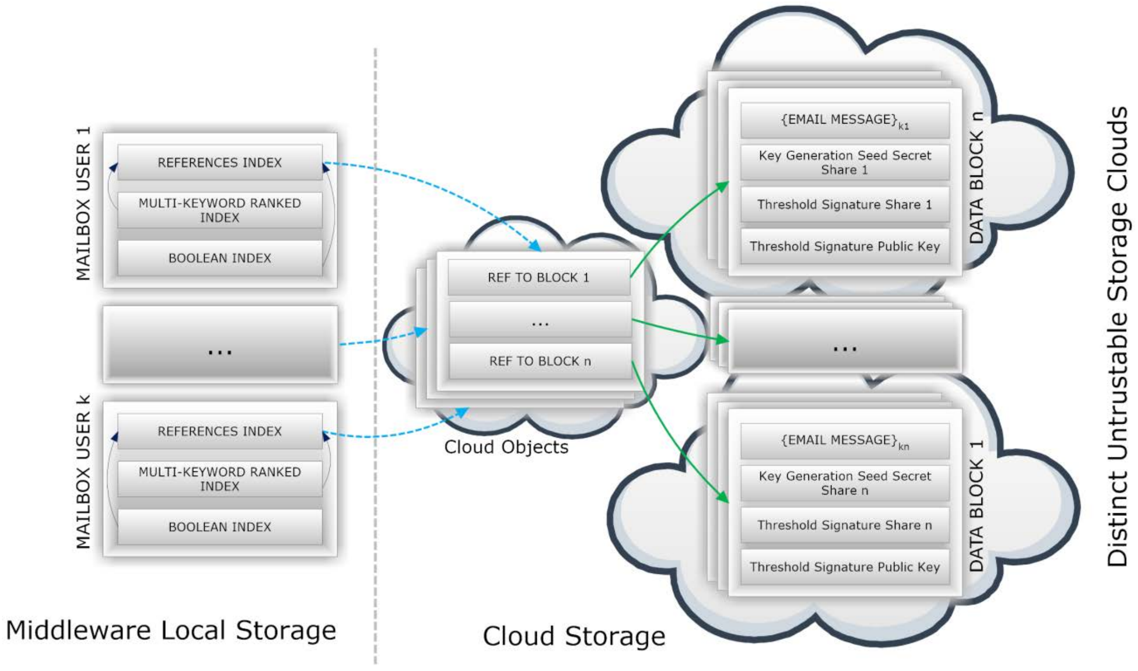

MAILBOX USER 1
REFERENCES INDEX
MULTI-KEYWORD RANKED INDEX
BOOLEAN INDEX
...
MAILBOX USER k
REFERENCES INDEX
MULTI-KEYWORD RANKED INDEX
BOOLEAN INDEX
Middleware Local Storage
REF TO BLOCK 1
...
REF TO BLOCK n
Cloud Objects
Cloud Storage
{EMAIL MESSAGE}$_{k1}$
Key Generation Seed Secret Share 1
Threshold Signature Share 1
Threshold Signature Public Key
DATA BLOCK n
...
{EMAIL MESSAGE}$_{kn}$
Key Generation Seed Secret Share n
Threshold Signature Share n
Threshold Signature Public Key
DATA BLOCK 1
Distinct Untrustable Storage Clouds

Figure 11: TSKY-TMS data model

As shown in Figure 11, the data model is divided in two sections, the trusted middleware data storage and the untrusted cloud storage repositories. Although mailbox oriented the middleware data model is generic, this is, as described before, any data storage application can use the data model, provided that contextualized or un-contextualized algorithms are defined. Applications without indexing capabilities can also explore a similar data model using the system as a raw storage system, i.e., the middleware provides to applications PUT, GET and REMOVE operations (no SEARCH operation).

For this particular case, the middleware locally stores three indexes, a reference index, consisting in a structure that directly maps each user's message unique identifier to a unique in-cloud Cloud Object reference. The identifiers of the messages consist in small integers that are incremented at each message arrival; the references are tokens consisting of cloud objects identifiers concatenated with a key used to encrypt the pointed object. Whereas the local storage is defined by application specific layers the cloud storage is inherent to the TSKY framework itself, meaning that any other application would explore an identical cloud storage data model.

The multi-keyword ranked index, held in middleware local storage, consist in an index over data content with homomorphic properties, enabling search operations over encrypted message contents using the provided REST or RMI interfaces.

3.5. System Model Generalization

The system model, architecture and data model presented in the previous sections (subsections 3.3 and 3.4) follow an approach of the TSKY middleware as instantiated to be used in a user local machine or running in a proxy server, according to different deployment scenarios as initially stated in subsection 3.3. The generalization of the system model assumptions for the case of cloud based deployment can require the addition of new components as represented in the following architecture (Figure 12).

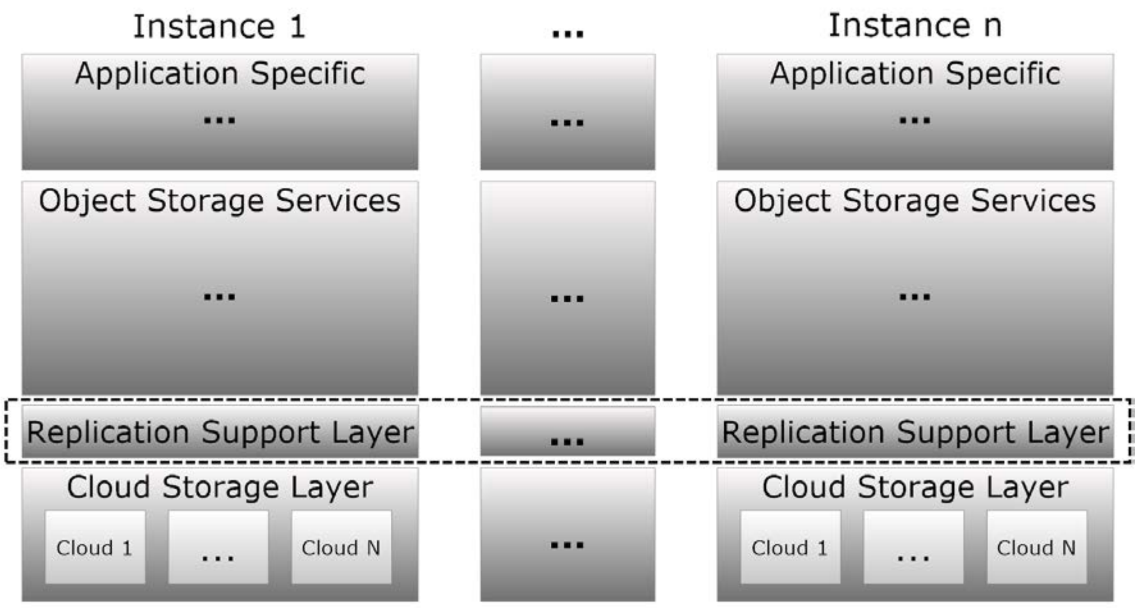


Figure 12: Cloud based deployment architecture

The presented architecture, in Figure 12, consisting in a generalization of the system model and architecture as implemented and evaluated in this thesis, it defines a scenario and a work direction in which a replication component is necessary. In the presence of multiple instances a layer must be provided in order to keep updated info on shared structures, namely indexes. Such shared structures could be supported by distributed databases as Apache Cassandra[1] or Basho Riak[2], however this direction has already been addressed and discussed in the context of TSKY project [54]. Other issues could be raised when it's assumed that instances could run on untrusted public computing clouds.

[1] Available at: https://cassandra.apache.org/ [Accessed 22/Sept/2013]
[2] Available at: http://basho.com/riak/ [Accessed 22/Sept/2013]

The use of such environments raises some concerns regarding the security of indexes and other structures. To accomplish so, a minimization and circumscription of the TCB of the middleware and its components must occur in order to guarantee security and privacy of stored data. To do so hybrid solutions, with some components running in untrusted environments combined with components running locally, could be used. Our design and implementation, as described in the next section, has main focus on the proxy variant in which no multiple-instances, state or index replication are addressed.

4 Implementation

The TSKY-TMS middleware design has been architected and implemented in multiple phases. Initially, we began by developing the TSKY framework capable of accommodating the multiple modules that were posterior developed. The email repository service deployed over TSKY framework (TSKY-TMS) comprehends a basic set of applicational interfaces, targeted to Mail User Agents or MUAs. The lower end of the TSKY framework was designed to support adapters/drivers for multiple cloud storage providers explored as storage base for the applications' data. Finally and most important a core layer was developed in order to support a set of data processing modules namely an encryption/decryption module, a secret sharing module, a threshold signatures module, a MAC module, a combination module used to build the data block to be replicated across the multiple storage nodes.

The frontend interface is made by an SMTP standard email interface used as message send endpoint and were developed using SubEtha SMTP [55]. The authors of such library describe it as in compliance with RFC2821 and in the performed tests it has reveal no lack of support to the used operations (message sending operations). The frontend also contains a POP3 interface used as message pop endpoint that were developed from scratch in partially compliance with RFC1939, this is, POP3 endpoint implements most of the commands necessary for communicating with common MUAs. Finally, the REST endpoint developed as a means to execute usual push and pop operations and, unlike POP3 and SMTP interfaces, support search operations over mailbox data. The REST interface has some interesting properties regarding REST widespread usage as web service interface, this is, as a web service interface REST could be used by existing web mail services as backend storage and so offering TSKY-TMS as a service to existing applications. The REST endpoint was developed using Jersey RESTful library[3] that communicates with an RMI middleware interface in order to execute the core operations directly. In this way the overhead introduced by the use of a Java servlet running is eliminated in tests where search operations are out of scope, this is, the REST service runs only when tests involving search operations are needed.

4.1. Storage Module

As stated before at the lower end of the middleware architecture a storage module, responsible for managing a set of storage adapters provides to the above layer basic data store, retrieval and deletion operations. Were developed 8 adapters: One local, that stores the data directly in the hard disk (in a desired location); one to Dropbox file storage service, using Dropbox's API library[4]; two adapters for Google Cloud Storage and Amazon S3 using jets3t library[5]; one adapter for LunaCloud Storage Service using Amazon S3 SDK[6]; one for Nirvanix using the provided Java SDK[7]; one for Rackspace using JClouds library[8]; and finally one for Microsoft Azure Storage Service using Azure SDK for java.[9]

[3] Available at: https://jersey.java.net/ [Accessed 22/Sept/2013]
[4] Available at: https://www.dropbox.com/developers/core/sdks/java [Accessed 19/Sep/2013]
[5] Available at: https://jets3t.s3.amazonaws.com/index.html [Accessed 19/Sep/2013]
[6] Available at: https://aws.amazon.com/sdkforjava/ [Accessed 19/Sep/2013]
[7] Available at: http://developer.nirvanix.com/ [Accessed 19/Sep/2013]
[8] Available at: https://jclouds.incubator.apache.org/ [Accessed 19/Sep/2013]
[9] Available at: http://www.windowsazure.com/en-us/downloads/ [Accessed 19/Sep/2013]

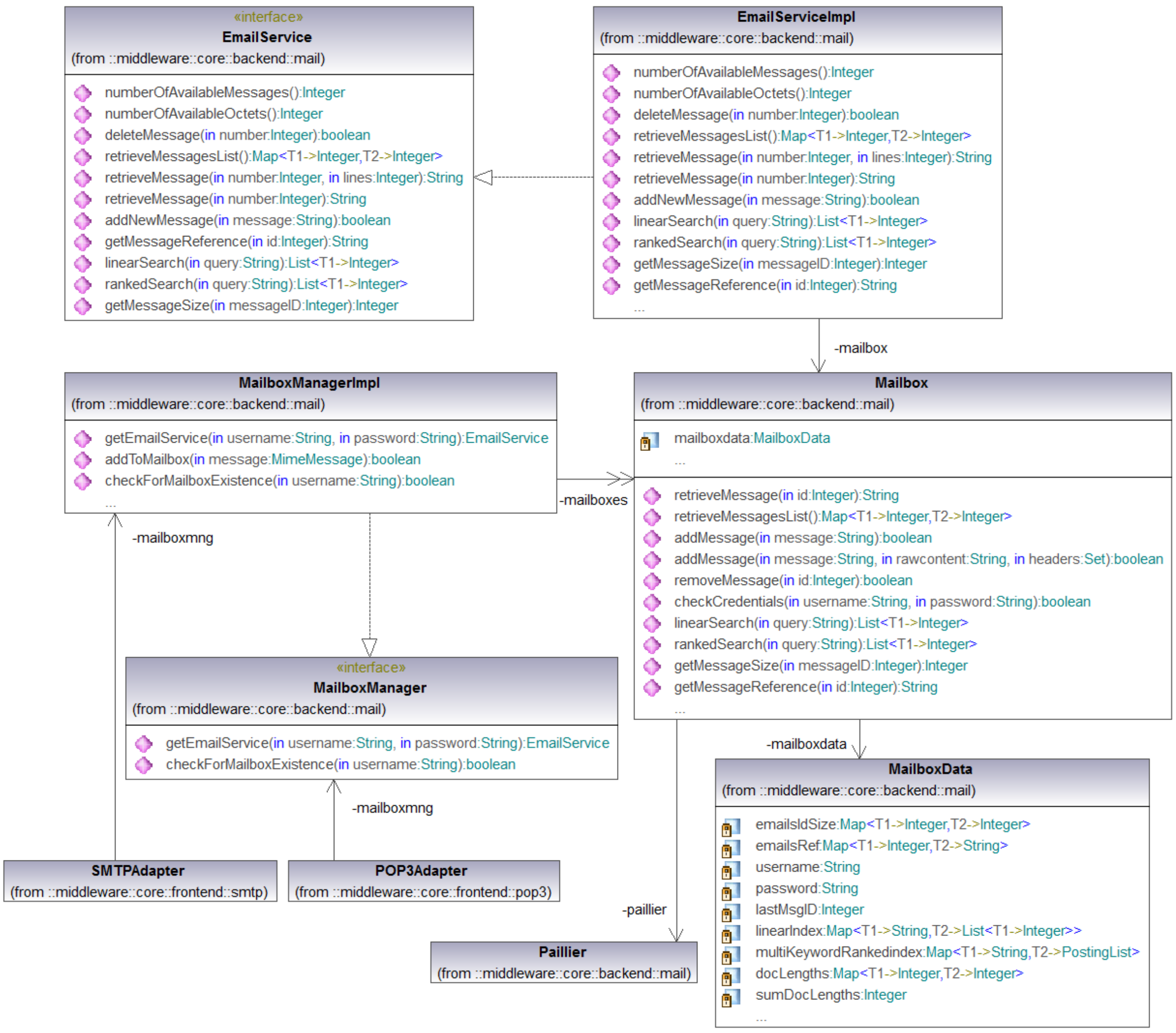

«interface»
EmailService
(from ::middleware::core::backend::mail)
numberOfAvailableMessages():Integer
numberOfAvailableOctets():Integer
deleteMessage(in number:Integer):boolean
retrieveMessagesList():Map<T1->Integer,T2->Integer>
retrieveMessage(in number:Integer, in lines:Integer):String
retrieveMessage(in number:Integer):String
addNewMessage(in message:String):boolean
getMessageReference(in id:Integer):String
linearSearch(in query:String):List<T1->Integer>
rankedSearch(in query:String):List<T1->Integer>
getMessageSize(in messageID:Integer):Integer
EmailServiceImpl
(from ::middleware::core::backend::mail)
numberOfAvailableMessages():Integer
numberOfAvailableOctets():Integer
deleteMessage(in number:Integer):boolean
retrieveMessagesList():Map<T1->Integer,T2->Integer>
retrieveMessage(in number:Integer, in lines:Integer):String
retrieveMessage(in number:Integer):String
addNewMessage(in message:String):boolean
linearSearch(in query:String):List<T1->Integer>
rankedSearch(in query:String):List<T1->Integer>
getMessageSize(in messageID:Integer):Integer
getMessageReference(in id:Integer):String
...
-mailbox
MailboxManagerImpl
(from ::middleware::core::backend::mail)
getEmailService(in username:String, in password:String):EmailService
addToMailbox(in message:MimeMessage):boolean
checkForMailboxExistence(in username:String):boolean
...
-mailboxes
Mailbox
(from ::middleware::core::backend::mail)
mailboxdata:MailboxData
...
retrieveMessage(in id:Integer):String
retrieveMessagesList():Map<T1->Integer,T2->Integer>
addMessage(in message:String):boolean
addMessage(in message:String, in rawcontent:String, in headers:Set):boolean
removeMessage(in id:Integer):boolean
checkCredentials(in username:String, in password:String):boolean
linearSearch(in query:String):List<T1->Integer>
rankedSearch(in query:String):List<T1->Integer>
getMessageSize(in messageID:Integer):Integer
getMessageReference(in id:Integer):String
...
-mailboxmng
«interface»
MailboxManager
(from ::middleware::core::backend::mail)
getEmailService(in username:String, in password:String):EmailService
checkForMailboxExistence(in username:String):boolean
-mailboxmng
-mailboxdata
MailboxData
(from ::middleware::core::backend::mail)
emailsIdSize:Map<T1->Integer,T2->Integer>
emailsRef:Map<T1->Integer,T2->String>
username:String
password:String
lastMsgID:Integer
linearIndex:Map<T1->String,T2->List<T1->Integer>>
multiKeywordRankedindex:Map<T1->String,T2->PostingList>
docLengths:Map<T1->Integer,T2->Integer>
sumDocLengths:Integer
...
SMTPAdapter
(from ::middleware::core::frontend::smtp)
POP3Adapter
(from ::middleware::core::frontend::pop3)
-paillier
Paillier
(from ::middleware::core::backend::mail)

Figure 13: Mailbox Layer Class diagram

The Frontend or applicational context layer provides the platform for Mail User Agents or MUAs to communicate with the middleware itself. This layer (as stated earlier) defines SMTP and POP3 standard interfaces, provided by `SMTPAdapter` and `POP3Adapter` (as defined in Figure 13). Additionally the `EmailService` is available remotely through RMI service, so other applications could be easily adapted to invoke over-mailbox operations. As stated before, the REST interface, needed to test search operations was developed by the means of the RMI provided interface. `EmailService` interface provides all email messages handling operations (retrieve, delete and search). When an operation request arrives from SMTP or POP3 adapters an `EmailService` instance

(`EmailServiceImpl`) is created for a designated mailbox (sender or recipient mailboxes). The `EmailService`, after a simple authentication mechanism (`checkCredentials`), can manage the mailbox directly. The Mailbox class contains all mailbox management logic while `MailboxData` contains the mailbox actual data. The mailbox data includes user authentication data, email messages identifiers along with search indexes and the references for the cloud objects (objects that store in-cloud replicas' references). The detachment between mailbox logic and data enables the use of `MailBoxData` as a data container that can be serialized and stored. In the current implementation the `MailBoxData` is stored locally in a trustable machine (along with middleware running instance). As future work we propose the use of the cloud itself to store such objects.

The `MailBox` defines three indexes, one reference index, used to get the in-cloud `CloudObject` reference given a unique email identifier, and two search indexes, a linear index over the header contents (sender, recipient, date and subject) interesting to execute rapid search operations. The second search index is a Multikeyword Ranked search index that uses Pallier partial homomorphic scheme, allowing searching operations over message contents retuning the results sorted by relevance.

4.3. Core Modules

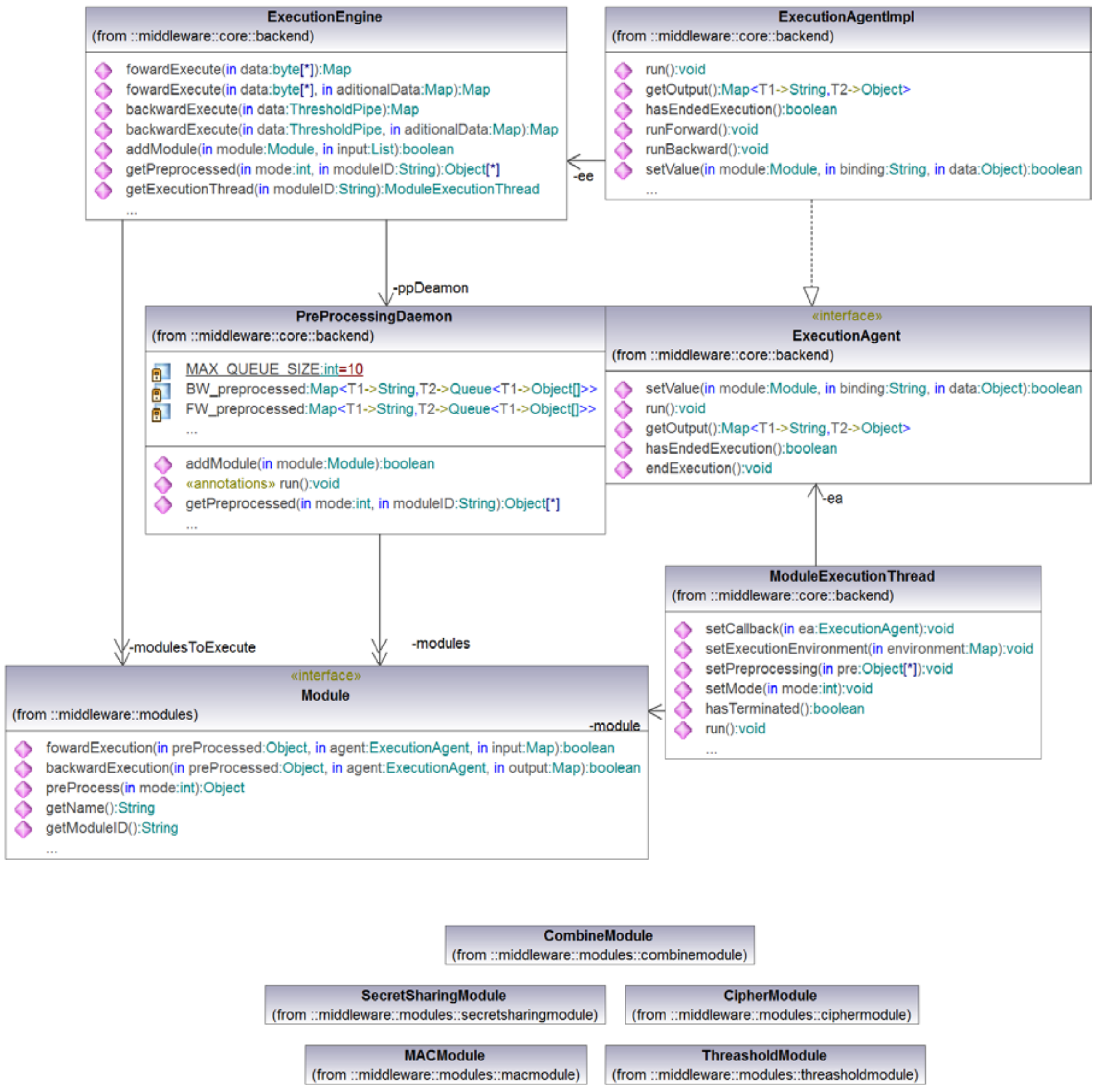

ExecutionEngine
(from ::middleware::core::backend)
fowardExecute(in data:byte[*]):Map
fowardExecute(in data:byte[*], in aditionalData:Map):Map
backwardExecute(in data:ThresholdPipe):Map
backwardExecute(in data:ThresholdPipe, in aditionalData:Map):Map
addModule(in module:Module, in input:List):boolean
getPreprocessed(in mode:int, in moduleID:String):Object[*]
getExecutionThread(in moduleID:String):ModuleExecutionThread
...
-ee
ExecutionAgentImpl
(from ::middleware::core::backend)
run():void
getOutput():Map<T1->String,T2->Object>
hasEndedExecution():boolean
runForward():void
runBackward():void
setValue(in module:Module, in binding:String, in data:Object):boolean
...
-ppDeamon
PreProcessingDaemon
(from ::middleware::core::backend)
MAX_QUEUE_SIZE:int=10
BW_preprocessed:Map<T1->String,T2->Queue<T1->Object[]>>
FW_preprocessed:Map<T1->String,T2->Queue<T1->Object[]>>
...
addModule(in module:Module):boolean
«annotations» run():void
getPreprocessed(in mode:int, in moduleID:String):Object[*]
...
«interface»
ExecutionAgent
(from ::middleware::core::backend)
setValue(in module:Module, in binding:String, in data:Object):boolean
run():void
getOutput():Map<T1->String,T2->Object>
hasEndedExecution():boolean
endExecution():void
-ea
ModuleExecutionThread
(from ::middleware::core::backend)
setCallback(in ea:ExecutionAgent):void
setExecutionEnvironment(in environment:Map):void
setPreprocessing(in pre:Object[*]):void
setMode(in mode:int):void
hasTerminated():boolean
run():void
...
-modulesToExecute
-modules
«interface»
Module
(from ::middleware::modules)
-module
fowardExecution(in preProcessed:Object, in agent:ExecutionAgent, in input:Map):boolean
backwardExecution(in preProcessed:Object, in agent:ExecutionAgent, in output:Map):boolean
preProcess(in mode:int):Object
getName():String
getModuleID():String
...
CombineModule
(from ::middleware::modules::combinemodule)
SecretSharingModule
(from ::middleware::modules::secretsharingmodule)
CipherModule
(from ::middleware::modules::ciphermodule)
MACModule
(from ::middleware::modules::macmodule)
ThreasholdModule
(from ::middleware::modules::threasholdmodule)

Figure 14: Core Layer class diagram

As we can verify in Figure 14, all core modules implement a common Module interface that defines a set of main operations:

- **Preprocessing** – This operation enables performance improvements by executing the context independent operations, i.e., slower and time consuming data independent operations could run before the arrival of the actual data. For instance, encryption/decryption keys could be generated before the data is ready to be ciphered;

- **Forward Execution** – This operation is executed when the data needs to be processed on arrival from the upper layers (middleware frontend or modules). For instance, the data needs to be encrypted or signed or the encryption keys' seed need to be splitted in multiple secret shares;
- **Backward Execution** – This operation is executed on data retrieval operations arriving from the upper layers (middleware frontend or modules). For instance, the data needs to be decrypted or the signature needs to be verified, or a set of secret shares need to be combined in order to recover the keys' seed.

When the middleware is instantiated a `PreProcessingDeamon` is executed creating modules preprocessed data. For each executed operation, the `ExecutionEngine` creates an `ExecutionAgent` responsible by running the modules, wrapped in `ModuleExecutionThreads`. The base of TSKY framework is composed by a set of dependable modules, providing distinct guarantees:

Threshold Module - Developed using existing source code[10], this module implements the RSA threshold signature algorithm as defined in the article [35]. The module was created defining two stages, a preprocessing stage, where the dealing phase occurs and a signing phase in which the key shares are used to sign the designated data, in this case, plain email messages. This module, running in forward mode, receives the data as input returning a parameterized number of shares accordingly with the desired threshold along with a public key. When running in backward execution mode, this module receives a set or subset of signature shares, the public key along with the plain data in order to verify data integrity and authenticity;

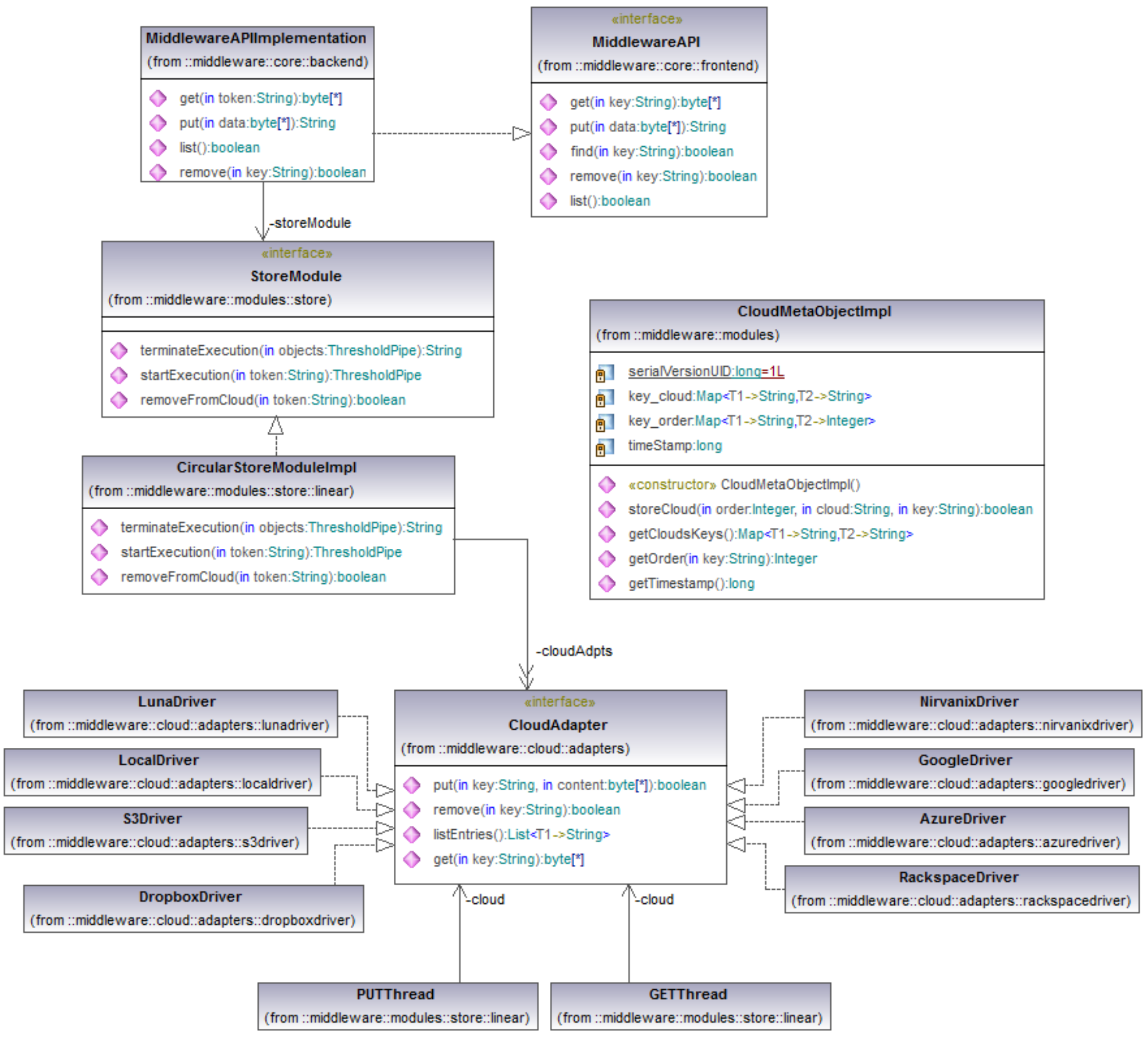
Secret Share Module – Developed from scratch, this module in forward execution receives as input data bytes and splits them into a parameterized number of shares, accordingly to the desired threshold. The used secret sharing scheme can be parameterized to use various secret sharing algorithms namely Shamir, Blakley or Asmuth-Bloom Secret Sharing Schemes. In backward execution this module receives a set or subset of non-corrupt shares and returns the original data/secret;

[10] Available at: http://code.google.com/p/threshsig/ [Accessed 29/Sept/2013]

Cryptographic Module – Developed from scratch and using JavaSE and Bouncy Castle security provider, this module can be parameterized to use common cryptographic encryption/decryption algorithms with different block ciphers and key sizes. For the executed tests, we have used Bouncy Castle provider and AES256 standard encryption algorithm;

Combining module – This module unifies all parts (encrypted data, metadata, keys, signatures) present in the data model into single data blocks that are then passed to the storage that perform a replicated linear storage over the multiple storage clouds.

4.4. Cloud Storage Layer



«interface»
MiddlewareAPI
(from ::middleware::core::frontend)
get(in key:String):byte[*]
put(in data:byte[*]):String
find(in key:String):boolean
remove(in key:String):boolean
list():boolean
MiddlewareAPIImplementation
(from ::middleware::core::backend)
get(in token:String):byte[*]
put(in data:byte[*]):String
list():boolean
remove(in key:String):boolean
-storeModule
«interface»
StoreModule
(from ::middleware::modules::store)
terminateExecution(in objects:ThresholdPipe):String
startExecution(in token:String):ThresholdPipe
removeFromCloud(in token:String):boolean
CircularStoreModuleImpl
(from ::middleware::modules::store::linear)
terminateExecution(in objects:ThresholdPipe):String
startExecution(in token:String):ThresholdPipe
removeFromCloud(in token:String):boolean
CloudMetaObjectImpl
(from ::middleware::modules)
serialVersionUID:long=1L
key_cloud:Map<T1->String,T2->String>
key_order:Map<T1->String,T2->Integer>
timeStamp:long
«constructor» CloudMetaObjectImpl()
storeCloud(in order:Integer, in cloud:String, in key:String):boolean
getCloudsKeys():Map<T1->String,T2->String>
getOrder(in key:String):Integer
getTimestamp():long
-cloudAdpts
«interface»
CloudAdapter
(from ::middleware::cloud::adapters)
put(in key:String, in content:byte[*]):boolean
remove(in key:String):boolean
listEntries():List<T1->String>
get(in key:String):byte[*]
LunaDriver
(from ::middleware::cloud::adapters::lunadriver)
LocalDriver
(from ::middleware::cloud::adapters::localdriver)
S3Driver
(from ::middleware::cloud::adapters::s3driver)
DropboxDriver
(from ::middleware::cloud::adapters::dropboxdriver)
NirvanixDriver
(from ::middleware::cloud::adapters::nirvanixdriver)
GoogleDriver
(from ::middleware::cloud::adapters::googledriver)
AzureDriver
(from ::middleware::cloud::adapters::azuredriver)
RackspaceDriver
(from ::middleware::cloud::adapters::rackspacedriver)
-cloud
-cloud
PUTThread
(from ::middleware::modules::store::linear)
GETThread
(from ::middleware::modules::store::linear)

Figure 15: Cloud Storage Layer class diagram

The cloud storage layer handles all adapters as independent storage supports. Each adapter must implement an interface offering a PUT, GET and REMOVE operation receiving as parameter a token and/or data, depending on the operation. The storage process is done by this layer is a linear basis, without any location or cost context. The use of context based storage will be addressed as profiling techniques in future work section.

As defined in Data Model section the multiple replicas pointers are stored in a, so named, Cloud Object (`CloudMetaObjectImpl`), this object is created every time a PUT request is performed and it is interpreted for every GET request performed. This object keeps track of the cloud location and identifiers of each replica.

4.5. Algorithms

In order to provide the set of security services defined as objectives of the current system (TSKY-TMS system), we have carefully designed two algorithms, one intended to store data in dependable way distributed by the multiple repositories and the other intended to retrieve such data.

```
Algorithm 1: PUT Operation
  Input: $DATA$
1 begin
2     $newIndexEntries \longleftarrow processAndIndex(DATA)$;
3     $HomomorphicEncrypt(newIndexEntries, PaillierKey, SearchKey)$;
4     $seed \longleftarrow random()$;
5     $keyGenerator \longleftarrow KeyGenerator(seed)$;
6     $TSS \longleftarrow generateThresholdSignatureShares(DATA)$;
7     $SSS \longleftarrow generateSecretSharingShares(seed)$;
8     $cloudObject \longleftarrow CloudObject()$;
9     for $i \leftarrow 1$ to $|C|$ do
10        $K_i \longleftarrow keyGenerator.next()$;
11        $DATA'_i \longleftarrow encrypt(DATA, K_i)$;
12        $replica_i \longleftarrow DATA'_i||TSS_i||SSS_i||TSS.PubKey$;
13        $RRef_i \longleftarrow SHA1(replica_i)$;
14        $cloudObject.Add(RRef_i, i)$;
15    end
16    $masterKey \longleftarrow keyGenerator.next()$;
17    $cloudObject' \longleftarrow encrypt(cloudObject, masterKey)$;
18    $masterRef \longleftarrow SHA1(cloudObject')$;
19    for $c_i \in C$ do
20        $c_i.PUT(RRef_i, replica_i)$;
21        $c_i.PUT(masterRef, cloudObject')$;
22    end
23    $storeReferenceIndex(masterRef, masterKey)$;
24 end
```

Algorithm 1: Send message (data store operation)

When a message is sent (or injected in the TMS) through the SMTP or REST endpoints, it is delivered to the mailbox manager that is charged of extracting all the words from message body and attachments inserting them into the search index. The parse of email data is enabled by Java Mail [56] while the extraction of attachment' content is made by Apache Tika [57] (document data extractor) and Apache Lucene [58] (prepares document content), in this way we could provide search capability over email headers and content including some attachments. Once the indexing is done the mailbox manager perform a PUT operation request to the layer below, with all the message data. Then the core proceeds as described in Algorithm 1, returning a master key and reference that is inserted into reference index within mailbox layer and stored locally on disk for persistency purposes. Each of encrypted email replicas along integrity proofs are then stored in a set of different storage clouds.

```
Algorithm 2: GET Operation
   Input: $masterRef, masterKey$
   Output: $DATA$
 1 begin
 2     $cloudObject' \longleftarrow c_x.GET(masterRef) : x \in C$;
 3     $cloudObject \longleftarrow decrypt(cloudObject', masterKey)$;
 4     for $i \leftarrow 1$ to $K$ do
 5         $replica_i \longleftarrow c_i.GET(RRef_i)$;
 6         if $(SHA1(replica_i)! = RRef_i)$ then
               // corrupted replica
               // ignore replica
 7         else
               // continue
 8         end
 9         $TSS_i \longleftarrow replica_i.TSS$;
10         $SSS_i \longleftarrow replica_i.SSS$;
11         $DATA'_i \longleftarrow replica_i.DATA'$;
12         $TSS.PubK \longleftarrow replica_i.TSS.PubKey$;
13     end
14     $seed \longleftarrow recoverSSSecret(SSS)$;
15     $keyGenerator \longleftarrow KeyGenerator(seed)$;
16     foreach $DATA'_i$ do
17         $DATA_i \longleftarrow decrypt(DATA', keyGenerator.next())$;
18         $isValidData \longleftarrow checkTSScheme(DATA_i, TSS, TSS.PubK)$;
19         if $(isValidData)$ then
               // return valid DATA
20             $DATA = DATA_i$;
21         else
               // continue
22         end
23     end
       // unable to recover valid DATA
24 end
```

Algorithm 2: Receive message (data retrieval operation)

When a message fetch is requested via the POP3 or REST endpoints the request is forwarded to the mailbox manager, detaining all the data necessary to recover the message from the storage clouds. Once the mailbox manager obtains the master keys and references from the index associated with a particular user mailbox, the mailbox manager invokes a GET operation over core layer, providing a master reference and key needed to retrieve the message. The core layer then proceeds as described in Algorithm 2, requesting the data from the multiple storage clouds, applying the operations used in Algorithm 1 in reverse order and returning the plain email to the mailbox manager which in turn returns it to the POP3 and REST endpoints.

The data processing flow (order of execution), including a set of cryptographic mechanisms, is justified by a set of security properties we wanted to attain:

- The use of symmetric encryption using different keys (generated by the same seed) provide data confidentiality while eliminates the possible correlation attacks that may occur (vaguely probable). Nevertheless, the process of using different keys, introduces a tradeoff related to the encryption process overhead (further discussed in evaluation section), furthermore the inexistence of any similarity between blocks excludes the possibility of using Byzantine agreement protocols at storage level. Though, the use of Byzantine agreement protocols is out of scope in the current thesis;
- The use of threshold signatures provide unforgeable authentication and integrity proofs (based on RSA) while reduces the asymmetric keys management issue. Alternatively MAC integrity proofs were used and tested (see evaluation section), such proofs can be forgeable if the attacker can get the access to the key locally stored in middleware secondary memory (hard disk), emphasizing the key management issue;
- The use of secret sharing schemes over key generation seed offers not only, the zero correlation factor between the multiple generated shares, as reduces the local key management overhead while offering seed resilience. Initially was intended to use such techniques (secret sharing) over the actual data, but preliminary results (see evaluation section) revealed that for larger secrets such schemes are unpractical;
- The replicated storage of the encrypted data block along with signature verification key, signature and seed shares across multiple clouds ensure data retrievability and availability under Byzantine faults.

5 Evaluation

In this section will be presented a system overall evaluation and an evaluation performed to each component individually. The presented evaluation consists in a security analysis of the multiple components and their performance tradeoffs. Initially will be presented the performances of the multiple cloud storage providers using raw random data without any additional processing, i.e., only latency and bandwidth was taken into account in the presented values. Secondly will be presented the performances of the individual modules, in order to determine whether an asynchronous execution framework makes the difference or if there is space for further improvements. Finally will be presented performance metrics of the overall solution using multiple storage clouds along with a critical analysis about the middleware system practicality in a real life situation, where a client requesting remote operations are waiting for on-time responses. The evaluation of the system has been made in various phases, starting by the testing of each module individually seeking for performance bottlenecks.

5.1. The TSKY Framework

The TSKY-TMS middleware was developed by the means of TSKY framework. This framework provides contextual and execution support for a set of data processing modules. The framework explores a threshold pipe model, in which data is available as soon it is acquired and processed (as a producer consumer model). For instance, since clouds have different data retrieval latencies (GET operations) and since a threshold is defined as soon as the sufficient number of replicas is retrieved the data processing can proceed to the next phases. Furthermore, as soon as one data block is processed it can proceed to the next processing stage. Although not very significant in the architecture, the use of pipe based processing can be useful when each replica stores data that can be independently processed.

The TSKY framework provides an execution context to the multiple modules and cloud adapters by means of a well-formed XML file. This configuration file includes authentication material for all cloud storage adapters and module specific configurations like threshold values or cryptographic algorithms. Furthermore, the configuration file contains IO parameters of each module defining the data processing flow. In current implementation, no verifications are made regarding the restrictions imposed by a data processing flow (all input/output data must complete the flow and all inter-dependencies must be respected), so such issue will be addressed in future framework improvements.

5.2. Test execution environment

All the presented tests where made using the same machine, so both clients and middleware where instantiated locally and so no latency can be observed or accounted from clients to the middleware itself. The machine running such tests has the following characteristics:

- Operating System: Microsoft Windows 8 64bits (Java 7 x64 JVM)
- CPU: Intel Core i7-3630QM @ 2.40GHz (4x Cores Laptop)
- RAM: 16 GB

The tests performed on individual components where made using random data bytes (generated on demand) and for the overall system (including indexing mechanisms), the Enron Email Dataset was used instead[11]. The performed tests were performed in-campus using the eduroam academic network access points.

[11] Available at: https://www.cs.cmu.edu/~enron/ [Accessed 22/September/2013]

5.3. Cloud Provider performance benchmarks (PUT/GET)

The first test, as shown in Figure 16, presents the performance times of data insertions into each a set of the cloud storage providers (**GS**-Google Cloud Storage, **S3**-Amazon S3, **NV**-Nirvanix, **RC**-Rackspace, **LN**-Luna Cloud, **DB**-Dropbox and **AZ**-Microsoft Azure). The presented times are the mean of 10 iterations runs of the PUT operation with random data blocks with sizes of 1, 10 and 100 Megabytes. The random data generation time is not included in the presented times.

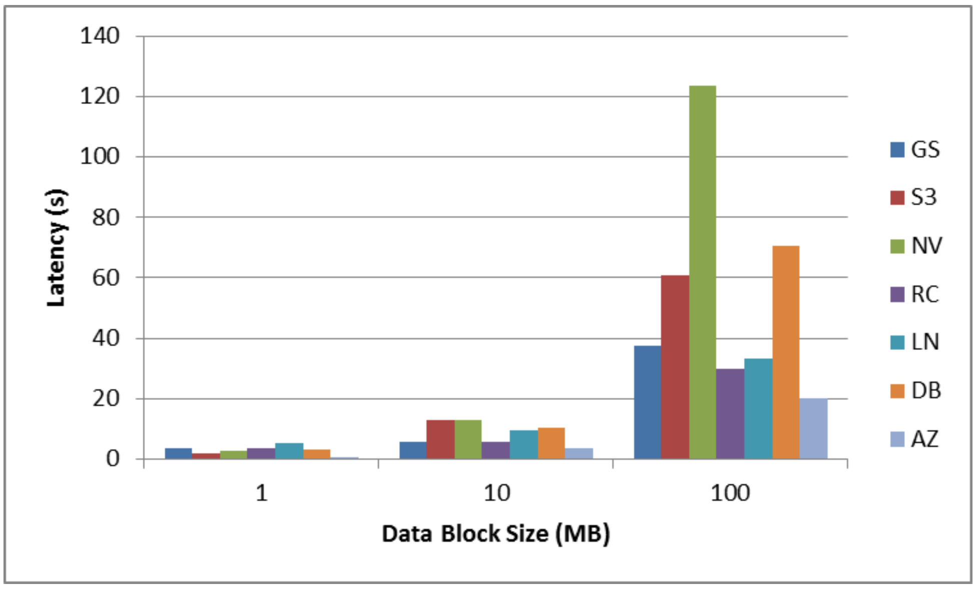


Figure 16: Performance graphic of PUT operation (in seconds and megabytes)

	GS	S3	NV	RC	LN	DB	AZ
1	3.7869	1.8136	2.7357	3.6998	5.2686	3.3417	0.7133
10	5.9312	12.8307	13.1439	5.8583	9.7202	10.2313	3.5994
100	37.6226	60.8223	123.4802	29.8275	33.4254	70.791	20.2205

Table 2: Performance table of PUT operation (in seconds and megabytes)

In the Table 2 and Figure 16 we verify that with file size growing the throughput times increase significantly. With a 100MB store/upload/put operation we verify that Microsoft Azure accomplishes the best performance followed by Rackspace, Luna Cloud and Google Cloud Storage performances. We can verify increasing fluctuations regarding the data block sizes, this demonstrates that for larger data blocks the overhead introduced by the set of chosen providers is determinant in overall latencies. Furthermore, small data blocks transfers, for instance 1MB in size, have similar latencies showing that bandwidth is determinant is such operations. Through some off-record tests we verify that over-cloud operation latencies can vary on timely basis (different hours throughout the day). The diversity imposed by the providers data centers locations along with variable bandwidth availability provide an interesting environment to further investigate profiling techniques combined with ubiquity factors which are addressed as future word (see future work section).

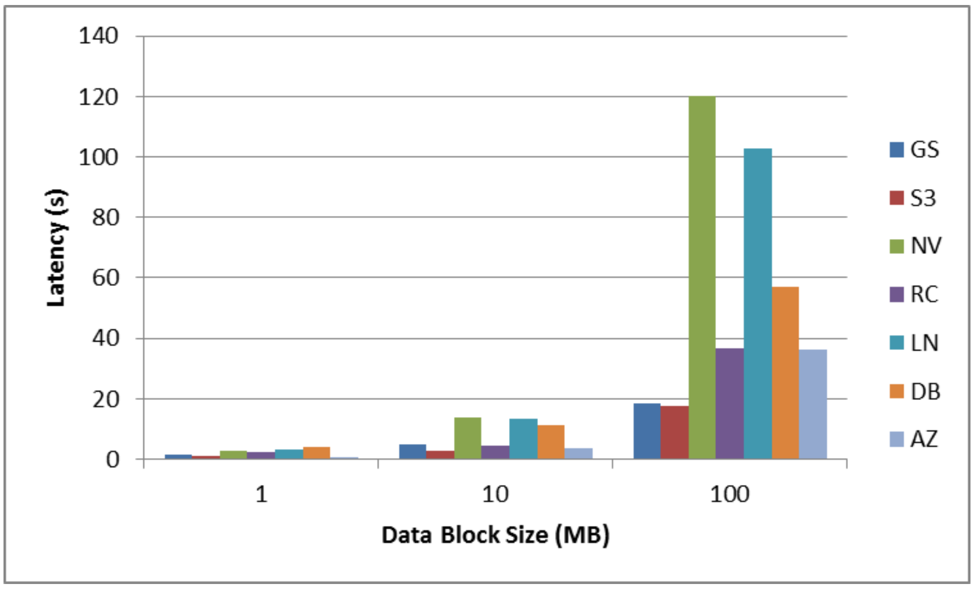


Figure 17: Performance graphic of GET operation (in seconds and megabytes)

	GS	S3	NV	RC	LN	DB	AZ
1	1.6062	1.2431	2.6679	2.4339	3.1359	4.0688	0.729
10	4.7695	2.909	13.6336	4.3534	13.371	11.4045	3.7341
100	18.6261	17.6095	120.1477	36.6935	102.9719	57.2228	36.4715

Table 3: Performance table of GET operation (in seconds and megabytes)

By contrast in retrieval/download/get operations Amazon S3 accomplishes the best performances followed by Google, Rackspace and Microsoft Azure. These differences can be backed up by the use of different semantics used by the different providers. Nevertheless, and in similar way as PUT operation latencies, GET operation latency times have big fluctuations among the multiple providers opening a new research direction towards profiling techniques (as stated for PUT operation). Comparing both PUT and GET operations we verify that, although Nirvanix provider has the most significant latencies in both cases, Microsoft provider (for example) has the one of the most significant latency in GET operation but not in PUT operation, so the correlation between both graphics is not significant (Figure 16 and Figure 17). This weak correlation between both operations' latencies can be used as factor to determine the best set of providers. For instance, applications with large write operations throughput could benefit from a service with low PUT latency. Other fact we can observe in both GET and PUT operation times is the relation between data block size and latency. Although the block sizes are 10 times superior in each run, the latency is always inferior to 10 times the larger block size. This fact can be justified by the end-to-end latency combined with datacenter data allocation and storage latencies.

Provider	IP Address	Location	Latency
Google	173.194.45.12	United States, Mountain View, 94043	21ms
Amazon	178.236.6.225	Ireland	-
Nirvanix	208.84.100.17	United States, San Diego, 92122	47ms
Rackspace	174.143.184.158	United States, San Antonio, 78218	157ms
Microsoft	168.63.3.46	Netherlands, Amsterdam	-
LunaCloud	176.111.111.248	Portugal	9ms
Dropbox	107.22.161.187	United States, Ashburn	-

Table 4: Addresses, probable location and latency of cloud storage providers

Other than the physical characteristics of the providers' datacenters, facts like end-to-end bandwidth and latency can justify the throughput values presented above. End-to-end bandwidth and latencies (obtained via ping tool) are directly related with the data centers locations and so, in Table 4 we can verify the location of the different datacenters (obtained via IP tracing). For some providers were not possible to obtain latencies due to the zero responses made to ping requests (ICMP protocol). Although in Portugal, we verify that Luna Cloud get poor performances in GET operations compared with other worldwide providers which show that physical location and latency do not determine univocally the best provider in terms of performance.

Other observed fact is that although Dropbox uses Amazon services to provide their own service, in the performed traces, the logs show in resolved addresses that Dropbox service is provided by a computational instance (Amazon EC2) located in the US rather than by Amazon storage service (Amazon S3) which possibly indicates some further data processing actions, like encryption, replication or even indexing. Furthermore in PUT operations Amazon and

Dropbox manifest similar performances despite of location and customer-provider relationship between them. Creating a comparison, between our middleware complemented by some file system oriented applicational layer versus the service provided by Dropbox, we verify some divergences in contrast with Dropbox's service like the use of multiple storage providers and the total service auditability.

Again we can use probable location combined with ping latencies and operation latencies to get profiling metrics to use in profiling techniques determining the set of providers that best fit not only the location of the middleware instances as best fit the target application needs.

5.4. Threshold Module Benchmark

This subsection presents the test results and the evaluation on threshold signatures module that as stated in related work section provides unforgeable integrity and authenticity proofs (based on RSA) on a threshold basis. The module preprocessing stage is defined by an initialization phase, in which structures are initialized (**Init**) and a key share generation phase (**Gen**). In forward execution stage (or signing stage) the data is actually signed (**Sig**) resulting in a set of signature shares. Finally in backward execution stage (or signature verification stage) a set of signature shares are combined resulting in a valid signature that is verified (**Ver**) using a public key (generated in preprocessing stage). The performed tests present the mean times over 100 iterations for each block size (1, 10 and 100 megabytes) in milliseconds.

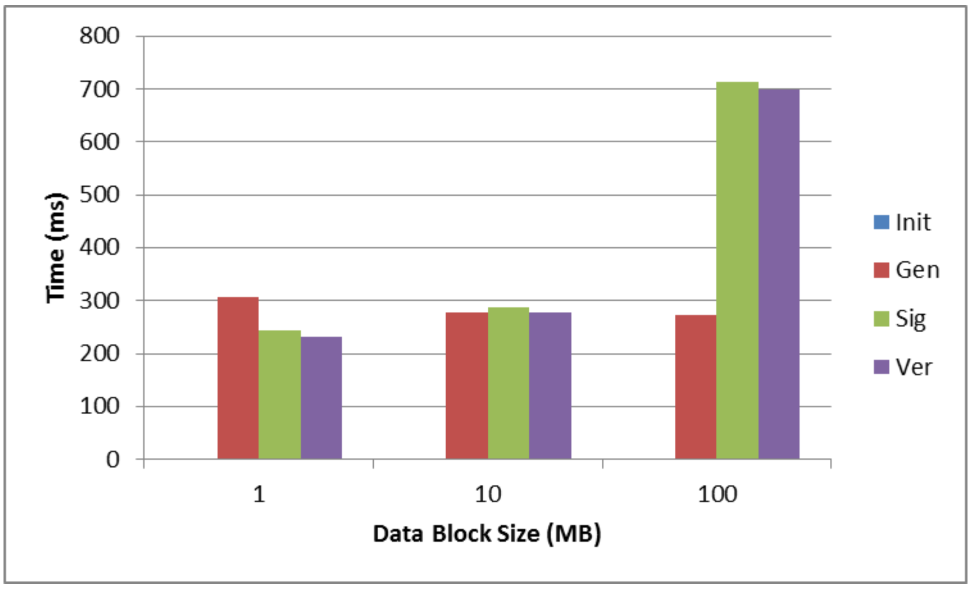


Figure 18: Performance of Threshold Signature Module in different stages

	Init	Gen	Sig	Ver	Total(w/o Ver)
1	0.66	305.47	244.59	232.63	550.72
10	0.02	278.25	286.65	277.65	564.92
100	0.03	272.32	714.33	699.63	986.68

Table 5: Performance of Threshold Signature Module in different stages

As we can verify in the Figure 18 and Table 5 the performance times for key generation, signing and signature verification are similar. The impact of key generation's times is the motivation behind the use of a preprocessing mechanism, inherent to the framework itself. In other words, all modules have to implement a common interface that includes a preprocessing method that executes independently of and concurrently with the middleware invoked operations. In this case we verify that, as preprocessed keys are available the processing time is reduced by approximately 1/3 comparing with a completely sequential execution. For instance, taking into account the availability of preprocessed keys (dealing/key generation phase), for 1MB of data the overall threshold signature time would be reduced from 550.72 to 244.59 milliseconds.

Although in milliseconds, and so not much significant, the times in Table 5 would be much more significant (not tested, addressed as future work) if thousands or even millions of clients store email with attachments data. The signature verification process does not have any preprocessed data associated though the times are similar to signing times due to the absence of key generation. The signing and signature verification operations are significantly increased by the size of the messages to sign due to inner SHA1 hash operation over the message content. These times could be reduced using compression techniques that compress the data before the signing and encryption process.

5.5. Secret Sharing Module Benchmark

This parameterized module, allows the definition of three different secret sharing schemes. Each scheme uses different mathematical approaches and different algorithms. In Figure 19 and Table 6 we verify the performances of each scheme representing the mean times over 20 iterations of initialization times plus share creation time and secret recovery time, detailed in Table 7.

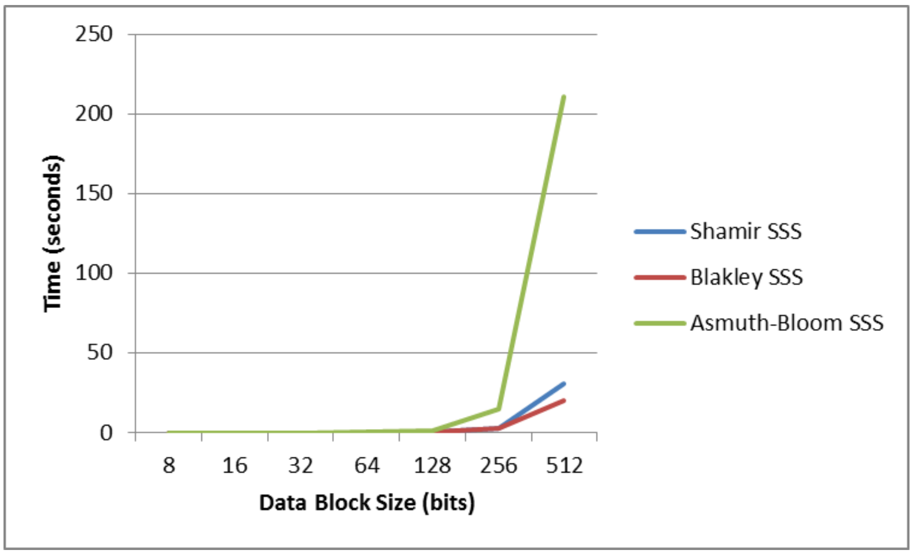


Figure 19: Performance of Secret Sharing Module using multiple schemes

	Shamir SSS	Blakley SSS	Asmuth-Bloom SSS
8	0.0048	0.0068	0.0092
16	0.00635	0.0081	0.0166
32	0.0074	0.0125	0.0375
64	0.0204	0.033	0.1465
128	0.16585	0.18385	1.16735
256	2.43775	2.66345	15.07435
512	31.11145	20.49685	210.79985

Table 6: Performance of Secret Sharing Module in seconds/bits

Both Table 6 and Figure 19 present the initialization times along with time needed to split a secret into a set of shares plus the time needed to reconstruct such secret. As stated before secret sharing schemes are very timely efficient regarding small secrets. As presented in Table 6 and as illustrated in Figure 19 secrets up to 128 Bytes (1024 bits) present acceptable performances. Values superior to 256 Bytes are unpractical on on-demand situations. These values are justified by the use of numerical (modular, arithmetic and matrixes) operations used in the implementation of each scheme. As discussed before and to mitigate such overhead introduced by larger data the use of secret sharing schemes is used at key level instead of data level.

#Bytes	Shamir SSS			Blakley SSS			Asmuth-Bloom SSS		
	Init	Gen	Rec	Init	Gen	Rec	Init	Gen	Rec
8	1.3	2.55	0.95	1.35	3.9	1.55	1.35	6.85	1
16	1.3	4.3	0.75	1.1	5.5	1.5	1.15	14.4	1.05
32	1.3	5.45	0.65	1.05	9.6	1.85	1.3	35.4	0.8
64	1.2	18.5	0.7	1	29	3	1.1	144.6	0.8
128	1	164.05	0.8	1.1	175.05	7.7	1.1	1165.25	1
256	1.2	2436.15	0.4	1.2	2638.65	23.6	1.1	15071.65	1.6
512	1.3	31109.55	0.6	1.1	20411.25	84.5	1.25	210794.5	4.1

Table 7: Detailed performance of Secret Sharing Module in milliseconds/bits

Comparing the three schemes we verify that the most efficient regarding secret share generation and secret recovery is Blakley scheme. Despite this fact, the performances between Shamir and Blakley schemes are similar and the secret recovery times are higher in Blakley scheme, furthermore and more important by security analysis we can argue about the properties of each scheme as defined in related work section, this is, Shamir Scheme is securer in a theoretical vision of the problem. In the analysis of the results we verify that no advantage is taken from preprocessing process, furthermore, we verify that secret recovery stage is much performance efficient than secret share generation time. Analyzing the introduced overhead we verify that in order to be acceptable on the TSKY-TMS solution (on-demand solution), the use of such schemes must be confined to small secrets, for instance 128 bits or smaller. For such reason the use of a seed as secret, instead of the generated keys or even the actual data, presents a viable solution since the generated seed size could be adjusted in order to improve performance times or on behalf of security of the mechanism itself.

5.6. Encryption Module Benchmark

In order to verify the overhead introduced by the use of symmetric encryption in the current solution performance test, tests presented in Figure 20 and in Table 8, were made using as base AES256 in CBC mode. Furthermore, and in order to detect possible further performance improvements other symmetric encryption algorism were benchmarked. To do the testing we have used random data blocks with sizes of 1, 10 and 100 Megabytes. The values presented in the following graphics and tables present average performance times made in 100 iterations.

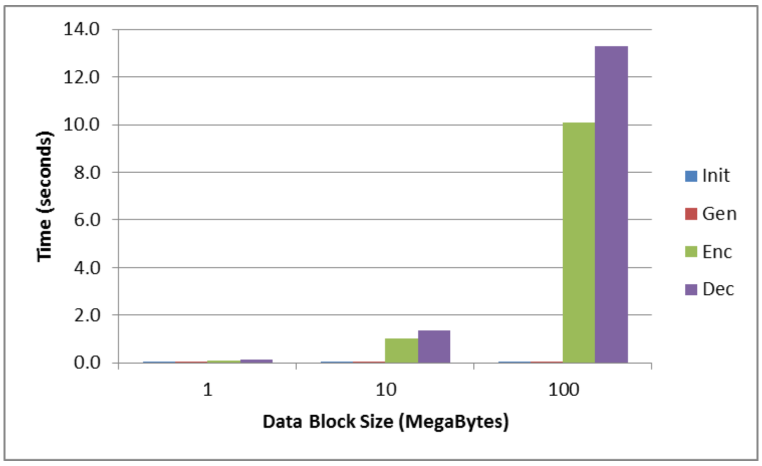


Figure 20: Performance of Encryption Module using AES256 in seconds/Megabytes

	Init	Gen	Enc	Dec
1	0.00424	0.00138	0.10821	0.14128
10	0.00109	0.00008	1.03016	1.36002
100	0.00110	0.00011	10.07862	13.27140

Table 8: Performance of Encryption Module using AES256 in seconds/Megabytes

In this subsection we present the encryption performance times comparison for different data block sizes and verifying that, as expected and as shown in Figure 20 and Table 8 in a symmetric encryption algorithm, the key generation times are completely negligible and so no further improvements can be done regarding the module itself. Furthermore, we can verify that decryption process is slower (although not significant). Nevertheless, other symmetric encryption algorithms were tested and compared on performance basis as presented below.

	AES256	3DES	Blowfish256	Blowfish448
1	0.24157	0.8045	0.2636	0.27018
10	2.22752	7.98716	2.52205	2.57867
100	22.69284	80.27249	25.17275	25.668

Table 9: Performance comparison (in seconds) symmetric encryption algorithms

As presented in Table 9 AES is the most efficient encryption algorithm. Although not standard, Blowfish algorithm has similar performances and has been considered to be secure. Concluding, the AES is a standard algorithm with proved efficiency and so we have no reason to adopt Blowfish or even the old 3DES that has proven weak in terms of security and poor in terms of performance.

5.7. Overall Solution Benchmark

To demonstrate system usability, we have performed a test that simulates a real use case scenario using Amazon S3, Nirvanix Cloud Storage, Rackspace Cloud Files and Google Cloud Storage as storage services to TSKY-TMS. The TMS service has run in a local machine (laptop) using wireless communications (eduroam), as defined in evaluation environment subsection. This subsection presents a brief evaluation of TSKY-TMS service through a set of end user performance tests. These tests basically consist in the extraction of performance metrics from an email client, sending and receiving messages to and from the presented system. Each message received by middleware is redundantly stored in the four different storage clouds, as defined earlier in this subsection. The tests were divided in multiple iterations: first sending and receiving 10 messages, 100 messages, 1000 messages and finally 10000 messages. Due to limitations in Gmail service it was impossible to conduct tests with more than 1000 messages. The used subset of email messages taken from Enron online dataset contains messages from 1Byte to 200Kbytes with plain text content. The conducted tests allowed us to reason about service acceptance compared with today wide spread well known email services.

The overall (TSKY-TMS) solution benchmark includes not only execution times of all the above modules through the evaluation dataset defined in test execution environment subsection as includes times introduced by POP3 and SMTP message processing. Table 10 presents three types of metrics: the endpoint metrics consisting in the latency observed by end user (like in Gmail service); the core execution metrics include message processing time, indexing and cryptographic operations. The endpoint and core performance in message receiving case include over-cloud operations due to the synchronous needs, i.e., unlike in message send operations, in message retrieval operations the system needs to retrieve and process the messages from the clouds before returning them to the client. Still, in message send case, the endpoint and core performance do not include over-cloud operation times since once the message is delivered to the system the endpoint return success to email client. The cloud performance times represent the operation execution latency over the slowest cloud (due to parallel cloud requests).

# Msgs	Test Type	GMAIL		TMS-MAC		TMS-TS	
		Send	Receive	Send	Receive	Send	Receive
10	Endpoint	18,108	1,327	1,44	13,5	2,55	18,97
	Core	-	-	0,456	13,455	1,606	18,932
	Clouds	-	-	20,2696	13,31	23,7148	18,063
100	Endpoint	187,521	12,989	10,48	131,09	23,12	133,48
	Core	-	-	1,106	130,992	11,727	133,422
	Clouds	-	-	467,188	130,288	235,289	126,702
1 000	Endpoint	1821,059	127,396	663,92	1240,33	734,76	1502,42
	Core	-	-	6,578	1240,258	76,344	1502,433
	Clouds	-	-	2694,173	1235,933	1845,449	1437,302
10 000	Endpoint	-	-	4812,64	12709,34	15302,61	13439,83
	Core	-	-	53,934	12726,06	1182,538	13506,49
	Clouds	-	-	13605,91	12685,67	13741,3	12848

Table 10: Performance (in seconds) of Gmail versus TMS (MAC and TS)

Through a brief observation of the Table 10 we verify that Gmail service accomplishes worse times in message sending operations on the endpoint point of view. This fact can be backed by a synchronous message delivery mechanism held by Google service. Although we cannot take this fact as a performance advantage, today's email services include message receipts which tell users the delivery state of their messages. In the presence of a synchronous send message operations it would be expected worse times as seen in Figure 21 where the overall time is 10% to 150% times worse, mainly due to over-cloud operations. Nonetheless the presented times can be considered acceptable, given that 1 to 5 seconds to send an email is acceptable.

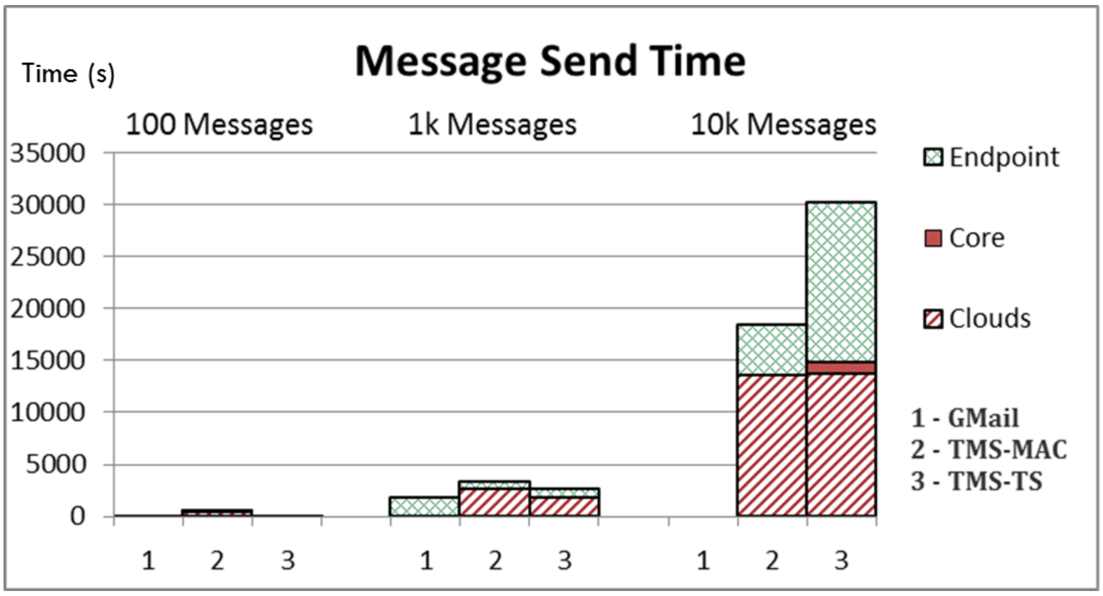


Figure 21: Message send times to GMAIL service, TMS-MAC and TMS-TS

On the other side we verify that email receiving times are worse in our system. The message deliver time is basically determined by over-cloud operations (Figure 22) taking about 99% of all time needed to deliver a message to end client. This time could be dramatically reduced by using better low latency cloud storage services, caching techniques or using the cloud just has a redundant support for storing email data. The overall times are 10 to 15 times worse than the ones involved in Gmail service.

Other verifiable matter is the fact that endpoint latency in message send operations of TMS-MAC versus TMS-TS substantially differ (Figure 21). This fact can be justified by the waiting compass imposed not only by the over-cloud operations as by threshold signatures process. As in TMS-MAC, in TMS-TS the waiting compass was artificially introduced in such way that all requests could be attended without jeopardize the system response capabilities.

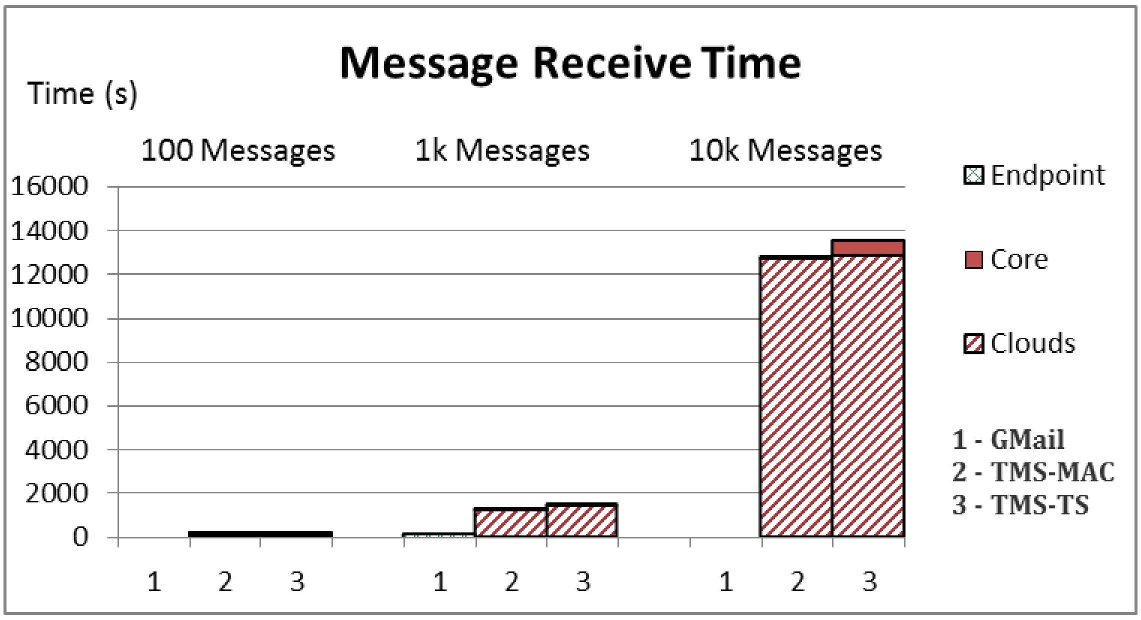


Figure 22: Message receiving times to GMAIL service, TMS-MAC and TMS-TS

Comparing TMS-TS, in which of Threshold Signatures schemes were used, with TMS-MAC approach, in which Message Authentication Codes were used as data integrity proofs, we notice that for message sending operations we get core processing times 3 to 35 worse. Despite this the message receiving times are just up to 1.5 worse in TMS-TS which can be considered acceptable.

Unlike in the synchronous receive operation, the considerable endpoint performance times in message sending operations exists due to an artificially added constraint so the TMS service would not drown under a large number of requests.

Excluding the endpoint times in message send operations and comparing both messages send and receiving times (Figure 21 and Figure 22) we verify that both operations attain similar performances. Note that graphics have different scales, and so while sending 10000 messages takes up 9.5 hours receiving the same messages will take only 4.5 hours.

5.8. Cloud Based Proxy Solution Benchmark

To evaluate the overall solution, we have performed a complementary test using two cloud-based virtual machines running the TSKY-TMS middleware platform. The tests were performed using two distinct Amazon EC2 micro instances running in different locations (Ireland and California, USA). The performed tests have consisted in sending/receiving 1000 email messages to and from the TSKY-TMS middleware, using the POP3 and SMTP endpoints. In these tests messages were sent from one EC2 instance (running in the Amazon Data Center in Ireland) to the other instance (running in the USA – California Data Center). Such complementary tests have revealed interesting results, as shown in the following figure.

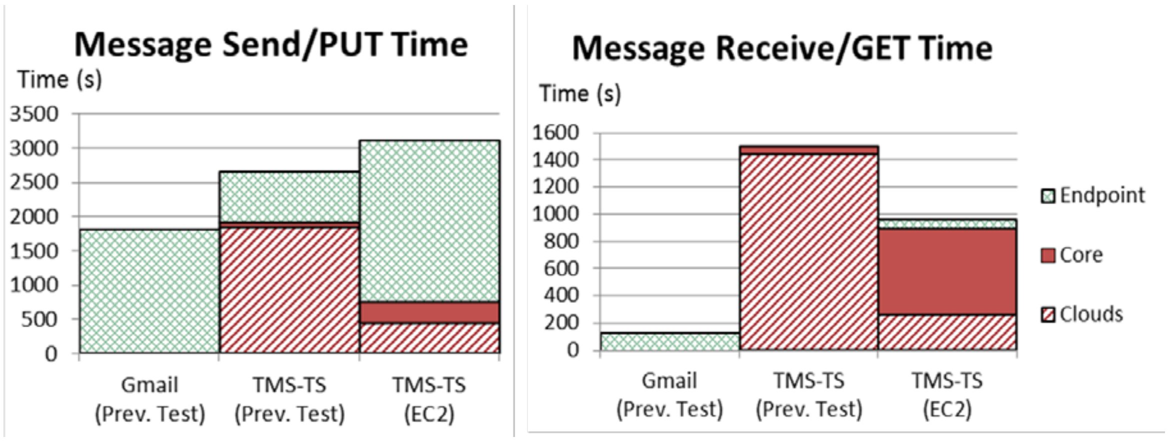


Figure 23: Operations latency with TMS-TS running in a computational cloud

The observed latencies for message sending and storing are similar to the previous tests, as represented in Figure 21 and Figure 22. However we verify that the overhead introduced by the over-cloud operations were substantially reduced. On the other hand, as expected, the client observed latencies are substantially increased due to the remote access between the two data centers, namely the latency between the email user-agent (client) to the TSKY-TMS middleware instance. Finally, we also verify worse performances regarding core processing metrics, namely memory access and processing performance. This fact is certainly due to the low-end hosting characteristics inherent to the used Amazon EC2 micro instance, running in a virtual machine with only one virtual core, and 0.6GB of available memory. This computing power when compared with the processing and memory resources of the local machines used in the tests presented in the section 5.7).

Regarding the observed latencies for message retrieving we verify better performances comparing with previously made tests and again, reduced latencies concerning over-cloud operations. Unlike in the case of message send operations, message retrieving operations exhibit a more significant overhead in core processing times. Such fact can be backed up by the limited amount of computational power available combined with the maximization of the CPU usage and message transmission and reception rates. For message sending observations we induced a transmission delay between messages, in order to guarantee system reliability. The used time delay in this case was 25ms. This delay is required because in the scope of the TSKY-TMS implementation and the focus of the thesis contributions, we did not consider the implementation of specific handling services for high-concurrency control, memory buffering or asynchronous batch processing and queuing optimization to deal with high transmission rates and high volumes of send/receive requests. Then, the delay parameterization was a simple and pragmatic mechanism to optimize the system for the evaluation purposes.

Concluding, we verify that the use of TSKY-TMS, running in a computational cloud, leverages the performance of the over-cloud operations whose are considered the bottleneck of the overall performance, as observed in the previous tests. Furthermore, such performance improvement motivates a rational for the use of location awareness and operation profiling mechanisms, in order to further reduce the cloud operations overheads.

6

Conclusion and Future Work

Conclusions

As the cloud computing and storage solutions provide interesting characteristics and costs, the corporations tend to deploy new applications and data or migrate existing applications and data to such platforms, nevertheless some issues are raised when the criticality of the data and applications is taken into account. Some current events [23], [25], [26] and [52] have shown that cloud based solutions suffer from lack of user control, lock-in, availability and security issues namely in terms of privacy.

The main objective of the current dissertation was to design and implement a dependable middleware system/service capable of managing data stored in such public cloud storage assuring a set security properties over the stored data. This is, the stored data (considered critical and/or sensitive) should be stored and kept private, corruption free, authentic and available. Such objectives seem to be incompatible to the above considerations (public cloud storage solutions raise various security issues). To assure such properties in such hostile environments TSKY middleware made use of multiple cryptographic mechanisms along with replication process eliminating the dependency of each cloud storage provider. This is, it is assumed that no passive or active collusive attacks can occur between multiple storage providers.

We have based our solution in a combination of parts of existing solutions along with novel approaches in order to assure a set of well-defined security properties. Components implementing symmetric crypto, secret sharing schemes, index with homomorphic properties or replication are present in various state-of-the-art systems, like DepSky [11] or iDataGuard [12] (further described in related work section). As means to accommodate such components we have developed an extensible framework (TSKY framework) supporting, not only various data processing modules, as supporting a set of cloud adapters, capable of communicating with multiple cloud providers' services.

As mean to test the TSKY storage service and framework an email repository service, named TMS, was developed. This service stores private email data across multiple untrusted cloud storage repositories. The stored data is signed using threshold signatures resulting in a set of signature shares (or alternatively MAC integrity proofs), then data is encrypted using different keys (generated from the same seed) resulting in distinct data encrypted blocks. In turn, the seed, used to deterministically generate the encryption keys, is splitted in shares using a secret sharing scheme. Finally an encrypted data replica along with a signature share and a seed share is stored in distinct clouds. This process or data flow of storing data in the various clouds is defined in a configuration file, and has future work execution flow can exchange by the addition of new modules or the replacement of existing ones.

The evaluation of the TSKY-TMS implementation has led to promising results regarding the overall system performance. Namely we have verified that, although TSKY-TMS get worse performance times comparing with a well-known wide spread email service, TSKY-TMS get times that could be considered acceptable, in respect to system usability (1 to 5 seconds to receive/send a message). Furthermore, our evaluation has provided good metrics regarding cloud storage providers services enabling further optimizations, in order to reduce such overhead introduced by over-cloud operations.

Future work directions

To further research, in the use of multiple untrusted data repositories, and with main focus in the development of TSKY framework a set of future work matters can be addressed in different relevant dimensions related with this dissertation results:

- Further evaluation of the current TSKY-TMS implementation with more extensive test benches, involving larger data volumes and higher operations throughput, searching for possible refinements and optimizations in the current design specifications and implementation. For further testing and evaluation a webmail based application could be designed in order to get more reliable usability metrics and to do a more consistent comparison with existing webmail services;
- Support and use of multiple middleware instances in order to test system scalability issues. The use of multiple instances, supported by a concurrency support mechanisms will allow the deployment of the TSKY-TMS system enabling new test benches regarding ubiquity and performance issues;
- Inclusion of multiple components and mechanisms as certificates (as means to authentication of multiple middleware instances and users), convergent encryption, garbage collection (to reduce storage costs) and corruption detection mechanisms (Proof of Data Integrity and Retrievability);
- Design a concurrency support model, tested using multiple users evoking operations over same (or different) data. The designed model could include low contention locking mechanisms for multiple instances case or local concurrency control for single instance. The use of low contention locking mechanisms could raise security issues regarding the platforms used to store such locks. Such concurrency solution can follow the initial approach as stated in [11]. Alternatively an approach using reliable distributed coordination (using distributed coordination tools or frameworks), as in [14], could be used;
- Implement and test a larger set of cloud storage providers making use of profiling techniques to determine best usage heuristics (based on cost

model per data volumes, cost per types of operations, communication latencies or other metrics related with provided guarantees). To address this problem, one possible idea is to start by implementing and oracle component, modeling and providing the dynamic behavior of the different used clouds, using the following metrics: latencies, bandwidth, data storage/operation costs or physical location in a location awareness system. The same oracle can also, complementarily, provide input profiling information for specific applications candidate to run on top of the TSKY middleware. Such a component can benefit from available monitoring information, about some metrics and indicators of different cloud-storage services, over time;

- Use of caching, compression and erasure coding techniques as profiling complements to reduce cloud operation latencies and costs;
- Redesign TSKY-TMS to run as a service in a computational cloud (TSKY-TMS as service). Refine implementation to support out-of-the-box solutions using different combinations of the existing modules and implementation of a system generalization model, as initially anticipated in the system model characterization (section 3). In this direction, a revised work of the current system model and architecture must address the possible migration of parts of TSKY framework to run on cloud as a service, as a work dimension directly related with the previous one;
- Relocation of indexing capabilities to the TSKY service itself, i.e., generic text indexing capabilities, as well as, the integration of complementary homomorphic encryption algorithms, for the possible integration of similarity searches, involving text and multimedia contents and information retrieval support.

References

[1] Cisco, "Global Mobile Data Traffic Forecast," Cisco, 2013.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia, "A View of Cloud Computing," *Communications of the ACM,* vol. 53, no. 4, pp. 50-58, 2011.

[3] Y. Chen and R. Sion, "To Cloud Or Not To Cloud Musings On Costs and Viability," *Proceedings of 2nd ACM Symposium on Cloud Computing,* 2009.

[4] EMC, "The Cloud Service Provider Report, issue 21," 2012.

[5] K. R. Choo, "Trends and Issues in Crime and Criminal Justice," Australian Institute of Criminology, 2010.

[6] G. Melvin, "Survivability and information assurance in the cloud," *Proceedings of the 4th Workshop on Recent Advances in Intrusion-Tolerant Systems (WRAITS'10),* pp. 194 - 195, 2010.

[7] TClouds, "Project TCLOUDS – trustworthy clouds - privacy and resilience for Internet-scale critical infrastructure," [Online]. Available: http://www.tclouds-project.eu/. [Accessed 23 September 2013].

[8] F. Rocha, S. Abreu and M. Correia, "The Final Frontier: Confidentiality and Privacy in the Cloud," *IEEE Computer,* vol. 44, no. 9, pp. 44-50, 2011.

[9] N. Santos, R. Rodrigues and B. Ford, "Enhancing the OS against security threats in system administration," *ACM/IFIP/USENIX 13th International Middleware Conference,* pp. 415-435, 2012.

[10] N. Santos, R. Rodrigues, K. P. Gummadi and S. Saroiou, "Building Trustworthy Cloud Services with Excalibur," *21st USENIX Security Symposium (USENIX Security '12),* 2012.

[11] A. Bessani, M. Correia, B. Quaresma, F. André and P. Sousa, "DepSky Dependable and Secure Storage in a Cloud-of-Clouds," *EuroSys '11 Proceedings of the sixth conference on Computer systems,* pp. 31-46, 2011.

[12] R. C. Jammalamadaka, R. Gamboni, S. Mehrotra, K. Seamons and N. Venkatasubramanian, "iDataGuard An Interoperable Security Middleware for Untrusted Internet Data Storage," *Proceedings of the ACM/IFIP/USENIX Middleware'08 Conference Companion,* pp. 36-41, 2008.

[13] K. D. Bowers, A. Juels and A. Oprea, "HAIL: A High-Availability and Integrity Layer for Cloud Storage," *Proceedings of the 16th ACM conference on Computer and communications security,* pp. 187-198, 2009.

[14] H. Abu-Libdeh, L. Princehouse and H. Weatherspoon, "RACS: A Case for Cloud Storage Diversity," *Proceedings of the 1st ACM symposium on Cloud computing,* pp. 229-240, 2010.

[15] "Amazon S3," [Online]. Available: https://aws.amazon.com/s3/. [Accessed 19 September 2013].

[16] "Google Cloud Storage," [Online]. Available: https://cloud.google.com/products/cloud-storage. [Accessed 19 September 2013].

[17] "Nirvanix Public Cloud Storage," [Online]. Available: http://www.nirvanix.com/products-services/cloudcomplete-public-cloud-storage/index.aspx. [Accessed 19 September 2013].

[18] "Rackspace Cloud Files," [Online]. Available: http://www.rackspace.com/cloud/files/. [Accessed 19 September 2013].

[19] R. Choubey, R. Dubey and J. Bhattacharjee, "A Survey on Cloud Computing Security, Challenges and Threats," *International Journal on Computer Science and Engineering,* vol. 3, no. 3, pp. 1227-1231, 2011.

[20] Cloud Security Alliance, "Expanded Top Ten Big Data Security and Privacy Challenges," 2013.

[21] "Controlling Data in the Cloud Outsourcing Computation without Outsourcing Control," *Proceedings of the 2009 ACM Cloud Computing Security Workshop,* 2009.

[22] Hitachi Data Systems, "How to Improve Healthcare with Cloud Computing," 2012.

[23] Engadget, "Gmail accidentally resetting accounts, years of correspondence vanish into the cloud?," 27 February 2011. [Online]. Available: http://www.engadget.com/2011/02/27/gmail-accidentally-resetting-accounts-years-of-correspondence-v/. [Accessed 19 September 2013].

[24] PCworld, "Hotmail Data Loss Reveals Cloud Trust Issues," 3 January 2011. [Online]. Available: http://www.pcworld.com/article/215365/hotmail_data_loss_reveals_cloud_trust_issues.html. [Accessed 19 September 2013].

[25] Sophos, "Many Amazon S3 cloud storage users are exposing sensitive company secrets, claims report," 29 March 2013. [Online]. Available: http://nakedsecurity.sophos.com/2013/03/29/amazon-s3-cloud-storage-data-leak/. [Accessed 19 September 2013].

[26] TheGuardian, "NSA Prism program taps in to user data of Apple, Google and others," 7 June 2013. [Online]. Available: http://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data. [Accessed 19 September 2013].

[27] A. Shamir, "How to Share a Secret," *Communications of ACM,* vol. 22, no. 11, 1979.

[28] K. Kaya, S. A. Aydın and Z. Tezcan, "Threshold Cryptography Based on Asmuth-Bloom Secret Sharing," 2007.

[29] I. N. Bozkurt, K. Kaya, A. A. Selc and A. M. Güloglu, "Threshold Cryptography Based on Blakley Secret Sharing," *Information Sciences,* 2008.

[30] B. Ferreira and H. Domingos, "Searching Private Data in a Cloud Encrypted Domain," *Proceedings of the 10th International Conference in the RIAO (OAIR 2013),* 2013.

[31] IEEE Computer Society Publications, Multimedia Information Extraction, John Wiley & Sons, Inc., 2012.

[32] S. Heinz and J. Zobel, "Efficient Single-Pass Index Construction for Text Databases," *Journal of the American Society for Information Science and Technology,* vol. 54, no. 8, pp. 713-729, 2003.

[33] C. D. Manning, P. Raghavan and H. Schütze, An Introduction to Information Retrieval, Cambridge: Cambridge University Press, 2009.

[34] K. S. Jones, S. Walker and S. E. Robertson, "A probabilistic model of information retrieval: development and comparative experiments," *Information Processing and Management,* vol. 36, no. 6, pp. 779 - 808, 2000.

[35] V. Shoup, "Practical Threshold Signatures," *EUROCRYPT'00,* pp. 207-220, 2000.

[36] J. Cowling, D. Myers, B. Liskov, R. Rodrigues and L. Shrira, "HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance," *7th USENIX Symposium on Operating System Design and Implementation,* 2006.

[37] J. Sousa and A. Bessani, "From Byzantine Consensus to BFT State Machine Replication: A Latency-Optimal Transformation," *Ninth European Dependable Computing Conference,* pp. 37-48, 2012.

[38] L. Tavernini, "Lucio Tavernini Home Page," 15 August 2011. [Online]. Available:

http://tavernini.com/tablet_notes/2013_fall/mat_3633.001/mat3633note01.pdf. [Accessed 19 September 2013].

[39] R. Martin, "Introduction to Secret Sharing Schemes".

[40] W. Stallings, "The Chinese Remainder Theorem," in *Cryptography and Network Security Principles and Practices*, Prentice Hall, 2011, pp. 254-257.

[41] A. J. Menezes, P. C. Oorschot and S. A. Vanstone, "Secret Sharing," in *Handbook of Applied Cryptography*, 1996, pp. 524-528.

[42] M.-S. Hwang and T.-Y. Chang, "Threshold Signatures: Current Status and Key Issues," *International Journal of Network Security,* vol. 1, no. 3, pp. 123-137, 2005.

[43] C. Gentry, "Fully Homomorphic Encryption using Ideal Lattices," *Proceedings of the 41st annual ACM symposium on Theory of computing,* pp. 169-178, 2009.

[44] . M. van Dijk, . C. Gentry, . S. Halevi and V. Vaikuntanathan, "Fully Homomorphic Encryption over the Integers," *29th Annual International Conference on the Theory and Applications of Cryptographic Techniques,* pp. 24-43, 2010.

[45] R. A. Popa, C. M. S. Redfield, N. Zeldovich and H. Balakrishnan, "CryptDB: Protecting Confidentiality with Encrypted Query processing," *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles,* pp. 85-100, 2011.

[46] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer and R. P. Wattenhofer, "Farsite: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment," *Proceedings of the 5th symposium on Operating systems design and implementation,* 2002.

[47] S. Narayan, M. Gagné and R. Safavi-Naini, "Privacy Preserving EHR System using Attribute-Based Infrastructure," *Proceedings of the 2010 ACM workshop on Cloud computing security workshop,* pp. 47-52, 2010.

[48] K. P. N. Puttaswamy, C. Kruegel and B. Y. Zhao, "Silverline: Toward Data Confidentiality in Storage-Intensive Cloud Applications," *Proceedings of the 2nd ACM Symposium on Cloud Computing,* 2011.

[49] A. A. Ucla, A. Avizienis, J.-c. Laprie and B. Randell, *Fundamental Concepts of Dependability,* 2001.

[50] "Microsoft Azure," [Online]. Available: http://www.windowsazure.com. [Accessed 19 September 2013].

[51] "Dropbox," [Online]. Available: https://www.dropbox.com/. [Accessed 19 September 2013].

[52] Dropbox, "Yesterday's Authentication Bug," 20 June 2011. [Online]. Available: https://blog.dropbox.com/2011/06/yesterdays-authentication-bug/. [Accessed 19 September 2013].

[53] LunaCloud, [Online]. Available: http://www.lunacloud.com/en/cloud-storage. [Accessed 19 September 2013].

[54] A. M. C. Guiomar, "T-Stratus - Confiabilidade e Privacidade com Nuvens de Armazenamento de Dados," DI-FCT-UNL, 2013.

[55] I. McFarland, J. Stevens, J. Schnitzer, E. De Oliveira and S. Hernandez, "SubEtha SMTP," [Online]. Available: http://code.google.com/p/subethasmtp/. [Accessed 21 September 2013].

[56] Oracle, "JavaMail API," [Online]. Available: http://www.oracle.com/technetwork/java/javamail/index.html. [Accessed 20 September 2013].

[57] Apache, "Apache Tika Toolkit," [Online]. Available: https://tika.apache.org/. [Accessed 20 September 2013].

[58] Apache, "Apache Lucene Project," [Online]. Available: https://lucene.apache.org/. [Accessed 22 September 2013].

Glossary

ABE or Attribute Based Encryption is a type of public key encryption in which the decryption is dependent, not only from a secret key and the correspondent cipher text , as is dependent of a set of attributes, this is, to decrypt such data user associated attributes has to match cipher text attributes.

AES or Advanced Encryption Standard also kwon as Rijndael is a block cipher adopted as standard by the US government.

API Application Programming Interface.

CA Certification Authority.

CBC Cipher Block Chaining.

EHR or Electronic Health Records is a concept of uniting a set of electronic heath information (like medical history, medication, allergies, laboratory tests and radiology images) of each and every patiant in a single record. Theoretically such records can be shared among different heath care facilities and staff in order to facilitate the access to such data.

HAIL High-Availability and Integrity Layer for Cloud Storage.

HMAC or Hash-based Message Authentication Code, can be used to verify message integrity and authenticity. The inner process of MAC generation can use any hash function as MD5 or SHA1. The security of an HMAC depends on the security of the used hash function. HMACs are used in IPSec and TLS protocols.

HTTP or Hypertext Transfer Protocol, is a application level protocol used as communication base in World Wide Web. The protocol defines eight different request methods (GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS e CONNECT). For each request a standard response code follows.

IaaS or Infrastructure as a Service, is a provision model in which the organizations outsource the equipment used to support the operations. Hardware, storage, servers and networking can be outsourced via a service provider charged of housing, running and maintaining such components.

MIME or Multipurpose Internet Mail Extensions, it is a standard that defines electronic message content format. S/MIME extend MIME standard by including cryptographic security services enabling message authentication and integrity check.

Multitenant, refers to a principle in which a same software, service is provided to multiple clients or organizations (tenants).

OEK Object Encryption Key.

PaaS or Platform as a Service, is a provision model in which the client creates/uses and deploys software to the provider premises controlling configurations. The provider may also provide facilities to application design, development and testing.

PDP Proof of Data Possession.

POP3 or Post Office Protocol is an applicational layer protocol used to remotely access electronic messages (mailbox access). Defined in RFC 1939 allow that all messages in a mailbox to be transferred sequentially to a local machine.

POR Proof Of Retrievability.

RACS Redundant Array of Cloud Storage.

RAID or Redundant Array of Independent Drives, defines a way of organizing individual physical disks in order to improve reliability or attain better performances. Standard defines RAID 1 to 6 and each standard can be recombined to form new topologies.

REST Representational State Transfer.

SaaS or Software as a Service sometimes referred as on-demand software, is a provision model in which the provider offers the client out-of-the-box, sometimes already configured and ready to use, software solutions.

SLA or Service Level Agreement, defines the contracted service deliver times including expected availability times and performances of the provided service over a certain period of time. The SLA can define some standard metrics, as for instance, mean time between failures (MTBF), mean time to repair or mean time to recovery (MTTR). These agreements also include exclusion terms safeguarding the provider from unforeseen circumstances that can lead to loss of availability or durability.

SMTP or Simple Mail Transfer Protocol, is an Internet standard electronic email transmission applicational protocol. Defined in RFC 5321 the protocol is used to send and receive email messages from MUAs (Mail User Agents) to mail servers or to exchange messages within email servers.

SSL or Secure Sockets Layer, is a standard, widely used, application layer protocol providing secure communications over the Internet. Such protocol can be used as wrapper with other known protocols, as email exchange protocols.

TCB or Trusted Computing Base, defines or delimits a set of hardware or software components as trustable in a security vision. This is, it is assumed that no vulnerabilities, bugs or attacks can occur in such components.