REPUBLIC OF NORTH MACEDONIA
UNIVERSITY "SS. CYRIL AND METHODIUS" IN SKOPJE
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Arbër Sehar Beshiri

BUILDING OF DISTRIBUTED RESTful SERVICES AND RESOURCES FOR AUTHORIZATION ISSUES

Doctoral thesis

Skopje, 2023

UNIVERSITY "SS. CYRIL AND METHODIUS" IN SKOPJE
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Arbër Sehar Beshiri

BUILDING OF DISTRIBUTED RESTful SERVICES AND RESOURCES FOR AUTHORIZATION ISSUES

Doctoral thesis

Skopje, 2023

Doctoral student:

M. Sc. ARBËR SEHAR BESHIRI

Thesis:
BUILDING OF DISTRIBUTED RESTful SERVICES AND RESOURCES FOR AUTHORIZATION ISSUES

Mentor:
Prof. Dr. ANASTAS MISHEV
University “Ss. Cyril and Methodius”
Faculty of Computer Science and Engineering
Skopje, North Macedonia

Defense Panel:
Prof. Dr. IVAN CHORBEV, Chairman
University “Ss. Cyril and Methodius”
Faculty of Computer Science and Engineering
Skopje, North Macedonia

Prof. Dr. ANASTAS MISHEV, Mentor
University “Ss. Cyril and Methodius”
Faculty of Computer Science and Engineering
Skopje, North Macedonia

Prof. Dr. SONJA FILIPOSKA, Member
University “Ss. Cyril and Methodius”
Faculty of Computer Science and Engineering
Skopje, North Macedonia

Assoc. Prof. Dr. BOJANA KOTESKA, Member
University “Ss. Cyril and Methodius”
Faculty of Computer Science and Engineering
Skopje, North Macedonia

Prof. Dr. MENTOR HAMITI, Member
South East European University
Faculty of Contemporary Sciences and Technologies
Tetovo, North Macedonia

Scientific area:
WEB SERVICES AND DISTRIBUTED SYSTEMS

Date of defense:
25.10.2023

To my extraordinary parents,

for their unwavering support, love, and motivation!

Арбер Сехар Бешири

ГРАДЕЊЕ НА ДИСТРИБУИРАНИ RESTful СЕРВИСИ И РЕСУРСИ ЗА АВТОРИЗАЦИЈА

- АПСТРАКТ -

Во комплексните системи, многу сервиси бараат пристап до споделени ресурси. Во тие случаи, минимизирањето на дуплирањето на податоците е многу пожелно. SOA го поедноставува развојот на таквите системи со воспоставување на посебни сервиси кои можат да се користат од повеќе други сервиси (апликации). REST е архитектонски стил како дел од SOA кој служи како основа за дизајнирање и градење на современи веб сервиси (апликации). Обезбедува методи за развој на дистрибуирани сервиси вообичаено наречени REST или RESTful сервиси. Еден од клучните аспекти на REST е тоа што е без зачувување на состојба, што значи дека услугата не задржува никакви информации за претходните барања. Вообичаено, кога развиваме сервис REST за одредена база на податоци, ги претставуваме сите податоци преку соодветната сервис. Во сценаријата каде што е потребна зголемена флексибилност, како што се оние кои вклучуваат авторизација и приватност, треба да се ограничи обемот на податоци до кој апликацијата (сервисот) има пристап. За да се постигне ова, врз основа на REST, се имплементира посветена сервисна точка кој може да се справи со потребното филтрирање.

Следејќи ја процедурата за автентикација, повеќето апликации базирани на веб-сервиси користат механизми за контрола на пристап специфични за апликацијата за да донесат одлуки за авторизација. Сервисите можат да комуницираат едни со други, понекогаш потпирајќи се на односи засновани на доверба. Меѓутоа, ако се добие неовластен пристап до одреден сервис, тоа потенцијално може да го загрози целиот безбедносен систем. Во RESTful сервисите, авторизацијата игра огромна улога во однос на безбедноста, така што сервисите постојано се проверуваат во однос на овластување на корисниците. Безбедноста како суштински аспект на сервисите влијае на тоа дека серверите нужно го содржат механизмот за авторизација и тие мора да ја автентицираат секоја услуга за секое нивно барање. Управувањето со дозволите за автентикација го управува посветен сервис во рамките на дистрибуираната околина. Истовремено, овој пристап го олеснува создавањето на попрецизни имплементации на сервиси бидејќи одговорноста за управување со автентикацијата се пренесува на централната служба, која ги координира сите други сервиси. Во овој труд предложена е рамка за управување со а на RESTful услуги и се опишува, имплементира, анализира и споредува можноста за управување со авторизацијата за пристап на сервисите до одредени други сервиси (ресурси). Студијата е заклучена со презентација на експериментални резултати и евалуација добиени од имплементацијата на REST сервисите врз основа на предложената рамка.

Клучни зборови*:* RESTful услуга, SOA, авторизација, дозволи за авторизација, автентикација, безбедност

Arbër Sehar Beshiri

BUILDING OF DISTRIBUTED RESTful SERVICES AND RESOURCES FOR AUTHORIZATION ISSUES

- ABSTRACT -

In systems, numerous services require access to shared resources. Ideally, under such circumstances, minimizing data duplication is highly desirable. SOA simplifies the development of such systems by establishing distinct services that can be utilized by multiple services (applications). REST is an architectural style rooted in SOA that serves as a basis for designing and building contemporary web services (applications). It provides a method for developing distributed services commonly referred to as REST or RESTful services. One of the key aspects of REST is that it is stateless, meaning that the service does not retain any information about previous requests. Typically, when developing a REST service for a specific dataset, we represent the whole data via the relevant service. In scenarios where enhanced flexibility is needed, such as those involving authorization and privacy concerns, should limit the dataset accessible by an application (service). To achieve this, based on the REST approach (style), it is implemented a dedicated service endpoint that can handle the required filtering.

Following the authentication procedure, most web services-based applications use application-specific access control mechanisms to make authorization decisions. Services can interact with one another, sometimes relying on a trust-based relationship. However, if unauthorized access is gained to a particular service, it could potentially jeopardize the whole security system. In RESTful services, authorization plays a tremendous role in terms of security, so services are constantly charged with authorizing users. Security as an essential aspect of services affects that servers necessarily contain the authorization mechanism, and they must authorize each service for each of their requests. The management of authorization permissions is handled by a dedicated service within the distributed environment. Simultaneously, this approach facilitates the creation of more streamlined service implementations as the responsibility for authorization management is transferred to a central service, which directs all other services. A RESTful services' authorization management framework is proposed and the possibility to manage service access authorization to specific services (resources) is described, implemented, analyzed, and compared. The study is concluded with the presentation of experimental results and evaluation derived from the implementation of the REST services based on the proposed framework.

Keywords: RESTful service, SOA, authorization, authorization permissions, authentication, security

ACKNOWLEDGEMENT

First of all, I want to express my deepest gratitude to God for enabling me to complete this academic cycle. I also want to express my thankfulness to my mentor, Prof. Dr. Anastas Mishev, for his suggestions, criticism, advice, and patience throughout the whole process of my Ph.D. studies and dissertation research work. Thanks to all professors and collaborators for their support and advice during the preparation of this research.

Last but not least, I want to sincerely thank my parents and family for their unconditional support, unwavering love, encouragement, and motivation! To everyone mentioned above, I express my genuine gratitude. Your influence has been immeasurable, and I am honored to have had you by my side on this journey.

Thank you to everyone!

AUTHORSHIP DECLARATION

Herein, I provide my personal statement for the originality of the stated doctoral dissertation that I duly provide a reference of the quoted sources, and that this paper is not used in any other university study for acquiring other knowledge.

Arbër Beshiri

I declare that the electronic version of the doctoral dissertation is identical to the printed doctoral dissertation.

Arbër Beshiri, p.s.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Page is intentionally left blank

1. INTRODUCTION

In recent years, the fast increase of internet-primarily based services has brought about a growing demand for secure and scalable authorization mechanisms. Authorization, as an important factor of system security, ensures that merely authenticated and authorized users are granted access to resources and services. With the advent of distributed systems and the rise of Representational State Transfer (REST (RESTful)) architecture, the need for effective authorization solutions that can seamlessly integrate with these frameworks has become more pressing.

It is absolutely crucial to provide the security of the systems that operate on the web, the messages that are exchanged, and the communication channels should also be ensured, to enable cooperation within the service-based organization. Authentication and authorization are fundamental security elements that must be met and guaranteed because they are essential for the security of web services [1]. RESTful services are a way to represent a web service that adheres to the REST architectural style. An important aspect of REST is that it is stateless. This means that the service does not keep any information about previous requests. This ensures that RESTful services can be properly authenticated and authorized.

In RESTful services, authorization usually relies on client or system authentication. Formerly the identity is confirmed via authentication, the service can utilize an access control list or role-based access control to determine whether the authenticated user (identity) has the necessary permissions to access the requested information (resources) or act as demanded. In some cases, the authorization decisions may be made by the service itself and in some cases may be delegated to an external authorization server [2].

In the context of RESTful services, authentication, and authorization are typically implemented using a combination of standard protocols (techniques). There are several different approaches to authentication and authorization in service-oriented architecture (SOA). Some common approaches include HTTP basic authentication, which involves transmitting the credentials through an HTTP header. It is not very secure because the credentials are sent in plain text; therefore it is usually only used over HTTPS. Token-based authentication operates using a token that is considered a string of characters that characterizes the authenticated user or service. A token is created by the authentication

server and sent to the user, which includes the token in following requests to the service. The service verifies the token and grants or denies access based on the consequence. It is considered a more secure method than basic authentication, as the credentials are not sent in plain text [3].

OAuth is a widely accepted authorization standard that allows clients to act in the name of the resource owner (e.g., a user (service)) and access server resources. OAuth uses a combination of tokens and secrets to authenticate and authorize client requests. A typical instance of a token used to authenticate users (services) is the JSON Web Token (JWT). It is a JSON object with the server's digital signature and information about the user (service), including the user's roles and permissions. In the context of RESTful services, JWTs are often used as a means of transmitting authenticated identity information and authorization data between systems [4].

The key insights show that the difficulties with authorization in services primarily derive from the communication between them and the difficulty in applying security measures in each service. This results in added complexity during development and expands the potential attack area, as distinct consideration should be provided for each service.

The purpose of this study is to investigate, build, and implement distributed RESTful services and resources that effectively address the challenges and complexities associated with authorization issues. By leveraging the principles of distributed computing and the REST architectural style, this research aims to develop a robust and scalable solution that enhances the security and flexibility of authorization mechanisms in a distributed environment. Specifically, in the research work is:

- Explored the existing challenges and limitations in authorization mechanisms within SOA and RESTful services, including issues related to scalability, performance, and security.
- Conducted a comprehensive review of the principles and best practices of RESTful architecture, analyzing their potential applications in addressing authorization issues.
- Proposed a framework for RESTful services and resources that provides a secure and scalable infrastructure for managing authorization across multiple nodes and organization environments.
- Developed and implemented the proposed framework, utilizing suitable technologies and protocols to demonstrate its effectiveness in overcoming common authorization challenges.

- Conducted testing and evaluation of the framework, measuring its performance, scalability, and authorization (security), and comparing it with existing authorization solutions in distributed RESTful services.
- Analyzed the results and draw conclusions regarding the effectiveness of the proposed solution, identifying its strengths, weaknesses, and potential areas for further improvement.
- Provided recommendations and guidelines for the community on the adoption and implementation of RESTful services and resources for addressing authorization issues in real-world scenarios.

The other focus of this study is on managing authorization in REST services. The purpose is to enable the declaration of more detailed and specific authorization permissions. The management of authorization permissions is handled by a dedicated service within the distributed environment. The objective is to enable the declaration of permissions and authorization workflows that can encompass multiple REST services. Simultaneously, this approach facilitates the creation of more streamlined service implementations as the responsibility for authorization management is transferred to a central service, which coordinates all other services. In this case, we have examined the fundamental principles of the REST architecture, the distinctive features of RESTful web services, and the communication mechanisms employed in various in-house services within an organization's ecosystem.

In systems of this nature, numerous services require access to shared resources. Ideally, under such circumstances, minimizing data duplication is highly desirable. SOA simplifies the development of such systems by establishing distinct services that can be utilized by multiple services (applications). Typically, when developing a REST service for a specific dataset, we represent the whole data via the relevant service. In scenarios where enhanced flexibility is needed, such as those involving authorization and privacy concerns, should limit the dataset accessible by an application or service. For example, if a service provides the students' list, it is not desirable for a university learning management system to have access to all the personal information associated with each student. To achieve this, based on the REST approach (style), one would typically implement a dedicated service endpoint that can handle the required filtering.

The study is founded on the following research objectives:

a) To review and analyze the existing literature on authentication and authorization mechanisms in SOA, with a specific focus on RESTful services.
b) To identify the key requirements for authentication and authorization mechanisms in RESTful services.
c) To develop a framework for authorization management in RESTful services that addresses the identified requirements.
d) To implement and evaluate the proposed framework using a set of RESTful services.
e) To compare and analyze the proposed framework with existing authorization mechanisms for RESTful services in terms of security, performance, scalability, and ease of implementation.
f) To identify the potential limitations and challenges of the proposed framework and provide recommendations.
g) To demonstrate the applicability of the proposed framework in real-world scenarios.

Generally, this study aims to contribute to the field of web services and distributed systems by proposing a practical and efficient approach to handling authorization challenges, ultimately enhancing the security, scalability, and performance of systems through the utilization of RESTful services and resources. The introduction section provides an overview of the research, including the underlying research objectives and contribution, motivation, aims of the research, hypotheses, and the overall structure of the thesis (in the next subsections).

1.1. Motivation

The REST architecture for creating contemporary web services has gained more attention in recent years from the web sectors and web services sectors. REST is a web architecture paradigm that Roy Fielding first described in his doctoral thesis. REST architecture is created in order to make the web more scalable and reliable as the network hypermedia system. The World Wide Web (WWW) is accelerated and developed on a broad scale by the presentation of REST. It has demonstrated unique possibilities for systems and improved simplicity, dependability, resource reuse, performance, and service flexibility [2].

Through a collection of limitations, including uniform interface development, client-server and stateless architecture, the REST architecture influences distributed systems. The advantages of REST-based systems and applications include their lightweight conditional nature, simple accessibility, enhanced visibility, and simplicity in scalability. Typically, REST systems are implemented using the Hypertext Transfer Protocol (HTTP). Compared to HTTP, REST has much more restrictions, such as the HATEOS restriction, which HTTP does not handle [5].

Most RESTful services and APIs ignore the HATEOS limitation. REST does not rely on any specific communication protocol, unlike HTTP, which is typically used in web services and APIs. Different alternatives can also be provided by the way web services are composed. The process of creating new services from already-existing web services for particular business needs that are distributed globally via the Internet is known as web service composition [6].

Newly developed web services rely on the functioning of already-existing web services. Without any additional encapsulation, the service response with input features is provided in the resource. RESTful services use HTTP protocol and its methods, therefore they are based on the uniform invoke interface. Additionally, they have a consistent collection of HTTP status codes for deciphering invoke responses, making RESTful services simply accessible to clients. The declarative technique is used to create service-oriented applications that provide more flexibility, reliability, and scalability [2], [7].

In order to meet the functionality of web services, web composition has specific needs that should be taken into consideration. The service actors are one example of this. Various actors, including humans and machines, can invoke methods in various web services as part of the web service composition. This information is required by the service developer to precisely address the service's requirements during web service implementation. Web composition is crucial for service reuse and manipulation for various purposes, especially in convoluted businesses [6], [7].

RESTful services are concerned with providing and showing resource(s), hence their development involves combining different web resources to produce new applications (resources). To enable the optimal statement of authorization, it is necessary to explicitly characterize the REST services as part of the composition process. In a distributed system with the ability for considerably

greater flexibility and simplicity across services and data transmission between services, the permission coordinator is used to coordinate authorization and permissions [8].

Authentication and authorization are essential security concerns in SOA, particularly in relation to RESTful services. Authentication is the manner of verifying the identity of a user (or service). This is often executed through the use of credentials (username and password). In a RESTful service, authentication is generally dealt with through consisting of an authentication token within the request header, which the service can use to verify the identity of the requester [7].

Authorization is the process of granting access to resources or actions based on the authenticated identity. This is often done through the use of roles or permissions. In a RESTful service, authorization is typically handled by including information about the requester's permits in the request header or by including it in the payload of the request [9].

Here are some potential motivations based on the context of SOA:

a) Ensuring secure access to resources: Resources in an SOA may be exposed by several services, and these resources need to be secured against unauthorized access. While authorization enables service providers to apply access control restrictions based on the validated identity of the client, authentication enables service providers (SP) to confirm the identity of customers requesting access to these resources. It may make sure that only authorized users or systems can access resources that are protected by implementing authentication and authorization. This aids in preserving the availability, confidentiality, and integrity of those resources.
b) Resource management: By controlling access to resources, organizations can ensure that resources are used efficiently and effectively and that unauthorized users or systems are not able to consume resources unnecessarily.
c) Maintaining separation of concerns: In an SOA, different services may be responsible for different tasks and may need to interact with each other to complete a larger operation. Services can only access the resources they are permitted to use by employing authentication and authorization, which helps maintain the division of concerns and lowers the possibility of unexpected interactions between services. By centralizing security-related functionality and

simplifying the administration of user access to resources, implementing authentication and authorization can also assist in reducing the complexity of the system [10].

d) Enabling flexible access control: In an SOA, it may be necessary to grant different levels of access to different clients or to allow clients to access resources in different ways. SPs can set up authentication and authorization systems, which enable them to create versatile access control rules. This way, they can carefully manage how various clients interact with their resources.

e) Improving the security of the overall system: By implementing robust authentication and authorization measures, SPs can help ensure the overall security of the SOA by preventing unauthorized access to resources and reducing the risk of security breaches. Implementing authentication and authorization can also enhance the user experience. It lets users access resources that matter to them while preventing access to resources they shouldn't be able to reach [7].

f) Meeting regulatory and compliance requirements: Depending on the industry and the type of data being accessed, there may be regulatory or compliance requirements that mandate the use of authentication and authorization measures to protect sensitive data. When SPs concentrate on these areas, they can make sure they're meeting the necessary standards, avoiding possible fines or legal complications.

g) Ease of use: By implementing authentication and authorization controls, it becomes easier for users to access the services and resources they need. This works because SPs can use one set of login information to access various services, instead of having to remember different usernames and passwords for each service individually [9].

Furthermore, implementing authentication and authorization measures significantly boosts the security, reliability, and dependability of the service. This essential step ensures that only authorized users or devices can access it, a critical safeguard for protecting sensitive or vital data. This measure also helps deter unauthorized access and misuse of the service [11].

1.2. Aims of the research

The research's goal is to make it possible for various services within an in-house system to communicate with each other. In these kinds of systems, numerous services should use and share common resources. In such cases, it is ideal to eliminate data duplication as much as possible. SOA makes the implementation of these systems easier by supporting the creation of various services that are used and shared by a variety of applications (or different services).

Generally, when we build a REST service for a specific data collection, we transfer the whole data using the service. Or, in cases that required more flexibility by getting into consideration authorization issues and privacy, we can restrain the dataset that has access to the service. For instance, when the students' list is returned by the service, a learning management system should be restrained in the access so that it has no information for the whole student's personal data. Based on the above-mentioned proposes and considering REST approaches, a particular service endpoint would be implemented, despite the possibility that a generic service may provide the essential filtering.

For the purpose of enabling the declaration of concise authorization, the study is focused on building RESTful services and resources for authorization issues. In the in-house distributed system, a specific service that is separated will be used to manage the authorization of the whole REST service. The objective is to offer permission (authorization) declarations and workflows that could also involve numerous REST services. Furthermore, since the authorization management is delegated to be in the specific service that coordinates whole services, this strategy enables thinner service functionalities and implementations.

1.3. Hypotheses

The study is based on the following hypotheses (H) that are explained below.

H1: Any declarative language may be used to describe the RESTful service interface as well as the implementation of RESTful service by following the recent RESTful practices and guidelines.

In order to comprehend this hypothesis, let's divide it into two distinct parts:

a) *Describing the RESTful service interface:* In the context of REST, the interface of a service defines the set of operations or endpoints that can be accessed by clients. This includes specifying the available resources, their representations, and the supported HTTP operations [12]. The hypothesis suggests that any declarative language can be employed to describe the interface. These languages aim to convey the intended result rather than giving detailed, sequential commands. Some instances of declarative languages encompass XML, JSON, YAML, and RDF. When we utilize these languages, we can describe the format and attributes of the RESTful service interface in a way that's easy for people to understand [7].
b) *Implementing the RESTful service:* Once the interface is described, the next step is to implement the service itself, which involves handling incoming requests and generating appropriate responses [13]. The hypothesis suggests that any declarative language can be employed for implementing the service while adhering to the recent RESTful practices and guidelines. In practice, setting up a RESTful service usually means crafting code that can deal with HTTP requests, oversee resources, and carry out the required actions on those resources. Though imperative languages such as Java, Python, or JavaScript are often the go-to choices for this task, some believe that declarative languages can also be employed effectively. However, it's crucial to remember that this belief hinges on the existence of the right tools, frameworks, or libraries within the chosen declarative language to help facilitate the development of RESTful services.

H2: Service resources can be built as a collection of possible HTTP requests, and they can be essentially used for authorization issues based on permissions.

In order to gain a comprehensive understanding of this hypothesis, let's divide it into two parts:

a) *Building service resources as a collection of HTTP requests:* In the context of a RESTful service, resources represent entities that can be accessed and manipulated through the API. These resources are typically identified by unique URIs [4]. The hypothesis proposes that we can build service resources by thinking about the various ways we can use HTTP requests to manage or engage with these resources. Let's take "users" as an example within a service: the range of potential HTTP requests might involve using GET requests to fetch user details, POST requests to establish new user profiles, PUT requests to modify user information, and DELETE

requests to remove user accounts. By defining the available HTTP requests for each resource, the service resource collection is built.

b) *Using service resources for authorization based on permissions:* By associating specific permissions with each resource and HTTP request combination, the service can enforce access control and determine whether a user or client is allowed to perform a particular action. We can establish these permissions by taking into account the user's role, privileges, or other relevant factors [13]. To illustrate, suppose a user has the authorization to create new user profiles; in that case, they would be given permission to use the HTTP POST request for the "users" resource. Conversely, if they don't have the necessary permission, the service will decline their request. When we structure resources and link them with their respective HTTP requests, along with assigning the right permissions, the service gains the ability to regulate and oversee access to its functions and data. It's essential to highlight that how well this approach works relies on how effectively the service implements its authorization system. This may involve validating permissions against user roles, integrating with an authentication system, or employing other authorization mechanisms.

In summary, the hypothesis suggests that building service resources based on possible HTTP requests and utilizing them for authorization based on permissions can be a practical and effective approach to building and securing RESTful services.

H3: HTTP methods and their arguments (parameters) can be used to establish a way for generating authorization permissions.

The hypothesis states that HTTP methods and their associated arguments or parameters can be utilized to establish a mechanism for generating authorization permissions.

To understand this hypothesis, let's break it down:

1. *HTTP methods:* HTTP provides many methods that specify the action type performed on a resource. HTTP methods that are usually used: GET (for fetching (retrieving) the resource), POST (for creating the resource), PUT (for updating the resource), and DELETE (for deleting the resource) [5].

2. *Authorization permissions:* They decide which tasks a user or client can do with a resource. These permissions are usually given out depending on specific conditions, like the user's role, their privileges, or other relevant factors. To illustrate, someone designated as an "admin" can usually perform any action they want with a resource, whereas an ordinary user might have fewer options available to it [14].

The service can determine whether the user or client has the necessary permissions to do the operation they are requested by looking at the specific HTTP method being used and the information given. The service can impose fine-grained access control and make sure that only authorized users can carry out specified activities on resources by mapping particular HTTP methods and their corresponding arguments to authorization permissions. It's worth mentioning that the way this authorization system is put into practice can differ based on the particular needs and technologies being used. The service may need to integrate with an authentication system, maintain the user roles and permissions, or utilize other mechanisms to validate and enforce the authorization permissions [4]. In summary, the hypothesis suggests that by leveraging HTTP methods and their arguments, services can establish a mechanism for generating authorization permissions and controlling access to resources based on the type of action requested by the client (service).

H4: By generating specific authorizations and managing them, one method for building services or service resources may be addressed. In this context, we can incorporate options based on data filtering that reflect on service authorization rights and permissions.

In order to grasp this hypothesis, let's divide it into three separate parts:

a) *Generating specific authorizations:* Authorizations refer to the permissions or access rights granted to users or clients to perform certain actions on resources. The hypothesis suggests that by generating specific authorizations, a method for building services or service resources can be addressed. This means that the service can define and manage fine-grained authorizations that align with the specific requirements of the system and its users.
b) *Managing authorizations:* Managing authorizations involves controlling and enforcing access rights within a service. This includes defining the available permissions, associating them with specific roles or users, and verifying authorization credentials or tokens during the service

interaction. By effectively managing authorizations, the service can ensure that only authorized users can perform specific actions.

c) *Incorporating options based on data filtering:* Data filtering refers to the process of selectively retrieving or manipulating data based on specific criteria or conditions. The hypothesis suggests that incorporating options based on data filtering can have an impact on service authorization rights and permissions. By applying filters to the data, the service can control the visibility and accessibility of resources based on the user's authorization level or other factors.

Let's take a service that maintains a client database as an example. The service may provide permissions like "view customer details," "edit customer information," or "delete customer records." By generating specific authorizations and managing them, the service can grant or restrict access to these actions based on user roles or specific permissions. Now, incorporating options based on data filtering can further enhance the authorization mechanism. For instance, the service might permit users with the "manager" position to access and edit all client data, while users with the "salesperson" role can only read and edit records related to the customers they have been allocated. Users can only access and modify the appropriate portion of data thanks to this data filtering based on user-specific criteria, which has an impact on the service authorization rights and permissions.

By combining specific authorizations and incorporating data filtering options, services can provide controlled and personalized access to resources based on the user's role, privileges, or other relevant factors. The service may utilize specific frameworks to apply data filtering and enforce authorization rights [13]. The hypothesis suggests that by generating specific authorizations, managing them effectively, and incorporating data filtering options, services can address the construction of service resources and enhance the service authorization mechanism to provide controlled access to resources based on user-specific criteria.

H5: Several authorizations may be enforced (attributed) to the service or service resource. Since the operations indicated by the HTTP methods can be mirrored by the separated authorization sets, authorization should always relate to the same relevant URI.

To gain a comprehensive understanding of this hypothesis, let's analyze it by separating it into three distinct parts:

a) *Enforcing authorizations:* Authorizations refer to the permissions or access rights granted to users or clients to perform specific actions on resources [12]. The hypothesis suggests that a service or service resource can have authorizations enforced or attributed to it. This means that different actions or operations on the resource can be associated with distinct authorizations.
b) *Reflecting HTTP methods with separate authorization sets:* HTTP methods indicate the type of action to be performed on a resource [8]. The hypothesis suggests that these HTTP methods can be mirrored by separate sets of authorizations. To put it simply, every HTTP method can come with its own permissions, which specify who has the ability to carry out a particular action on the resource. For instance, let's take the example of a service resource representing "customer records." Within the service, various permissions can be established, like "access customer information," "modify customer details," or "remove customer records." Each of these permissions aligns with a particular HTTP action: GET for seeing (retrieving), PUT for making changes, and DELETE for removing. By enforcing separate sets of authorizations, the service can control access to different actions on the resource and ensure that only authorized users or clients (services) can perform those actions.
c) *Relating authorization to the relevant URI:* The hypothesis suggests that authorization should always be related to the same relevant URI. Since the operations indicated by the HTTP methods are performed on a specific URI, the hypothesis emphasizes that the corresponding authorizations should be associated with the same relevant URI. Continuing with the previous example, the authorizations for viewing, editing, and deleting customer records should all be linked to the URI that represents the customer resource. This ensures that the appropriate authorization is checked and enforced consistently whenever a client attempts to perform an action on that URI. By associating the relevant URI with the corresponding authorizations, the service can maintain consistency in authorization checks and ensure that the correct permissions are verified for each operation on the resource.

H6: The authorization system can be comprised of a modularized structure (layer) that manages, handles, and controls authorization requests. This is accomplished by switching from the conventional way of service and resource managing to the authorization management system of services and service resources.

To gain a comprehensive understanding of this hypothesis, let's analyze it by splitting it into three separate parts:

a) *Authorization system as a modularized structure:* The hypothesis proposes designing the authorization system as a modularized structure or layer. This means organizing the components and functionality of the authorization system into distinct modules that handle specific tasks related to authorization management. This modularization allows for better organization, flexibility, and reusability of authorization-related code and logic.

b) *Managing, handling, and controlling authorization requests:* The authorization system is responsible for managing, handling, and controlling authorization requests [8]. Receiving authorization requests from users (services), confirming the requests' legitimacy and authenticity, and deciding on access privileges or permissions based on established rules or policies are all part of this process. The system may also log or track authorization-related activities for auditing or monitoring purposes.

c) *Shifting to an authorization management coordinated system:* It suggests moving away from the conventional way of managing services and resources to an authorization management coordinated system. The hypothesis proposes centralizing and coordinating the authorization management across multiple services and service resources. This allows for consistent and unified authorization mechanisms and policies across the system.

By adopting this coordinated approach, the authorization system can provide a centralized control point for managing access rights, permissions, and security policies. It enables a more efficient and scalable way of handling authorization across various services and resources within an organization or system.

The benefits of this modularized and coordinated authorization system may include:

- Flexibility: The modular design enables easy customization and adaptation of the authorization system to suit specific business requirements or changes.
- Centralized management: By centralizing authorization management, it becomes easier to enforce consistent policies, track authorization activities, and respond to security requirements.
- Scalability: The authorization-coordinated system can handle authorization for a large number of services and resources, accommodating the growth and expansion of the system.

The system may involve components such as policy engines, access control lists (ACLs) [13], role-based access control (RBAC) [5], or other mechanisms to facilitate the management and enforcement of authorization rules.

1.4. Methodology

Based on the hypotheses and goals of the research study, we will analysis of existing ways for building RESTful services and resources, will try to build new services and resources for authorization purposes, and we will propose the authorization management frameworks for managing authorization services (resources) based on RESTful service principles. REST [2], [5], [15] is the architecture for building web services, simple, heterogeneous, and web-based. Hereby includes how to manage authorization and permission of the resource state that is addressed and transferred via HTTP to a wide range of services [16]. Also, the REST design constraints architecture in order to simplify the use, development, and deployment of web data. The design of REST requires that architecture to be client-server in order to separate the user interface from data storage. This approach is in the interest of applications (services) in terms of interoperability, which is greater than formal contracts between peers. This is the biggest benefit because it provides that these components proceed regardless of each other.

By analyzing some ways for building service resources and authorization management [14], [12] we can define a precise language for building service operations. We can also define a manner for creating of authorization permissions based on operations [17] and their parameters.
Because the study has to do with building RESTful services and resources for authorization purposes and authorization management through an authorization service between services; the implementation of services based on proposed frameworks will be performed in order to derive results and verify the presented hypotheses. In this case, we will implement scenarios that cover all possible situations that tell management of services via authorization based on RESTful service operations and principles.

Defining authorization is included as a part of this research study. For using services and resources from one service or services set should be respected permissions of other services in order to have access to relevant services. In this case, permissions of services are defined as a right or ability to do something that is given to someone who has a decision or if it will be allowed to use or access

resources having specific permission. Allocation of common data via an interface (such as HTTP) should be protected in one way. The authorization option is considered a solution. The coordination model through authorization [18] is considered an effective model for service management. However, if a centralized mechanism for authorization gives benefits with optional data depending on applied permission, then it might be an approach that provides the permission rules not only for mapping of resources but also for defining of substructures of this entity. Each rule in the authorization mechanism will make possible the support of resource aware-authorization depending on the HTTP verb demand or entity characteristics. The authorization based on in-depth data and service characteristics, as well as in requests and responses of resources act as a finer level of direct allocation resources through URI.

The appropriate verifications will become the evaluation of the system services. Hereby will be considered authorization that one service will require to use the resources of other services by sending a request for the relevant service (by analyzing the URI of primary services, whether the URI matches to defining authorization of other services, then the request will perform with success, otherwise the service will not access the resources or other services).

1.5. Thesis structure

The structure of this thesis consists of six chapters that are succeeded by a section containing references.

Chapter 2: Fundamentals and literature review - includes information related to the content of the study. Initially, SOA, authentication and authorization mechanisms for SOA, RESTful services - characteristics, principles, concepts, service resources, resource identification, service and resource presentation, HTTP methods and codes, etc. are introduced. Then the security characteristics of RESTful services, authentication, and authorization mechanisms for REST are presented. Finally, it is discussed about the service interface description.

Chapter 3: Authentication and authorization in service-oriented grid architecture - this chapter provides key information on security issues of services in the grid architecture. First, the challenges in grid security are presented, including elements of authentication and authorization, then the models of these domains in the grid. The chapter continues with the presentation of authentication and

authorization infrastructures and technologies in grid services. It discusses the challenges, limitations, and benefits related to building secure services, specifically authentication and authorization in grid services. Finally, concluding remarks are presented to summarize the security issues in the aspect of authentication and authorization of services.

Chapter 4: Authentication and authorization in service-oriented cloud computing architecture - first, it presents issues related to the security of services in cloud computing, then the mechanisms of authentication and authorization of services in the cloud. The chapter continues with the presentation of contextual auditing mechanisms in the security of cloud services, specifically with elements of authorization. Since privacy plays an important role in service authorization policies, this aspect is also elaborated through the GDPR as a subsection of chapter 4. This chapter has also elaborated on the challenges, limitations, and benefits based on security characteristics, specifically authentication and authorization in services built and integrated into the cloud. Finally, there are presented the concluding remarks to round out the issue of security in the context of authentication and authorization.

Chapter 5: The authorization management framework for distributed RESTful services - initially, it presents important specifics about authorization issues, which are related to the contribution of the problems of the study, aspects for the elaboration of the management and functionality of the authorization in the case of interaction, access and exchange of resources between REST services. It introduces parameters that affect the building and limitation of the system during the interaction with resources and services based on authorization elements. Further, the proposed framework is presented, which enables the management of services' access to the resources of other services in the case when they have and do not have authorization. The chapter is concluded with the presentation of experimental results derived from the implementation of the proposed framework and discusses the challenges, benefits, and limitations encountered during the use and implementation of the respective framework.

Chapter 6: Service authorization management based on the proposed framework for RESTful services - presents the proposed framework in another format, with specifics about how it performs when services built into the framework attempt to access services or service resources. Aspects of service management in the context of authorizing access to services for use of other services' resources

and providing services with security elements, in this case through JSON Web Token, are presented in this chapter. Integration, interoperability, security, scalability, and performance of services are discussed, analyzed, and presented in this chapter about the services of the proposed framework. Subsections of this chapter present the challenges, limitations, and potential benefits encountered when building REST services for authorization purposes. The chapter continues with the comparison of the proposed framework with other existing frameworks and approaches. The chapter is concluded with the presentation of experimental results based on the implementation of services and the proposed framework in a real scenario. Finally, the REST services were evaluated based on the testing of the services implemented according to the proposed framework, compared to other existing frameworks and approaches, and the performance of the framework with REST services on local and cloud servers.

Chapter 7: Conclusion and future work - provides a concise overview of the accomplishments of this study and outlines potential avenues for future research.

2. FUNDAMENTALS AND LITERATURE REVIEW

The aim of this section is to give information about SOA, and RESTful services - characteristics, principles, concepts, service resources, and resource identification, authentication, and authorization mechanisms for REST. The section also includes details about service authorization and service interface.

2.1. Service-oriented architecture

The interaction between software units that provide services is made possible by the architectural style known as service-oriented architecture (SOA) [19]. It presents a method for developing, designing, managing, and deploying services that provide business reuse possibilities through the clear-cut interface and enable well-defined separation amongst service interface and service implementation. The SOA architecture supports service discovery, composition, and call. It relies on protocol and the message is based on document interchange [7].

SOA enables client system services and remote client functionalities. SOA is an architectural paradigm that aims to increase the agility and cost-effectiveness of a company by minimizing IT difficulties (burdens) throughout the company. It decreases the difficulties of information technologies by considering services as a logical option in the company. SOA can be utilized to set secure domains and facilitate systems to face fewer security risks. Additionally, providers of software as a service (SaaS) through SOA might present distributed systems to the Internet, necessitating more sophisticated security. Services are provided as a single component, but they can be also provided and implemented like web services or REST services [20].

Web Services Definition Language (WSDL) is used to define and describe web services in XML format, particularly their functionalities and data [6]. The word "web services" encompasses both web services and RESTful services. They are two various types of services that are distinguished from each other. Web services are equivalent to WSDL and WSDL techniques, whereas RESTful services are built using the REST architectural style. REST is similar to SOA because it is built on patterns, architectural style, and practices [7], [20].

Web services include the interoperability of multiple systems and resources. Since web services operate in a dynamic environment, the combination of many technologies to provide security for end-to-end services is needed. SOA is a broad and complex topic that encompasses a wide range of developments. There are now a few initiatives in place to ensure security services (authentication) between participating parties, as well as confidentiality, communication integrity, and authorization [7], [21].

SOA, as described, has a loosely coupled feature, making it vulnerable to security threats [22]. It implies that SOA must meet a number of criteria [19]: discovery of services, authentication of services, availability, user authentication, and confidentiality, control of access, integrity, and privacy. The open standards communities that produced web services generated a variety of security standards that are valuable for web services, and they are frequently used in SOA implementations, to assure security in a loosely connected SOA context [22].

The WS-security specification describes improvements in messages sent over SOAP to ensure integrity, confidentiality, and authentication of them. The authorization of web service does not currently have a standard framework. Various research groups are attempting to create authorization frameworks and principles for web services. Following the authentication procedure, most web services-based applications use application-specific access control mechanisms to make authorization decisions [23]. This contributes to a tendency to frequently try to reinvent the right elements, encouraging us to look more closely at the SOA authorization demands.

It is absolutely essential to provide the security of the systems that operate on the web, the messages that are exchanged, and the communication channels should also be ensured, to enable cooperation within the service-based organization. Confidentiality, authentication, authorization, integrity, and availability are all basic security elements that must be met and guaranteed because they are also essential for the security of web services and are considered active subjects whose application is yet researched in academia and industry [24].

In the study [25], several difficulties related to the security of web services were found. Web service threats include denial of service attacks, malicious code injection, and session hijacking [26]. Sultan, A. et al. [27] were interested in finding answers to challenges like WS-address spoofing and

SOAP operation spoofing in the context of dynamic web service building. But, solutions of this nature should be shifted to APIs for the purpose of expanding the security features of XML.

Kołaczek, G. et al. have described an architecture model with many SOA-related applications, and they proposed such a security model, which enables security functionalities in the case of building integrated systems [28]. Tampering, scanning, and the same attacks and threats affect WSDL files that determine the core context of a web service. Chakroborti, D. et al. [29] have demonstrated a model that enables the encryption and securing of WSDL files. Beer, M. et al. [30] have proposed and analyzed a secure mechanism for the RESTful web services, respectively for RESTful WCF services.

Threats and serious attacks affecting the security of SOA and their indications in SOAP web services were thoroughly analyzed by the relevant authors [31], who have derived conclusions and recommendations based on the literature reviewed and their experience with the respective issue. The study [32] shows a novel technique known as ATLIST to identify possible vulnerabilities in management practices in order to implement and adapt the SOA against them. In [21], it was used an expanded knowledge base to perform an evaluation of SOA security.

2.1.1. Authentication and authorization mechanisms for SOA

The user's identity is confirmed through authentication, while the authorization determines the transaction's legitimacy, assuring that all parties included have permission to take part. Access control should be ensured in the best possible way to avoid unauthorized parties, which try at all costs to gain unauthorized access to resources, services, and their functions [33].

2.1.1.1. SSO in SOA

Single sign-on (SSO) is an authentication service that allows a user to sign in to various apps (services) using a single set of credentials. SSO allows organizations and individuals to manage multiple accounts and passwords more easily. In a basic online SSO service (as in Figure 2.1), an application server component (agent) obtains an individual user's specific authentication credentials from a specialized SSO policy server, while attempting to authenticate a user against a certain user repository, essentially, a Lightweight Directory Access Protocol (LDAP) directory [34]. In the subsections that follow, some of the SSO configurations will be discussed.

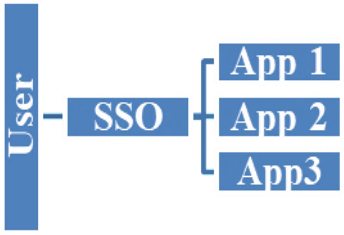

Figure 2.1. Basic principle of the SSO

2.1.1.1.1. Open ID in SOA

The OpenID standard creates a framework for information sharing between identity providers and OpenID acceptors. The OpenID Attribute Exchange is a standard feature that facilitates sending user attributes such as name and birthdate from the OpenID provider to the dependent party easily (a specific relying party can request various collections of attributes). Furthermore, services as well as the OpenID standard cannot impose a single mechanism for authenticating users, allowing for a range of authentication methods, including passwords and biometrics [35].

OpenID is based on SSO services' features, while the authentication is performed using the credentials only once to authenticate and access multiple web services. It is a decentralized authentication system. OpenID Connect is the latest version of the OpenID mechanism. It is built as the identity layer based on the OAuth protocol. In communications between the web interaction parties, OpenID Connect enables their encryption and sign-in [36].

2.1.1.1.2. SAML in SOA

The security assertion markup language (SAML) is an XML-based system for communicating security-related information among service consumers, identity providers, and service providers (SP) [37]. In the form of a SAML token, security information is represented in terms of assertions about a subject or a user. The SAML token does not contain any user credentials. It allows data communication between interaction parties to be encrypted. The user initially requests a web service from the SP, who thereafter requests and receives authentication assertions from the identity provider [38].

The SAML protocol is designed to address the following security concerns:

- Interoperability - the ability to communicate between security systems at different levels.
- User privacy refers to the ability to protect a user's identity and personal data (for example, credit card number).
- Federation - the ability for enterprises to securely share data.
- SSO enables users to log in to various applications.

SAML provides the ability to define a standard format for security information, as well as a standard method for transmitting secure data between apps. The principal receives local authentication services from the identity provider (user). The identity provider transmits the SAML assertion to the service provider when the principal sends the request. Following that, the service provider decides on access restrictions and privileges/permissions based on the received assertion [37].

2.1.1.1.3. XACML in SOA

Extensible access control markup language (XACML) [39] standard defines a descriptive access control policy language based on XML. According to policy rules, XACML explains how to analyze the permission request. The separation between authorization decision and point of usage is supported by the XACML concept. The authorization policy can be updated on the fly, and it impacts all customers instantaneously [20], [40].

XACML defines rules for granting access to resources based on the requester's attributes, the request protocol's features, and the authentication context. XACML specifies the constraints under which rules can be created, how rules can be coupled, and how rules are processed to make choices. The target is a determination request that leads to resources that are relevant to the issue (users or computers) [37], [40].

2.2. RESTful services

Roy Fielding came up with the expression REST to define an architectural (compositional) style, consisting of a set of principles for developing software structures based on network data [2]. REST is the software architectural style that is used to design loosely coupled distributed services. This architectural style is used for building modern web software. It is considered a way of creating

distributed services known as REST/RESTful services. RESTful is a principles collection that enables the building of web services heterogeneously, simply, and as a format based on the web. Its robust and adaptive techniques in support of communication in distributed environments doing REST are much more fulfilled [14], [12].

HTTP is the base of REST and enables interaction between services in different environments. The data services transmit through the web, while the HTTP like a communication protocol provides access and operations with stateless. Resources are distinguished through the unique URLs. HTTP provides addressing and transferring of the resource's state for different services [16]. RESTful design restricts the web structure for simplifying the development, use, and deployment of web services [12], [41].

Four principles of REST were presented:

- Resources must be identified;
- Using representations to be manipulated resources;
- Messages to be self-descriptive and
- Hypermedia as an application state engine [12].

These concepts come together to form a succinct and coherent metaphor for the web's processes and interactions. Resources are the building blocks of the internet. A resource presents something that may be identified as a hypertext target (e.g., a collection of resources, a script, etc.). The client receives a representation of the resource in response to a resource request, which may differ from the resource owned by the server. Messages with standard meanings are used to manipulate resources; we refer to them as HTTP methods [2].

The RESTful design uses the client-server paradigm architecture and enables the division of interface from data storage. This makes possible the components' development independently without affecting their stability. The constrained state enables a service to maintain the application state [12]. SOAP as a protocol is reserved for RESTful architecture and directs via HTTP protocol, the exchanged messages using envelopes without encapsulations are transmitted [42]. For such issues, interoperability is more important than the conventional relation between peers and this is in the interest of applications [14], [12].

A web service to be considered RESTful, it should respect the following constraints: uniform interface, stateless, cacheable, layered system, and client-server. Services that are built as above mentioned constraints are considered RESTful [43]. The limitations do not dictate the technology in which services can be developed. If they are respected the above-mentioned best practices, services can be scalable, reliable, highly efficient performance and portable [5].

RESTful acts in five major operations (as in Table 2.1):

- POST - uses to create a resource on the server. For example, POST/orders - creates an order, and POST/orders/235 - creates an order with id 235.
- GET - uses to return (retrieve) a resource or a collection of resources from the server. For example, GET/orders - returns a list (collection) of orders, and GET/orders/234 - return an order with id 234.
- PUT - updates (replaces) a current resource in the server. For example, PUT/orders - updates an order, and PUT/orders/236 - updates a current order with id 236.
- DELETE - deletes a resource from the server. For example, DELETE/orders - delete an order, and DELETE/orders/236 - delete an order with id 236.
- PATCH - updates partially a current resource from the server. For example, PATCH/orders - updates partially an existing order, and PATCH/orders/13 - updates partially an existing order with id 13 [14], [12].

Table 2.1. RESTful architecture basic methods

HTTP verb	**CRUD operations**
POST	Create
GET	Read (retrieve)
PUT	Update
PATCH	Update partially
DELETE	Delete

Services and servers in the REST structure exchange data through the interface and standardized protocol. REST supports HTTP, while the authorization is applied at the protocol level

using the HTTP interface and operations (GET, POST, PUT, PATCH, DELETE, etc.). REST operates with various resources presented in XML, JSON, or any other relevant format [12].

2.2.1. RESTful service resources

Resources are the primary focus of the REST architecture. A resource is anything that can be accessed or manipulated. Examples of resources can include "videos", "user profiles", "images", "tools", and "personas". They are capable of being linked with other resources. As an instance, within an e-commerce platform, a client has the ability to place orders for multiple products. The resources for products are associated with resources that are linked to the order in this particular scenario. Resources can also be organized into collections. In the context of an e-commerce service, the "orders" category represents a collection of individual "order" resource [13].

They are considered basic elements of web service. If we work with REST, the primary duty is identifying resources and their relations. Each resource has a unique identifier in the web platform known as the Uniform Resource Identifier (URI), and the best example of this is the Uniform Resource Locator (URL). A resource can be referred through several URIs, so it has not a limited number of URIs. For example, we can access a specific domain (resource) using http://example.com and/or http://www.example.com. Nouns are used to identify resources of RESTful services [12]. Information for the product from the database ProdDB can be accessed using the URL: http://product.service/products/1.

We should not use verbs for resource naming, because using them ensured a large list of URLs without patterns, which causes difficulties for their maintenance. To eliminate this problem, we should use nouns to name the resources. Another dilemma that appears when naming resources is to use singular or plural names [8]?! We always need to use nouns in the plural naming of the resources in order to have more consistency. For example, a specific product we can access in three ways that are presented as follows:

GET products/1
GET products/134
GET products?id=3.

In order to reduce the round trip number in the service (client), we should use composite resources. Such resources can be built by combining data (information) from other resources. For example, when we want to show our personal Yahoo page, initial aggregate news, weather, blogs, tips, etc., and then show them as a composite resource.

In cases when we design the URIs, they should be meaningful and well-structured. During the designing of URIs, we use path variables to divide elements into the hierarchy. Applying filters, sorting, and selecting specific resources is made through the parameters that are used to design URIs [8], [12]. In the following are given some examples that related to the structuring of URIs for getting cameras from the e-commerce site.

Table 2.2. Several examples related to the structuring (design) of URIs for getting cameras from an e-commerce site

Return all seven rated cameras	http://www.e-commerce.com/Cameras?review=7
Return all cameras from Samsung brand	http://www.e-commerce.com/Cameras?brand=Samsung
Return cameras that are released in the year 2018 (order by ascending)	http://www.e-commerce.com /Cameras?year=2018&sortbyASC=release date
Return cameras that have 20X zoom	http://www.e-commerce.com/Cameras?zoom=20X

2.2.1.1. URI templates

When working with REST services and APIs, it is important to accurately represent the URI structure. As an illustration, let us consider a blog (personal website) application with the URI http://personal.blog.example.com/2023/posts, which allows us to access all personal (blog) posts created in 2023. To work with this template, it is essential to understand the URI structure of http://personal.blog.example.com/year/posts, which indicates the scope of the URIs. The Internet Engineering Task Force (IETF) has defined URI templates in RC65703, providing a standardized method for depicting the structure of URIs. In the case of the example mentioned above, a possible URI standard could be http://personal.blog.example.com/{year}/posts [13].

In this instance, the curly braces indicate that the "year" field is a variable in the template fragment. Services can use as the input variable based on URI templates and replace the variable "year" with a specific value to retrieve personal (blog) posts for a particular year. On the server side, URI templates facilitate the process of parsing and retrieving the values of selected URI variables in an efficient manner [6].

2.2.2. RESTful service and resource presentation

The REST service can provide resources in various presentation formats, including HTML, XML, and JSON. The selection of formats for the REST service is dependent on the target audience. If a RESTful service is intended for internal use within an organization, it may only need to support JSON format, but a public-facing REST API may need to support both XML and JSON formats. JSON format is utilized in our scenario to present resources and facilitate operations within the REST service [6].

The important part of RESTful architecture is the way of showing resources to the clients (services). REST supports different formats for the presentation of resources, so we do not have any limitations during the presentation of resources in specific formats. For the presentation of resources, favorite formats are considered JSON and XML, but enable the presentation of resources also in other formats [12].

The format definition and media type in RESTful services depend on service needs and requirements. Therefore, the format that best describes the requirements and needs of services is considered adequate for use. Not just a format can be right for all service requirements. In case of the impossibility of defining the requirements and making the decision to use one of the specified formats, then JSON (application/json) or XML (application/xml) formats are widely considered. To choose the right media type we need to focus on details of errors and information that exist [8].

2.2.3. Communication protocol

REST services commonly rely on HTTP for establishing communication. The HTTP protocol serves as the foundational structure for the application layer of the web, acting as the most widely employed protocol for facilitating communication among various web components. Its design is

specifically tailored for the transfer of resource representations. As a result, HTTP and REST complement each other seamlessly. REST relies heavily on four primary HTTP methods: POST, PUT, DELETE, and GET. The initial specification of these methods can be found in the HTTP RFC 2616 document. The meaning of these methods was later revised in RFC 7231 [44]. A demonstration of invoking an HTTP method on a RESTful API (service) is provided in the listing 2.1.

```
GET /api/v3/pet/1 HTTP/1.1
Accept: application/json
User-Agent: PostmanRuntime/7.26.1
Postman-Token: ee6fe725-5def-4aa9-a9ac-9ed757b972e9
Host: petstore3.swagger.io
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Listing 2.1. Illustration of a GET request example [45]

In this instance, we initiate a GET HTTP method request to the specified endpoint, including the query parameter id set to 1. The initial line pertains to the type of method and the protocol version being used. The headers provide information about the different facets of the request. The Accept header is utilized to indicate the desired content type requested by the client. Subsequently, the service utilizes content negotiation to determine the suitable content type and communicates this information to the client through a response header. In this particular example, the format for the content is JSON. Content negotiation plays a crucial role in the context of RESTful APIs (services). It enables clients to select specific representations from the API (service) based on their individual requirements [45].

The Host indicates the fundamental URL for the given resource. There are numerous additional headers that can be incorporated into a request. The Connection header provides various control alternatives for the existing connection. In the given example, the term "keep-alive" signifies the reuse of the existing connection rather than establishing a new one.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 161
Connection: keep-alive

{
 "id": 1,
 "category": {
   "id": 2,
   "name": "Cats"
 },
 "name": "Cat 1",
 "photoUrls": [
   "url1",
   "url2"
 ],
 "tags": [
   {
    "id": 1,
    "name": "tag1"
   },
   {
    "id": 2,
    "name": "tag2"
   }
   ],
    "status": "available"
 }
```

Listing 2.2. Illustration of a GET response example [45]

The initial line of the service response contains information about the protocol version and the status code. In this scenario, the response status code of 200 indicates that the request is executed successfully. The service notifies the client indicating it has opted for the JSON content type, which is exactly what we requested. The response body contains the representation of the pet resource with the id 1.

2.2.4. HTTP methods

The "uniform interface" principle limits the communication between the client and server to a set of standardized HTTP operations, ensuring consistency and simplicity in the system communication process. The HTTP standard provides a range of methods that empower clients to communicate, interact with, and manipulate resources effectively. GET, POST, PUT, PATCH, and DELETE are among the frequently used HTTP methods. Safety and idempotency are two essential characteristics closely associated with these methods. The term "method" is employed in the HTTP

specification to denote HTTP requests, and the term "HTTP" is commonly used interchangeably with requests [13].

2.2.4.1. Safety

An HTTP method is considered safety if it does not result in any modifications to the state of the service. GET is recognized as a safety method since it is used to retrieve information or resources from a service without causing any modifications. This method is designed to perform a read-only function, ensuring that it does not alter the state of the service. The safety methods are employed for retrieving or reading resources. By utilizing the GET method, which is considered a safe operation, we can retrieve and read values with each invoke, as long as there is a potential for the values to be returned without altering the state [6].

2.2.4.2. Idempotency

If an operation is deemed idempotent, it will yield the same server state regardless of how many times it is applied. GET, PUT, and DELETE are classified as idempotent methods, ensuring that clients can repeat their requests and obtain the same effects with each subsequent request. Subsequent requests, including successful ones, have the same effects on the resource as the initial request, ensuring consistent outcomes. Once an order is removed from the e-commerce service, if the same removal is repeated, it will yield the same results because the order no longer exists on the server. When an order is created using a POST request, it can be observed that a new order is successfully generated. Due to its non-idempotent nature, the POST method does not guarantee consistent results when repeated, as it may lead to unintended consequences [13].

2.2.5. HTTP status codes

These codes facilitate the communication of service outcomes to the requesting client. They are classified into various categories as below:

- *Informational codes* indicate that the server has received and acknowledged a request, but the processing is not yet complete. The response codes fall within the 100 (1xx) series.

- *Successful codes* indicate that the request is successfully processed and accepted. These codes belong to the 200 (2xx) series.
- *Redirection status codes* indicate that while the initial request has been processed, the client or service is required to make further requests in order to fulfill the complete request. This occurs when there is a need to redirect the request to various locations in order to obtain a resource. These codes are categorized within the 300 (3xx) series.
- *Client error status codes* indicate that an error or problem has occurred in the client or service request. They are classified within the 400 (4xx) series.
- *Server error status codes* signify that the server encountered an issue or error while handling the client (service) request. These codes are classified under the 500 (5xx) series.

The aforementioned codes have a significant role in REST services as they serve as meaningful indicators that facilitate communication between services, ensuring appropriate actions can be taken based on the communication state [13].

Figure 2.2. HTTP status codes and their corresponding descriptions[1]

2.2.6. REST interfaces

The REST architecture is characterized by four specific properties, which can be outlined as follows.

[1] http://suso.suso.org/docs/infosheets/HTTP_status_codes.gif

2.2.6.1. Addressing

In the context of a service, it is necessary to identify each piece of data as a distinct resource, and subsequently address that resource using a URI. If a collection of resources has been defined, a GET method can be used to retrieve a list of those resources (if they exist). In addition, if a particular resource exists, the GET method can be utilized to retrieve that resource. In accordance with the REST style, it is necessary to address all of the defined resources [8]. Within our scenario, we are taking into consideration the process of addressing, whereby the defined resource(s) is accessed via a path of one or more links, leading to particular services.

2.2.6.2. Connectivity

This feature necessitates that a resource possesses interconnections with other resources, typically in the form of links. The connections between resources serve to link distinct resources and establish relationships between them in our scenario. The URLs are utilized to represent these connections. If a service is deemed entirely suitable for the RESTful architectural style, the resource presentation should include a list of links that enable further access to resources within the REST service [34].

2.2.6.3. Uniform interface

The uniform interface is one of the key constraints of the REST architectural style, and it is based on the use of a standard set of HTTP methods for manipulating resources. The consistency of interfaces is essential for all interactions between the client, intermediary components, and server involved. This streamlines the overall architecture by allowing components to evolve independently as far as they support the established contract [14].

2.2.6.4. Stateless

It is necessary for each request originating from a service to include all the necessary data for processing, as the server is not obligated to hold all information. This implies that each request is processed separately and autonomously. This specific feature of REST services enables improved collaboration with the system's current devices and infrastructure. Regardless of the impact of the

operations (GET, POST, PUT, PATCH, and DELETE), session or state data should not be concealed [5].

2.2.7. Endpoint REST service and resource identification

The identification of REST resources involves the utilization of URI endpoints. In order to be considered well-designed, REST services should feature endpoints, which are easily comprehensible and user-friendly. In the REST service scenario are considered and incorporated the most effective practices and common conventions for endpoint design. One common convention for the service is to utilize a URI base. It serves as the starting point for accessing the REST service. The REST API provides the ability to use subdomains, like http://api.web_domain.com or http://dev.web_domain.com, as the base of URI for accessing the service. To prevent any confusion of service names, there are established distinct subdomains. This further enhances the ability to reinforce authorization policies [6].

The next convention is that resource endpoints should be named in their plural form. To illustrate, the URL http://www.example.com/products is utilized to access a set of product-related resources. One way to access resources for specific products is by using a URI, like http://www.example.com/products/4321. Using a URI template such as http://www.example.com/devices/{products_id}, we can create a generalized way to access product resources [13].

The last convention involves organizing and presenting resources in a hierarchical manner using URIs to establish connections between them. Within the context of the service, the product resource is linked to a corresponding e-commerce resource [12]. To achieve this objective, we can use a hierarchy of endpoints that follows the format of http://www.example.com/products/{product_id}/e-commerce. This endpoint would retrieve and enable to manipulate all e-commerce associated with a specific product. Similarly, the endpoint www.example.com/products/{product_id}/e-commerce/{e-commerce_id} enables to retrieve and manipulate a specific e-commerce associated with a specific product.

2.2.8. SOA and RESTful security issues

The ecosystem is dominated by two types of APIs: SOAP and REST web services [10]. While SOAP is still widely used in the workplace, it is giving way to the current REST web service paradigm. Each of them makes data available via HTTP requests and responses, but in quite distinct formats and semantics, necessitating separate security considerations [33].

A conceptual reference model for web service security requirements is presented in Figure 2.3. The multiple standards are mapped to various functional tiers of a web service implementation in this reference architecture [7], [9]. SOAP has evolved over time to include extensions to deal with the unique security concerns of transactional messaging. It has been in existence for a sufficient amount of time and has been embraced by notable corporations, earning the support of OASIS and W3C. The major goal of XML-Encryption, XML-Signature, and SAML tokens is to improve the security of data submitted to and received from a SOAP service [46].

Due to the pattern's focus on how to supply and consume data rather than how to incorporate safety into data interchange, REST does not implement any specific security patterns. Those that use REST design patterns should consider the right degrees of security when coding, deploying, and transmitting their data. In the following are some of the differences between verifying the security of a REST service and a typical SOAP web service:

- SOAP requires a request payload (XML envelope); let us try to take advantage of this feature by sending a large number of badly formatted data, or even a vast quantity of data in a single request.
- SOAP infers the presence of an XPath parser on the back-end; let us try injecting known attacks for typical XPath libraries using some poor XPath.
- To route and process endpoint requests, REST mainly relies on the HTTP method.
- JSON structured communications, which, like XML, maintain a precise document/object structure, are frequently used in association with REST.

Web services are susceptible to vulnerabilities at both the message and service levels. Some of the biggest service-level risks are listed below [11]:

- Identity spoofing and malicious code injection: these attacks are generally carried out via XML files. An attacker can insert malicious code into the service, causing it to malfunction.
- Session hijacking: an attacker can acquire unauthorized access to resources by stealing the user's session token. As a result, false requests or responses are transmitted, and the service requester's and the provider's sessions are said to have been hijacked.

In the following are threats to web services at the message level:

- Message injection or modification: malicious messages can be introduced into communications sent between the client and the server.
- Communication replay: an attacker takes a legal message and replays it after a period of time to get confidential information through access to services that are not authorized. This is usually the first stage in compromising an online service, where the hacker takes control of the session (transaction) and manipulates the services.
- Message confidentiality and eavesdropping: online services are always vulnerable to message interception. Traditional security techniques such as VPN and SSL are insufficient to protect web services from hackers [11], [47].

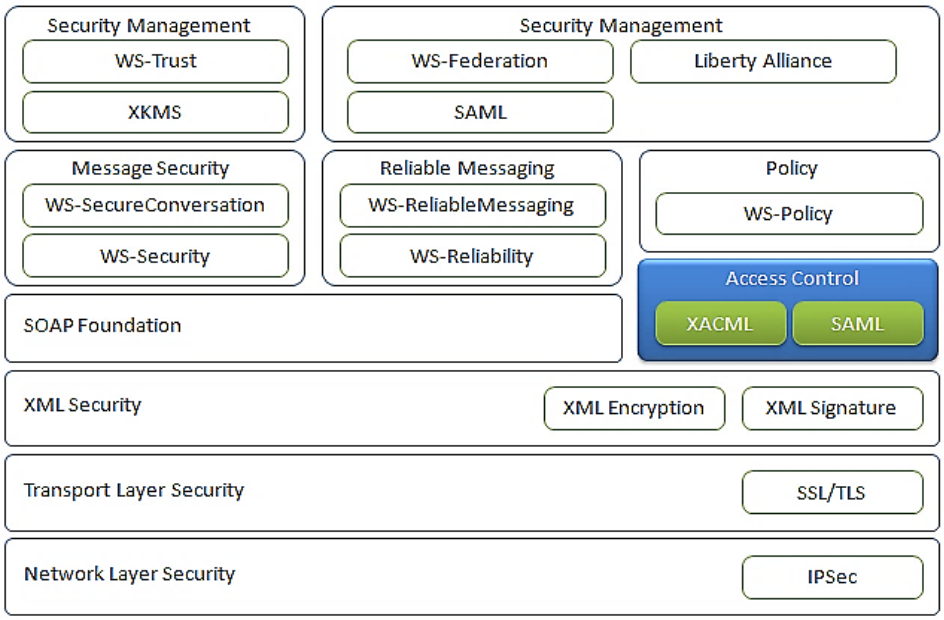

Figure 2.3. The security structure for web services [48]

2.2.9. RESTful authentication and authorization mechanisms

In RESTful services, authentication plays a tremendous role in terms of security, so they are constantly charged with authenticating users and other services. Security as an essential aspect of services affects that servers necessarily contain the authentication element, and they must authenticate each client and service for each of their requests. Many institutions have chosen some sophisticated authentication mechanisms, such as HTTP basic authentication; HTTP digest authentication, and token-based authentication [42].

HTTP basic authentication is a simple authentication scheme. The credentials are used in the HTTP header to verify the client's request for authentication. The authentication stages for HTTP basic authentication are depicted step by step in Figure 2.4. HTTP header considers as the authorization with base64 encryption. It sends the credentials as strings (a text format) and stores them in the HTTP authentication header. They are mostly unsafe. To overcome this vulnerability, data must send through the HTTPS protocol or SSL protocol that encrypts the HTTP channel that also credentials carried. HTTP basic authentication is faced with numerous security threats such as attacks affected by replay and injection, as well as middleware hijacking. Such attacks come because credentials usually stored in the HTTP header are indirectly exposed to the corresponding attacks.

HTTP digest authentication is considered more progressive than the previous authentication. It uses MD5 as the hash encryption mechanism to encrypt the user credentials. The hash can be protected by HTTP basic authentication against attacks. This is enabled by generating a nonce at the client endpoint. In addition, a server-generated timestamp can protect a client's message against the replay attack. HTTP digest authentication faces with some security flaws. It can be compromised using the Man-in-the-Middle technique because it doesn't have a way to crosscheck the server to the client. When it happens, HTTP digest authentication is changed to basic authentication [7].

Token-based authentication is a more sophisticated authentication and uses a token. In Figure 2.5 is depicted how the token-based authentication mechanism works. The user makes a request for service on the resource server (RS), while the browser of the user is redirected to the private key generator through RS. Initially, the user requests access to the authorization server (AS). Then, the user authentication is performed. It makes a request to receive a token from the AS. The RS then identifies

the token received from the AS. Finally, the RS uses the token to verify the user's messages. Since a token is used during the communication between the user and the RS, the third party makes sure that the credentials are protected. The usage of tokens during the authentication process has pushed Twitter, Facebook, Google, and Microsoft to implement the OAuth authentication method in their services [42].

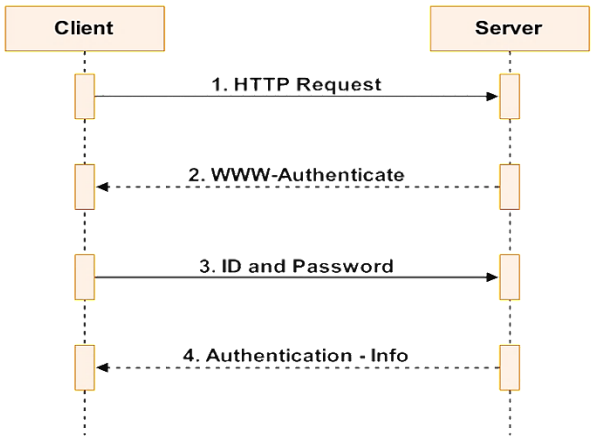

Figure 2.4. The authentication process based on HTTP basic authentication

REST is well-founded on the statelessness concept, the above-mentioned authentication methods require statelessness to be implemented through numerous requests. OAuth is being used as an authentication method in most organizations that provide web services [42]. Resources and roles are the main principles on which the authorization service is designed. Resources are considered the abstract feature and whole actions can be mapped in HTTP protocol operations [49]. The confidential information protection and security tasks are the benefits of centralized authorization mechanisms. Nevertheless, there are authorization mechanisms like ABAC and RBAC [43] that are discussed in the subsections below.

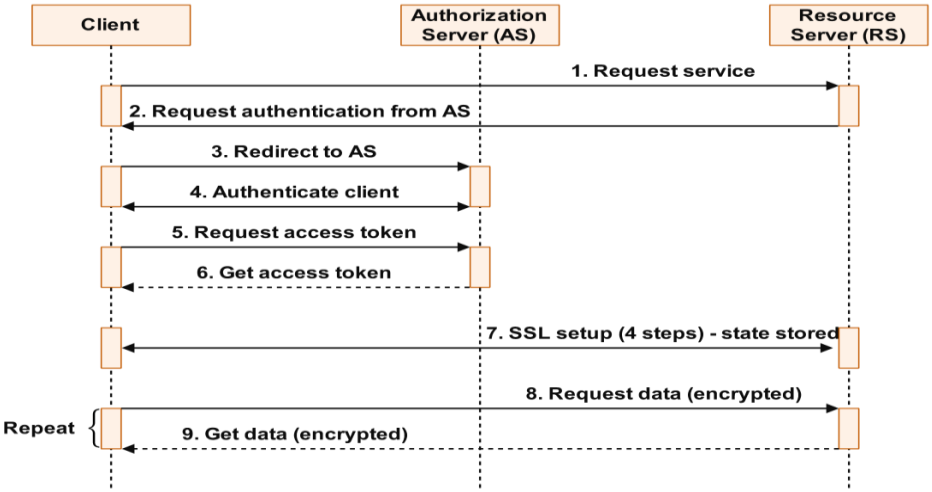

Figure 2.5. Token-based authentication scenario

2.2.9.1. Attribute-based access control in REST services

Attribute-based access control (ABAC) is considered a mechanism for controlling access and authorization permissions. It uses its policies to define various collections of attributes that are needed to control the user access privileges and rights separately, as indicated in Figure 2.6. The policies are mainly created through various types of attributes. The system is based on these policies and decides the access permissions. Here includes a series of attributes which are object and subject attributes as well as resource and environmental attributes. Under the ABAC mechanism, each user's role and permission is predetermined. This model enables the solution of many issues of authorization and enables flexibility in implementation [43], [38], [50].

The starting date of the user's work, his location, the user role, etc. can be the attributes. In addition, a subject is not necessary to be identified preliminary to the system. It must once be authenticated in the system and then ensure its attributes. Nevertheless, an agreement must be reached to know what type of attributes should be used and how many of them are taken into consideration to create access decisions [43], [51].

It is essential to be proposed a security policy that can work exactly with the ABAC model with web services, due the security policy is accountable for picking the substantial attributes that are used to create access decisions.

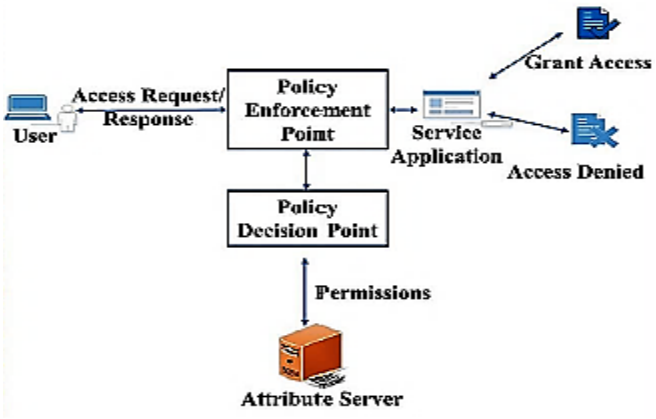

Figure 2.6. ABAC architecture and operations [52]

2.2.9.1.1. JSON web token

The JSON web token (JWT) presents a secure method that is used to define a collection of information amongst two interacting parties. A JWT is a three-part string including a header, a payload, and a signature [53]. It stands for the claim format that can mainly be used in restricted environments and is closely related to the HTTP protocol [54]; including the headers of authorization through HTTP and URI query features [55].

The JWT header includes data on how to compute the JWT signature. "alg" is placed in the header, and it is represented through any hash algorithm (RSA or HMAC SHA256) to produce (generate) the JWT signature [53]. The JSON header can be in the format below:

```
{
"typ":"JWT",
"alg":"HS256"
}
```

The payload of a JWT comprises the data stored within the JWT such as the user id, an email address, or something else: Payload = {"userId": "example@university.edu”}. To create the second fragment of the JWT, the payload must be encoded to Base64Url. For instance, the pseudocode of the payload that is encoded to Base64 can be as follows: $payload = base64_encode($jsonHeader) [55]. The header signature and payload signature are an integral part of the JWT signature, which are computed through the algorithm below:

data = header + "." + payload
hashedData = hash(data)
signature = base64urlEncode(hashedData) [56].

The JWT signature consists of headers and payloads encoded by the base64 algorithm and encrypted by the HMAC-256 encryption algorithm. For instance, in the following is given the pseudocode for creating a signature: $signature = HMAC ($header + $payload, $key). In the following is given as an example of the pseudocode of the combining of the header, payload, and signature to produce the JWT: $token = $header + ’.’ + $payload + ’.’ + $signature [55].

It is depicted the client authentication process on the server, performed through the JWT mechanism as shown in Figure 2.7:

- The client submits the request for authentication using his credentials or by logging in through his Facebook account, Google account, etc.
- Once the user has been authenticated with its credentials, and they are verified by the authentication server (authentication service provider), the authentication service generates a JWT from the records obtained by the user authorization. The generated JWT is given back to the client for use in the future as a request to the server’s protected resources.
- When a client makes a request to a protected resource server, the token is utilized.
- After the server gets the request, the information of the user authentication found in JWT is unpacked in order to define whether the request is valid. Depending on the validity or invalidity of the token, the server will react appropriately to send back a response.

JWT offers significant benefits, and it is considered the mechanism that enables scalability, especially in controlling user access in decentralized and wide-range distributed systems. JWT enables

the transfer of state to communication and the user cannot alter the data included within the token. The disadvantage of using authentication through the JWT is related to its use by the server, where it uses the same JWT for all requests coming from the user until it completes all requests towards the server, and is logged out of the system [53], [57].

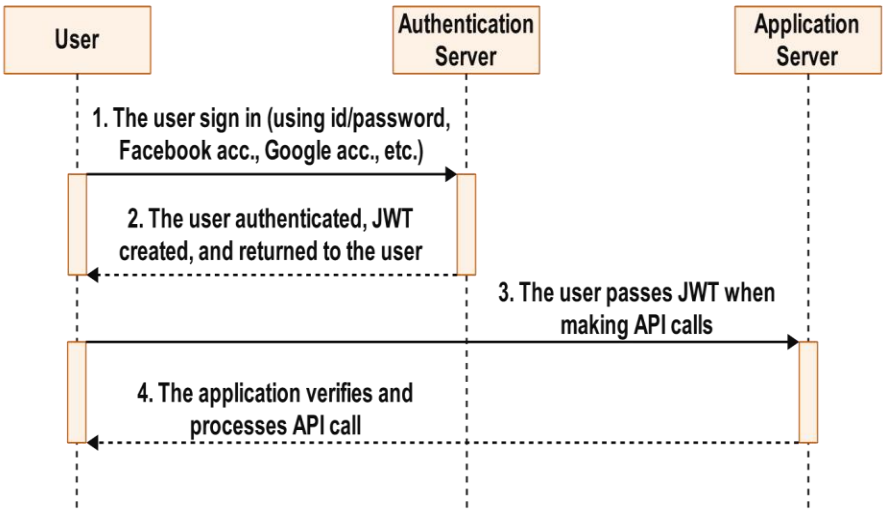

Figure 2.7. The step by step JWT's working scenario

2.2.9.2. Role-based access control in RESTful services

Role-based access control (RBAC) [58] is an access control mechanism that allows access rights for users according to roles and permissions (Figure 2.8). The policies of access control present rules that tell how the setup process authorization allows or denies users [59]. The user is defined as a human, process, machine, or network. A role presents the approval for performing an object operation, which can be an action, function, or duty. The object refers to information containers such as files, directories, database tables, or resources (printers, PC, etc.). The permission is defined as a 'tool' that enables the user to assign roles that are actually active along with user sessions [60].

In RBAC, user permissions are provided by various parameters, and they can be user roles, permissions of roles, and role-to-role relationships. Roles in RBAC are divided into two types: the application (technical) and organizational (business) roles. The first type of role involves the integration of various applications (services) particular rights or permissions-based tasks. The second type of role consists of various job functions and the rights of access allocated for employees. In organizations that have many users and required multiple permissions, RBAC is used to administrate their security [38], [50], [61].

RBAC primarily includes three principles for allocating permission to a specific user, like assignment of role, authorization of role, and authorization of permission. Based on such rules, users are given permission for accessing the resources. RBAC offers a confidential environment for setting permissions and privileges. The disadvantage of the RBAC is in terms of changing roles, which they change from time to time and there is always a need for real-time environment. Therefore, certain changes need to be checked and verified in that environment [38], [50], [61].

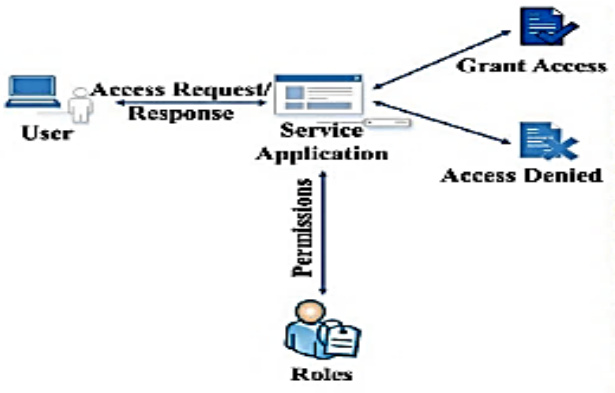

Figure 2.8. RBAC and operations [52]

2.2.9.2.1. OAuth in RESTful services

OAuth is an authentication mechanism used in web services, and it enables one-way or mutual authentication in the web environment. This authentication method allows the user to gain access to

another application (service) without the distribution of the password by the authenticated application (service). OAuth 2.0 is the latest version of OAuth [38].

The OAuth's goal is to complement OpenID and delegate access for users/services to the protected resources. This is enabled through the authorization server (AS) that generates tokens that do not contain information about user credentials [62]. OAuth offers a method of accessing the HTTP service from the application/service in the name of the resource owner that is used as an authentication standard by Microsoft, Google, and Facebook. The mechanism of OAuth is functioning on HTTP, which has a role in adapting the OAuth mechanism [63].

OAuth as an authorization framework enables third parties to access user/service resources without revealing the credentials. OAuth can execute the restrictive policies in the access domain and token expiration for restricting services' access to specific resources and functions [64]. OAuth is being utilized as a security layer and a standard protocol in web services [65]. In Figure 2.9 is presented the OAuth architecture including an independent authorization server. The interactive process between the parties in the OAuth architecture is depicted in the steps below:

- The client makes a request to the resource owner (RO) for authorization.
- The request received from the RO is transferred to the AS.
- By providing its credentials, the client requires authorization permission from the AS.
- The client's credentials and authorization permission are validated by the AS and if they are valid, then it returns a token for access to the client.
- The client is addressed to the resource server (RS) and requests the protected resource, while it is authenticated by providing the token for access.
- The token for access is initially validated by the RS, and if it is credible, the RS provides the required resource to the client [62].

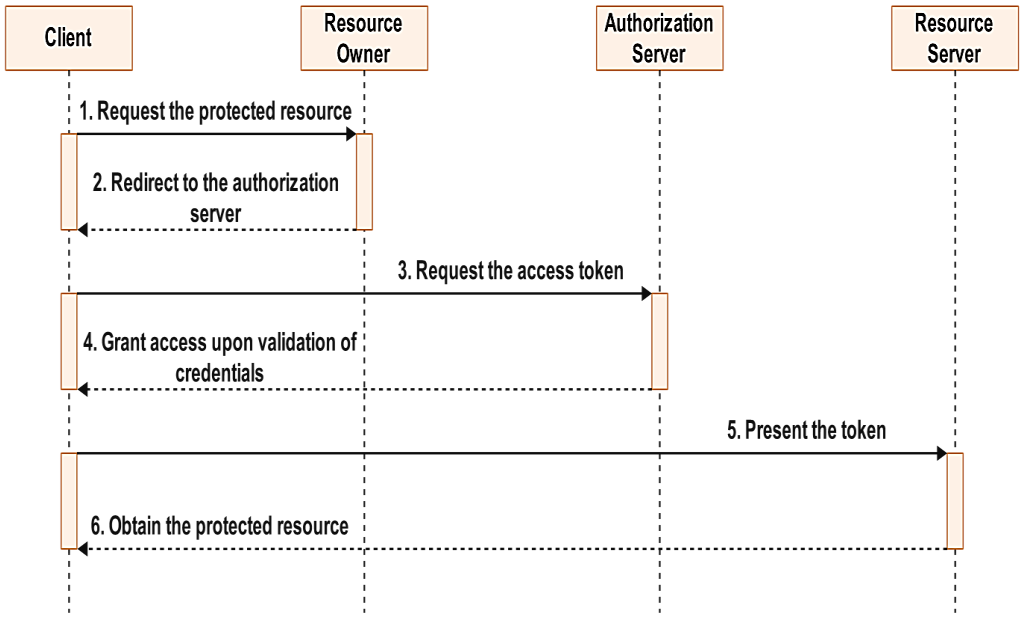

Figure 2.9. The OAuth architecture including an authorization server as an independent feature

2.3. Service authorization

Through supporting their users with reliable data both in terms of integrity and confidentiality, web services should contain secured, reliable, and protected data. Numerous technologies and methods are combined to provide secure web services. There are required many high-level protocols to overcome challenges like identity and trust. The identity represents the probability that the system authenticates participating pairs in transactions. On the other hand, trust indicates the authorization of parts that interact with the system in a defined manner [66].

The service provider determines which transactions are permitted once the client or service has been identified and this is referred to as authorization. What the client or service is allowed to do with the service resources that are exposed, can also be specified as authorization [20]. It mainly has to do with the main actor, in web services, who needs access permissions to use certain services. The principal (main actor) is an entity that has been granted authorization to use or access resources, whether it be as a person, system, or such entity [5].

HTTP is the protocol used by REST services, and it can be used to provide authentication via HTTP basic, HTTPS, or HTTP digest. These three authentication techniques are discussed in detail in one of the next sections. REST services can contain many components that may or may not be authorized for certain resources. For a request, the authorization header can be used to send the information authorization. The objective is to provide sufficient data in developed patterns that grant the appropriate users access. This necessitates that requests and responses become directly mapped to HTTP [7].

2.3.1. Open authorization

Open authorization (OAuth) is the open authorization standard that provides secured delegation access for the client application(s) to server resources [67]. By long-term securing their login credentials and identities, it gives access to the website users or client applications in the secure resources. OAuth is considered a robust protocol and it can be used at the endpoints of applications (services) or third-party websites. It is primarily made to operate with HTTP and considers token access that is granted by third-party users (through authorization servers), with the consent of the resource owner. The client (user) receives a token to access the restricted resources kept on the resource server [64].

OAuth is typically involved as the web surface secured method for web users to sign in to third-party services through their Google, Facebook, or Twitter accounts. OAuth was first introduced by Blain Cook in 2006. OAuth 1.0 was formally published in April 2010 as the standard approved by the Internet Engineering Task Force (IETF), and whole third-party Twitter services started using it in August 2010. The OAuth 2.0 was released in October 2012 [65].

2.3.1.1. OAuth 2.0

OAuth 2.0 is the successor of the protocol OAuth 1.0 and enables application access to the resources (data) [64]. It provides the particular web services' authorization and it focuses on the client (service) development simplicity [65]. OAuth 2.0 as a protocol is created to enable applications (services) to access data on other systems by taking into consideration authentication and authorization

as key elements for accessing specific resources. It focuses on providing authorization enforcement for web applications and services, desktop applications, mobile services, etc. [3].

In the following are described the steps that should be followed for a scenario of the OAuth included user authentication (Figure 2.10):

- The user inserts his credentials into the system. They pass through the Auth server (AS) toward the HTTP authentication header in the encoded format. Secure Sockets Layer (SSL) is used as a cryptographic protocol to enable the communications channel to be secured [68].
- The user is authenticated by the AS with its entered credentials. It then generates the token to restrict the time and send back a response to the AS.
- The API that resides in the resource server (RS) is invoked by the user. The RS passes a token to the HTTP server like a query-string.
- The RS after obtaining a token extracts it. It should be authorized in the authorization server.
- After the authorization is achieved successfully, the invoker receives a response [65].

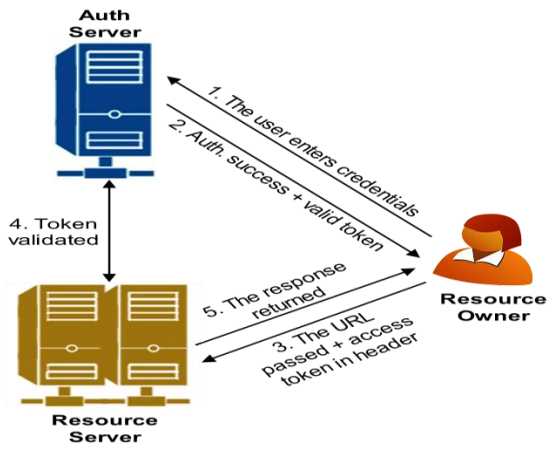

Figure 2.10. Authentication and authorization scenario in the OAuth

Through OAuth are separated the client roles from the resource owner (RO). The request of the client is accessed of resources checked by the RO, while data are hosted in the RS. The RO releases various numbers of credentials. The client despite having credentials for access to protected resources, also gets a token which is a string marked as the specific field, lifespan, and other attributes for

accessing. For third-party consumers, the authorization server issues a token for accessing that is approved by the RO. The client uses a token for accessing the hosted resources in the RS. There is included authentication and authorization. The RO is taken by delegating permissions. Several applications need to have authorization for accessing the services, especially for aggregated data. The responsibility for the management of tokens meets the services [65].

Details about four OAuth roles are given below. The RO is the entity that grants access to the end user for secured resources. The RS stands for the resource hosting server resources secured by tokens. The client is considered as the application made to manipulate the resources in the name of the RO - it is made possible by the protections authorized. The AS stands for the server that allows the customer to retrieve the token after effective authentication to the RO. OAuth 2.0 is a protocol for the application program interface (API) that supports authentication as well as authorization of interacting pairs. Tokens are correlated with a client's API invokes, allowing the service to verify the client's authorization [3].

OAuth enables the client application for sending the API query-string to the RS. The client authenticates the RS by including a token in the message of its API. The token was offered in advance to the client in the AS. There is an instance that the token is released by the AS - the involved part has granted its consent that the client to access certain attributes. The user/client communicates with the AS to ensure the token. It sends a claim to the AS to have access permission [65].

2.3.1.1.1. Authorization-based on OAuth 2.0

OAuth 2.0 consists of four roles: client, resource owner, resource server, and authorization server. It is presented as a framework that enables delegated access in an authorized manner. The resource owner represents an entity that has the authority for enabling access to a protected resource (for instance, an end-user) [69].

The server that hosts the resources secured by access tokens is known as a resource server. A client is introduced as an application that, with the permission of the resource owner, makes requests for the protected resource on its behalf. An authorization server is considered a server that releases

tokens for client access after successfully performing the resource owner authentication and securing authorization [70].

OAuth specifies how an API client can get tokens that represent the permissions collection of such API. Tokens are attached to the client invokes that address to the API by indicating its authorization in relation to the respective API. In the OAuth are using two types of tokens: the authorization token and the access token. To obtain an access token, the client communicates with an authorization server (AS) by delivering a request that contains an access grant. In certain situations, the client can additionally submit its credentials to the AS in response to a request message [69].

OAuth's flows are the interactions between the various roles involved in seeking authorization. Flows are sometimes considered to be the kind of authorization that has been provided. Authorization code, implicit, resource owner password credentials, and client credentials are the four flows available in OAuth 2.0 [65]. The flow is determined by the type of authorization grant and the kinds of grants supported by the API/service.

The flows are discussed and schematically depicted in the following this section:

- Authorization code flow is considered the most used flow, mainly with server-side services (applications).
- Implicit flow is utilized with mobile or web services (applications).
- Resource owner password credentials are the third type of flow that is mainly considered by trusted applications. The service can be the owner of a trusted application.
- Client credentials. The application APIs are used with this type of flow [69], [71].

A. Authorization code flow

It is considered the most used flow, and it is mainly involved in issues related to resource code on the server side. Here, client confidentiality can also be maintained. Because redirection is used in this flow, the application/service must communicate to the user agent. In this type, communication is performed between the client and the authorization server, during which is generated the access token [65] (as described step-by-step in Figure 2.11).

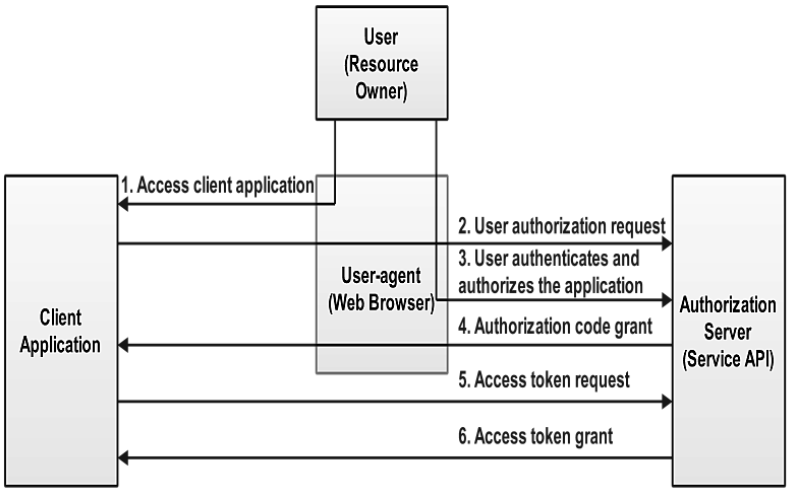

Figure 2.11. Authorization code flow scenario

In order to have more detailed information about the authorization code grant, a detailed scenario is shown in Figure 2.12.

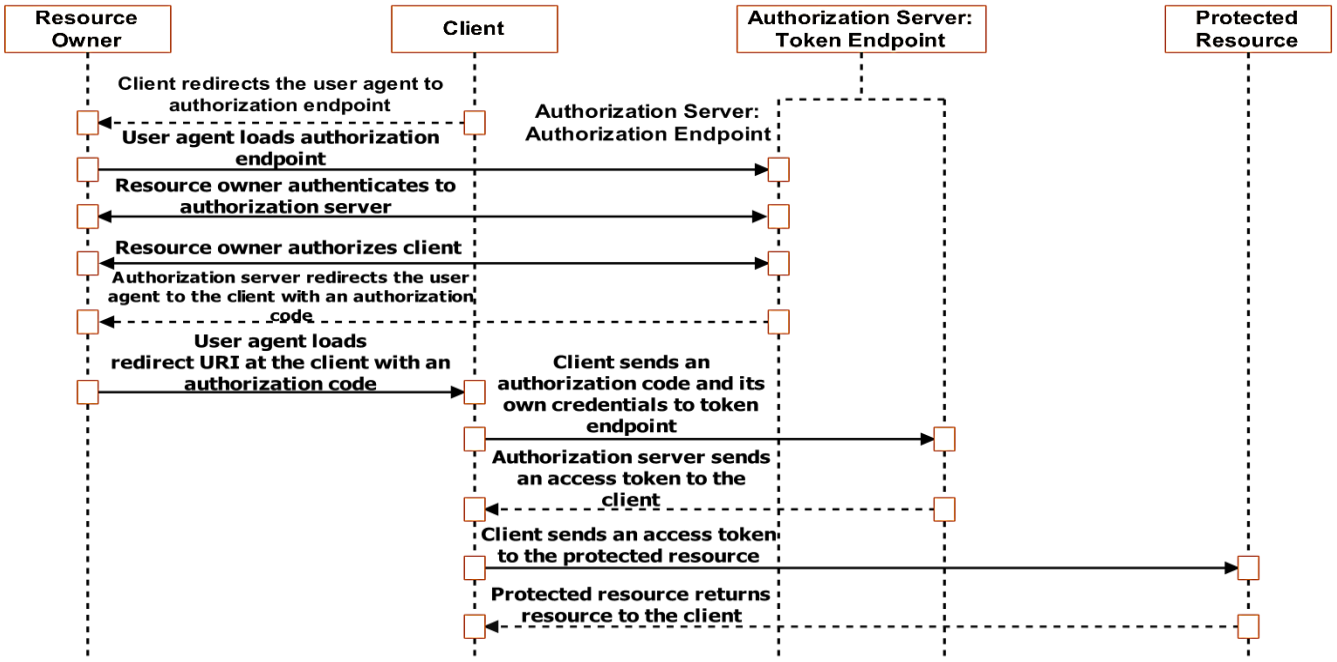

Figure 2.12. The detailed scenario of the authorization code grant

B. Implicit authorization flow

Implicit flow is the second type of OAuth 2.0 framework, in which the consumer accesses the authorization server to get the token for accessing. This flow is utilized in both mobile and web-based applications/services. The confidentiality of service is assured with this flow. The redirection is also used in the implicit flow. The token is handed to the user-agent, who then passes it to the application/service. The application is not directly authenticated in this flow. The action is redirected through the URI [69], [72]. The scenario of this flow is depicted step by step in Figure 2.13.

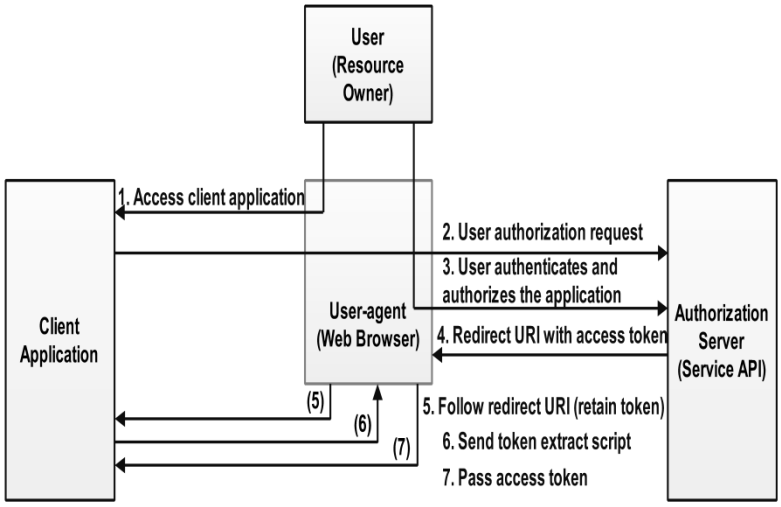

Figure 2.13. The implicit authorization flow step-by-step scenario

C. Resource owner password credentials flow

The third type is resource owner password credentials as shown in Figure 2.14. It is an authorization grant type in which the consumer provides the consumer ID and password for the application/service. Credentials are used by the application to obtain a service access token. Such a flow can be used whether no other flaws are enabled by the API service - in this case, the user should trust the respective application (service) [69], [71].

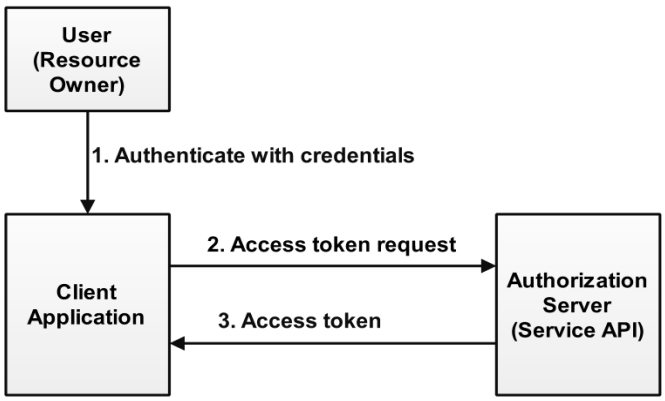

Figure 2.14. Resource owner password credentials flow

D. Client credentials flow

Client credentials flow (Figure 2.15) is the last type that supposes the authorization server need to have trust in the client application, so the authorization server authorizes all controls of the authorization to the client application. Then, the client application is allowed to access its service [69], [71].

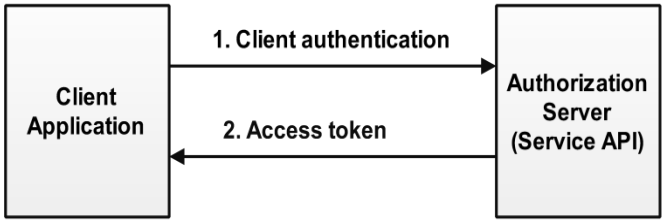

Figure 2.15. Client credentials flow

2.3.2. RESTful authorization and authorization management

There are several widely used methods for implementing RESTful authorization, such as employing tokens, keys, and cookies. A popular way to handle RESTful authorization involves using JWTs, which serve as a type of access token capable of authenticating and authorizing permission to

users or applications. Another way to manage RESTful authorization is by utilizing API keys. These are distinctive identifiers employed to verify the authenticity of users or applications. API keys are usually created by the API provider and then given to authorized clients. These clients can then utilize these keys to gain access to particular resources within the API. Lastly, cookies can also serve as a means of handling RESTful authorization. Cookies are commonly employed to keep track of a session's status and can be handy for storing authentication details, like a user's login information or an access token. Ensuring that RESTful authorization is securely implemented is of utmost importance. This can involve using secure communication protocols like HTTPS and employing more robust methods to protect sensitive data [4].

2.4. Service interface description

2.4.1. Web service description language

The functionality offered by web services is described using Web Service Description Language (WSDL) that considers as XML format and it is mainly focused on interface specification language. It is regarded as a standard for describing all the data required to interact with the web service. A web service is described and defined using WSDL, allowing the data description as the format based on machine-readable and elements like how the service was invoked based on parameters that are accepted, and what kind of data structures it proceeds. The current version of WSDL is 2.0; WSDL 1.1 was the previous one (Figure 2.16) [73].

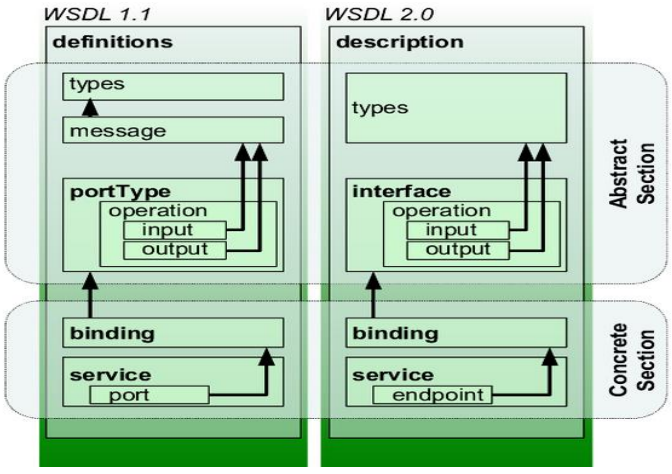

Figure 2.16. The WSDL versions [73]

Network endpoints as well as ports are regarded as WSDL-described services. The WSDL definition utilizes an XML format to describe them [73].

There are two primary components to the service description:

- **Functional representation (description).** This defines the specifics and the way web services are invoked. In this instance, the emphasis is on the specifics of message syntax and the configuration of the protocol network to convey the message.
- **Nonfunctional representation (description).** By adding additional headers, it enables the inclusion of other information in messages, such as the requester's security rules and instructions for how to operate in a certain environment.

A WSDL document is usually regarded as a collection of definitions that all have the same root element. The following XML elements are used to define the services:

- Type element that is considered as the data type.
- Message element that is regarded as the method.
- PortType element that can be thought of as the interface.
- Binding element that can be thought like the coded schema.
- Port element that can be regarded as the URL.
- Service element that can be considered as the collection of URLs [74].

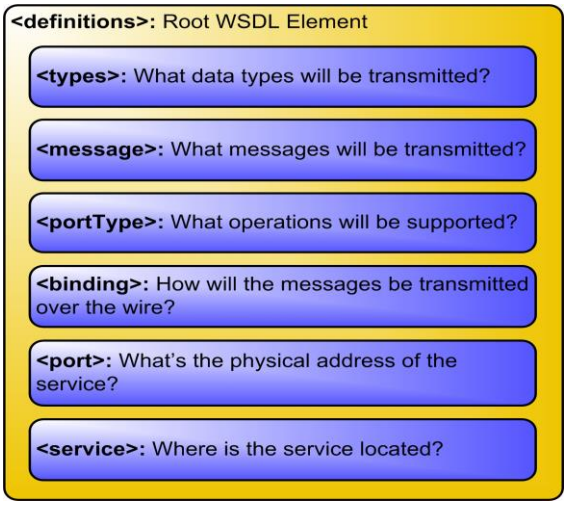

Figure 2.17. The service definition based on XML elements [13]

2.4.2. RESTful service description language

RESTful Service Description Language (RSDL) is regarded as the XML vocabulary and it is used for the purpose of designing, describing, and documenting RESTful services. REST is designed using RSDL, which is purely hypermedia-driven. The service should focus on the design of resources, linkages, and media formats, according to RSDL, and it should have an access point. It is beneficial for designers to consider a simple interface during the design stage. Through RSDL can be described the general operation of the RESTful services as well as their semantics. Resources may be discoverable from the home resource through the links, and RESTful services should possess an access (entry) point that resembles the home resource. These design constraints are made possible by the RSDL description's organizational structure.

Through RSDL are described the following elements:

- Media type, including the documentation and optional association to a schema or depiction that addresses all media types used by a service. This involves the various response bodies that the server can provide, how to search for links inside them, and how to spot particular types of links.
- Resources, making it possible for the service access (entry) point to be designed as a single resource.
- Links to every resource, as well as the resources that each corresponds to.
- Methods permitted for resources, as well as requests and responses related to them.
- HTTP headers, as well as custom headers.
- Authentication methods and identity providers that can be set up for the whole service or just a particular resource.
- URI templates and parameterized URIs.
- HTTP status codes are depicted at the service level (rather than at the request or resource level); although they can also be used as a reference at the request level whenever needed [45].

Consideration is given to the specifications of service interconnections, linkages, URIs, resources, methods, representations, and media types. An amount of resources that can be found using URIs is presented by RESTful services. There should be an access (entry) point for utilizing a service.

The home resource uses the URI to publish the entry point of the service. Other links to resources are found by using links that relate to the home resource information. Links express resource-related URIs that can be used in HTTP requests. The user may use any HTTP method in JSON or XML to provide a resource. Presentation is used to show the resource's real state as a series of bytes, with the representation syntax and semantics determined by the appropriate media type.

Presentation differs from resources in that it depicts the necessary information and their current state. We are unable to send HTTP requests for presentations; only resources may do this. Furthermore, a resource's presentation is the only thing we can parse or generate. These distinctions, which are outlined in the RSDL framework, are seen as being particularly significant. Media types can be used to offer schemas or specifications, for instance, which characterize the presentation's structure. The resource is a crucial element of RESTful services since it describes an HTTP request and how it can be applied to additional requests.

Links are regarded as the objects that show the associations between resources in the RSDL language. Their presentation is performed using media types. Resources can be represented in XML and JSON, and links can be used to transmit them in various ways using the media types. With the intention of making the development and implementation of REST designs relatively easy, the information provided for the REST design part should be expressed as clearly as possible in its structure. The URI's design and structure are taken into consideration as a particular element in this case.

The URI should be treated as opaque by clients, whereas URI as well as its response should be understood by the server. The URI structure is essential because it frequently holds metadata that is useful for anyone who wants to learn about APIs interactively. The URI variables are specified through the URI templates that are defined in the RSDL. The XML format, as well as other formats that are introduced in a form that is relevant to the RSDL description, can be used to describe the elements of the services and resources, but it can also include the schema documentation and media types. This makes it clear why REST information is defined in enormous size. Here, a semantic form that primarily uses XML or JSON elements and properties is also included [75].

For example, in the media types of HTML, the semantic refers to all specified browser behaviors that may be inferred from available data but exclude the use of links. A specific media type also contributes to the protocol semantics. For instance, in the Atom Protocol, it is possible for adding a new resource to a URI collection via a POST operation. Web Application Description Language (WADL) was first seen as a suitable description language for REST. It provides an in-depth and comprehensive description of the REST interface. This description language's structure is influenced by URI patterns and the implementation of server-side considerations takes precedence over the design of the hypermedia-driven.

WADL is criticized primarily for fusing client and server URIs that can lead to existing clients breaking the code of the server-side, and it is not regarded to be appropriate for describing REST services. If WADL is analyzed from the REST viewpoint, interfaces are exposed through static metadata, which does not adhere to the norm for data description and link discovery as much as it should. Furthermore, it provides tools for producing stub code - the kind of code that includes static metadata for clients and servers.

In addition to supporting URI and fixed routes across links and documenting problems over HTTP handling, WADL exposes URI and fixed routes explicitly. Data implied in the server and data provided to the client interface are both supported by WADL. When it comes to blocking data provided, RSDL and WADL should be similar in some circumstances. Similar to WADL, RSDL depicts resources and the linkages that lead to them. The RSDL relationships are always described by the links.

Along with WADL, RSDL also depicts resource distribution methods. When a particular semantic is demanded, RSDL uses HTTP status codes rather than describing intended error codes for every operation. It provides the global documentation of a certain status code's semantics for services. Using RSDL, it is possible to describe and provide schemas in a variety of formats, including text-based representations, JSON and XML schemas, and formats that are addressed by URIs [75].

3. AUTHENTICATION AND AUTHORIZATION IN SERVICE-ORIENTED GRID ARCHITECTURE

Systems and applications that enable resources/services to be integrated, managed, and distributed through numerous control domains are implied like a grid [76]. It is considered a distributed computer system that allows geographically distributed resources to be aggregated across the Virtual Organization (VO) participants. They are often known as members [77], [78]. Grid computing is considered a distributed computing infrastructure based on heterogeneous resources distributed over geographical areas. It controls and coordinates the resources' distribution and uses them dynamically, scalable, and in distributed VOs [9]. VO includes individuals, resources, and services that they associate with common purposes, but they are not locally placed in the single administrative domain [77].

Certification, group membership, and authorization contribute to the relationships between VOs. They are considered as a cover for relationships between members and their parent organizations. Such overlay enables trust, security mechanisms, and policies [76]. VOs consist of standalone autonomous domains and they are grid systems in an SOA. Therefore, the grid architecture works and the basis on web services and interconnection technologies. The grid must also have high-level security [79], [80].

In the grid, security is considered a serious concern because VO enables the distribution of resources across members/organizations and it needs to define who has authorization and how it can have accessing resources. Based on this issue, we have two basic concepts of security, i.e., authentication which provides the entities (e.g., the users) to be true that they want to be, and authorization, which defines the access degree of the entities that must be authenticated. These underlying issues contribute to the security infrastructure development of grids [76].

Authentication is a process (a mechanism) that enables confirmation of the claimed identity for systems that enables identifying their entities securely, while authorization is considered a systematized mechanism of the system that determines the access level of authenticated entities in secured resources. These two steps must be executed sequentially to provide the appropriate level of

network security. The grid is considered an open, distributed, loosely coupled network and heterogeneous [81], [82].

The grid is well-founded on SOA and VO as an integral part of it also belongs to this architecture. Users access grids through web services (science gateways) - portals between VO members (users) and grid resources. VOs consist of independent autonomous domains, where users and resources/services are not in single locations in a security domain and each VO has its security policies [83].

This section addresses authentication mechanisms and authorization mechanisms for controlling the access of distributed applications (services) and resources that run as grid services. The Open Grid Service Architecture [84] is taken as a model and grid service is described by WSDL extensions. It is accessed by using SOAP [85].

The section is organized as follows. In subsections are discussed the challenges faced the grid security, mainly in control and security policies, their coordination and distribution using resources, and VO third parties. Such challenges are addressed by grid security mechanisms through the domain policies decided in the VO. Here are studied and described the challenges faced by the grid authentication mechanisms, including credential confidentiality, flexibility, and encryption. SSO is included in this section as the main challenge of grid computing authentication. It is described the grid authentication models and mechanisms. There are presented authentication based on the Kerberos mechanism, password mechanism, and certificate.

Another subsection is included the grid authorization models and mechanisms. The push model that is functionalized through the subject of authorization is included the authorization mechanisms like CAS and VOMS, while the pull model, which is enabled through the direct contact of the subject with the resource, are included the authorization mechanisms like AKENTI and PREMIS. Grid authentication technologies and infrastructures such as GSI, Globus Toolkit, and authentication mechanisms such as PKI and X.509 are discussed in section six.

In the next subsection is described the grid authorization technologies and infrastructures. Authorization mechanisms such as CAS, VOMS, and PREMIS are described in detail, while the

VOMS and PREMIS architectures are schematically presented. EGI is presented in one subsection of this section, which is the e-infrastructure collaboration with advanced computing and data services for academic institutions and industry. Significant details are also provided for the EGI Check-In service and its high-level architecture. In the last subsections are included discussions and concluding remarks about the service security improvement.

3.1. Grid security challenges

Scalability, dynamism, and distributed virtual organizations as different groups and distributed users who share different resources in a coordinated way are elements that support and promote security requirements in large environments. Considering the security perspective of VOs, their key elements are participants and resources conducted by the policies and rules of classical members/organizations. The fundamental requirement is the VO access to resources of classical organizations that contain policies related to the local users [76], [86].

Such access should be established as an aspect of the binary trusted relationship among local users and organizations, and users and VO. It is assumed that trusted relationships among classical organizations and VOs or their members may not have as a whole. Such challenges are treated through the VO. They are addressed by grid security mechanisms through domain policies deployed in VO (as in Figure 3.1). Resources or organizations transfer some control policies to the VO third party, who coordinates relevant policies consistently to allow the distribution and coordinated use of resources [86].

Grid security can be complicated if new services/resources are deployed and occur constant changes during VO's lifetime. Such an example could be when a user creates stateful personal interfaces for current resources, or the VO creates directory services to maintain the data of members/participants. Like their static homologous, resources are secured and well-coordinated when interacting with other services [76].

Dynamic policy overlays combined in this form and dynamically developed entities bring the necessity for creating a grid security model with three functions as follows:

1 *Multiple security mechanisms.* Organizations as the VO portion should contribute sufficiently by investing in existing security models and infrastructure. The aim of grid security is to interplay with mechanisms and not to replace them.

2 *Services must be dynamically created.* Users should be able for creating new services productively and dynamically without administrator support. Services must interact securely and coordinate with other services. In this way, we have the possibility to name a service with the trusted identity and to give rights for the relevant identity without violating the local government policies.

3 *Trust domains must be dynamically established.* VOs must enable trust across their users and resources. The resources of VOs must be coordinated. The trusted domains can include as many organizations that should adapt dynamically when participants join the organization(s), even if they are created or removed from the VO. The conventional security management tools mostly included manual database editing and credentialing policies that did not meet the requirements of dynamic cases. The user-driven security model is considered an adequate model and enables users for creating domain entities and policies. Then, they can create and arrange resources/services within VOs [76].

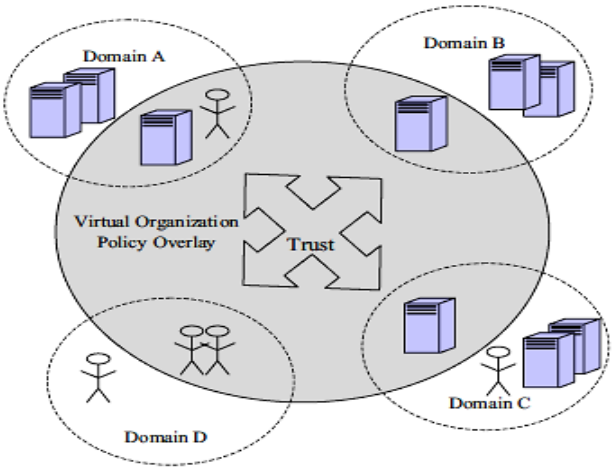

Figure 3.1. The VO policy overlay brings along participants out of various domains within a typical trust domain [76]

3.2. Grid authentication challenges

Different challenges are encountered in the authentication mechanisms such as credential confidentiality, and flexible and easy-to-use encryption. In the case of designing the authentication mechanisms, we encounter these challenges: the scalability issue - users are required to remember multiple combinations of usernames and passwords for their accounts, authentication as a technique (operation) is awkward and time-consuming, a single identity can be secured through multi-factor authentication, which can involve methods such as using a PIN code or obtaining out-of-band approval [87]. The authentication implementer encounters such challenges due to the features of the grid computing environment. In this session, we will discuss the main challenge of grid computing authentication, like SSO [76].

3.2.1. SSO in grid services

Single sign-on (SSO) is a property that relates to access control as well as it enables multiple actions to be taken by the user without having to do multiple times authentications. SSO enables two main aspects: (1) the user makes numerous requests for different services and (2) it delegates the rights/permissions to the authorized service(s) for enabling requests to other services in the name of the user. SSO is accomplished through a short-term cryptographic token on the basis of manual authentication done by the user (i.e. by providing his/her password) [87].

The token is used to authenticate further actions. It enables the delegation of services to remote systems. SSO is sustained by the security technologies like Kerberos, X.509 [88], and SAML. It is critical for use in distributed systems. Services authenticated through credentials (usernames/passwords) may be difficult to support by the SSO, due it is required protocol changes in specific situations. If changes occur in the protocol, the client and server of the existing software may be affected by major changes. Such cases can lead to compatibility problems. The SSO implementation in such services practically enables the user's password to be stored locally. It can be re-authenticated using this password [76].

For example, client-side X.509 authentication is sustained by web browsers and servers, but it is rarely used, because most web authentication is done through credentials. In cases the delegation is

requested; the user password is delivered to the remote service. This service imitates the user. Such behavior is insecure, but necessary due to the specific limitations that exist in the software. Grid enables a single login service for users and distributed resources through authentication mechanisms. The SSO service disadvantages: SSO service brings forth a centralized trust point for all grid users, but its universal adoption is limited, and integrating it can incur additional costs [87].

3.3. Grid authentication models and mechanisms

When we talk about the entity's identity, particularly a user in the specific context of a VO, then we are dealing with the authentication issue [76]. The integrity determines the strength of the authentication mechanism. In the password mechanisms, integrity depends on the password secret, and if it is discovered then can easily use. In terms of tokens, integrity depends on the hardness of obtaining and the possibility of copying the genuine token. The integrity in biometrics depends on the aptitude to get the biometric characteristics of an objective and the construction of an object that will deceive the system [87].

Secure and robust authentications are elements that the authentication mechanism must have on the grid environment. It also must achieve larger scale requirements in the distributed grid environment [76]. In the following subsections will discuss several main models and mechanisms of authentication for service-oriented grid architecture.

3.3.1. Certificate authentication

Certificate-based authentication can be considered the most prevalent authentication mechanism in the grid. This authentication type is sustained by public key infrastructure (PKI) to offer trusted authority tools for signing information. In this aspect, each entity possesses a public key (PK) which is consisted of cryptographic credentials and it enables certificate generation, as in X.509 certificate [76].

Certificates are usually signed and certified by an entrusted certificate authority (CA), which has information below (as in Figure 3.2):

- The subject name as a distinguished name (DN). DN is unique and serves to point out the individual or object representing the certificate.

- PK of the subject.
- A CA is the identity that signs the certificate. This is requested to prove that PK and identity are for the subject.
- Digital signature named by the CA.

A certificate request is submitted after the user has generated a public-private key. The private key is saved encrypted and utilized by the user, whereas the PK is placed in the certificate request. The user then sends a certificate request to the CA, sometimes encrypted by using the CA public key. After realizing the successful verification of the above process, the CA produces the certificate with the necessary information like the user's PK and the expiration date of the certificate. Then user signs the certificate utilizing its private key. The grid system relies on the PKI to utilize various certificates. Instead of using “long-lived digital certificates”, they benefit from proxy certificates.

The proxy certificate is a particular type of the X.509 certificate that it doesn’t need a signature from the certification authority. Proxy certificates are short time lived (up to some hours) and usually generated along the login phase. They are offered to users when existing the SSO mechanisms and credential delegation cases. The challenge of these certificates is the nonexistence of the revocation mechanism to avoid existing certificates in the system. The goal of the MyProxy server is to manage the user credentials to facilitate the duty of user maintenance and management. MyProxy as an online repository is located on a secure host and it serves to store credentials for the grid.

The MyProxy repository enables storing the certificate as well as the private key of users, and it enables them to obtain short-term proxy certificates from the repository when they have needed. Such capabilities are considered an advantage for storing the X.509 certificate and the user's private key on the user's own device. The recent tools and technologies needed to manage PKI certificates are complex. They consider the disadvantages and obstacles of this infrastructure [76].

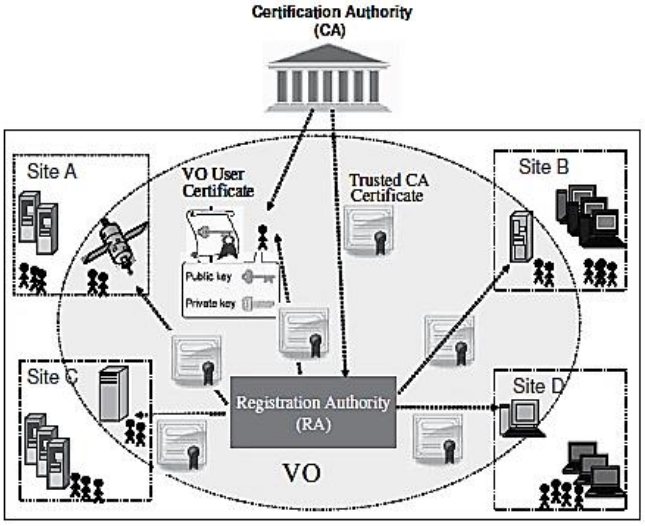

Figure 3.2. Grid authentication based on certificate model [87]

3.3.2. Kerberos authentication

Kerberos is an authentication mechanism that relied on the secret key that enables authentication on the network. As the third-party authentication mechanism, it comprises a trusted central authentication service, the Key Distribution Centre (KDC) for authentication users - uniformly distributing the ticket and mutually authenticating the user within a single Kerberos realm to enable the provision of SSO functionality [87]. The main two components of KDC are the Ticket Granting Server (TGS) and the Authentication Server (AS). Components work together to enable centralized user authentication. User(s) and service(s) distribute a secret key to the KDC, while it issues tickets that show the identity of the owner(s) [76].

The ticket possesses a limited lifetime and the user can connect through it to the end-service. In Figure 3.3 is shown the authentication of Kerberos Central Trusted Third Party (KCTTP) Grid. Fulfilling the essential specifications for grid authentication, maintaining authentication identity, and significantly improving security are credible advantages that Kerberos provides [87].

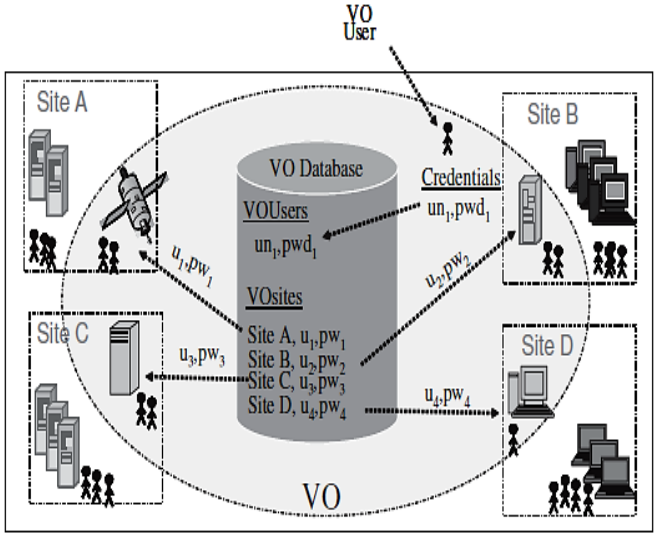

Figure 3.3. Authentication of the KCTTP Grid [76]

In Figure 3.4 are shown the necessary actions to request the service by following the steps:

1 The Authentication Server (AS) is the object where the client first connects and then it requests access to the Ticket Granting Server (TGS).

2 The TGS ticket is generated and encrypted by the AS together with the TGS public key. A random key of the session is generated by TGS and used later between the client and TGS. The whole data is encrypted through the client's private key and it is taken as the client's password.

3 Data about the ticket authentication and the target server name are sent through the client to the TGS. Time, client data, etc., are included as parameters in the authentication data. The TGS client's ticket is encrypted by the client through the public key of TGS.

4 TGS responds to the client with a ticket, which is then used for accessing the server [76].

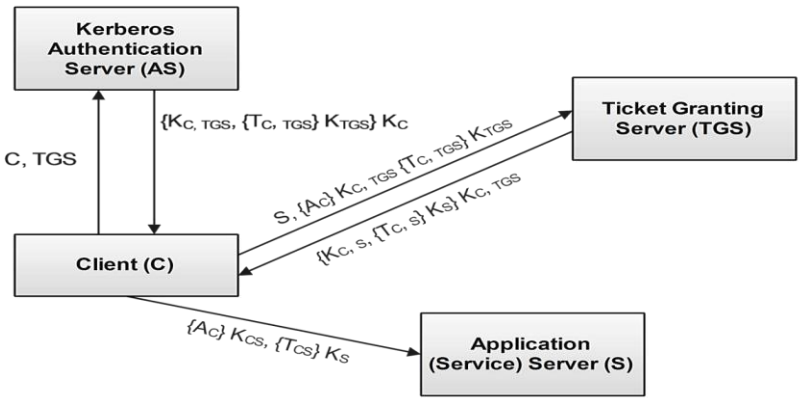

Figure 3.4. The necessary actions to request the service in the Kerberos system [76]

Kerberos fulfills the basic specifications for grid authentication. It presents some obstacles in the authentication of users/services. Whenever user authentication is required, the KDC also needs to intervene because it maintains the whole users and services list. Cross-realm authentication is very difficult to implement through Kerberos, and it is considered inter-organizational authentication. Every independent KDC serves a realm, so the creation of a trusted cross-realm connection is considered an obstacle for deploying Kerberos on the environments of the grid [76]. The Kerberos approach is more suited to local security environments (campuses, enterprises, etc.), while it is not suitable for high-performance and loosely coupled environments [89].

3.3.3. Password authentication in grid computing

Most computer systems perform the authentication by using the password because it is easy for implementing as well as computationally inexpensive. A user's password control, in spite of a hash password saved in some databases, is enabled by authentication. Controlling as a process is easy and does not request complicated calculations. The insufficiency of confidentiality is a reason that passwords must be encrypted for preventing unauthorized users from the system login. Choosing a complex password is considered a hard task due have tools that automatically can find the relevant password by using different methods like brute force [76].

Static and dynamic passwords are two passwords-based authentication mechanisms:

1 *Static passwords.* Traditional static authentication has a mechanism mainly consisting of credentials, and it is opposite to the dynamic access of users to the grid and the complication of the grid environment. The static password mechanism is not secure, easily vulnerable to threats, vocabulary, and bypassing attacks. Its mechanism can easily be broken. It is considered an inadequate mechanism for sensitive computing environments.
2 *Dynamic passwords.* It is also referred like a one-time password and its mechanism generates a dual operator factor, a fixed user identification code (private key), and changeful factors (i.e., time, random number, etc.). If the attacker finds the password cannot use and falsify the user identity, because of the security that contains this authentication mechanism. It is resistant to the various attacks that threaten the password [87].

In the Table 3.1 are shown the limitations and threats of the static password authentication mechanisms compared to the dynamic password authentication mechanisms. Even though the dynamic password-based authentication mechanism has advanced, it is difficult to use in grid computing environments due to some challenges that emerge in its implementation. Several challenges are: the difficulty in fulfilling the grid authentication demands like SSO, mutual authentication, and delegation of credentials through password authentication. However, passwords are commonly used to log in to the grid infrastructure (e.g., private keys in the MyProxy repository, tickets in Kerberos, etc.) [76].

Table 3.1. Comparison between static and dynamic password authentication mechanisms [76]

	Static Passwords	*Dynamic Passwords*
Anti-denies	Anti-denies information is not contained.	Dynamic passwords can only be produced by password investors, and they have anti-denies information.
Ani-fabrication	It can be simply guessed or fabricated.	Such passwords possess the cipher-text and do not have a one-to-one pair of clear and cipher text. They are very difficult to guess and analyze.
Anti-replay	Not supported.	Supported.
Ani-	It can be easily	There are cancellation and overtime cancellation functions, so

exposition	decrypted.	the disclosure of them poses no risk.

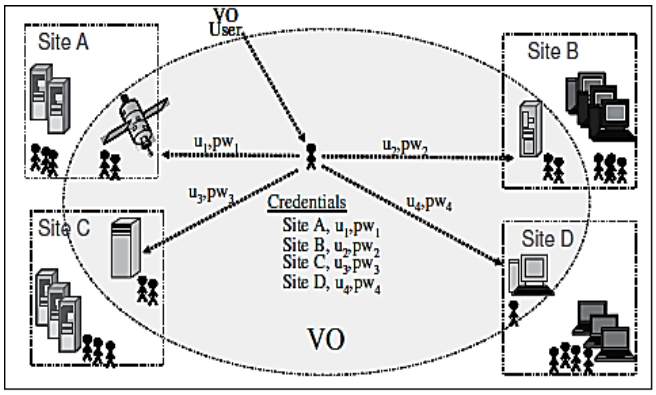

Figure 3.5. Grid authentication based on password mechanisms [87]

3.3.4. Public key authentication in grid computing

Certificate-based authentication is considered the most prevalent for use in grid authentication mechanisms. To provide trusted authority tools, it relies on PKI and enables the information signature which can be used for authentication purposes. Here, entities have encrypted credentials that are public key-based and used in the creation of a certificate, like the X.509 certificate. The certificates are signed and certified by a trusted certification authority (CA) as described in Figure 3.6. The fundamental components are considered PKI-based certificates which are de facto as the standard in grid security infrastructure [76]. In distributed environments such as grids, they provide convenient, portable, and scalable authentication mechanisms. We have many requirements for grid authentication, but one of them is trust delegation. It is enabled by PK authentication using a proxy certificate [87].

Several issues should be considered:

1 *Reconcilability between various PKIs.* The validity of the path certificate is to be appropriated for X.509 and in accordance with the other PKIs. The deployment of trust between many CAs occurs after the certificate policy (CP) is reviewed and verified that both provide equivalent elements of trust and verification throughout the process as a whole where they operate.

2 *Certificate revocation.* When the amount of grid users is increased, then the grid administrators encounter serious concerns in managing and distributing security, managing usernames and passwords, and revoking compromised credentials. In certain cases, the user's private key may be missed or compromised. Therefore, a way should be found to revoke the specified public key certificate (PKC). The revocation of the certificate is a problem in this case, and it has been shown since PKC was used. A certificate revocation list (CRL) is considered an old revocation method, but widely used. It can be taken as a basis for publishing the revoked certificates list. To validate a signature, the relying party is required to retrieve from a repository the complete chain of certificates leading to the trusted root, along with any available revocation data pertaining to those certificates. If CRLs do not update in real-time, then long intervals between the distributions of CRLs result in their stagnation and failure to revoke information when needed. The distribution of CRLs is expensive and they are sensitive and vulnerable towards denial of service attacks even if they are simple.

3 *Poor usability.* Users must have valid credentials for use in the grid. Obtaining credentials takes time, and in certain cases, prospective users are required to be interviewed. It is discouraged in certain zones if there is no grid CA or Certificate Registration Authority (RA) [87].

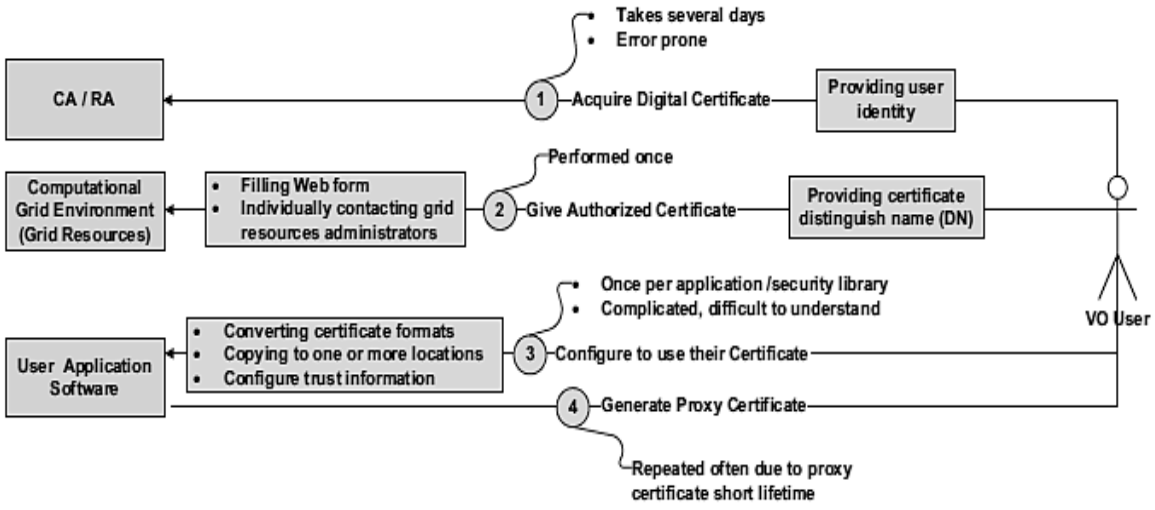

Figure 3.6. Authentication as a process based on the certificate [76]

3.4. Grid authorization models and mechanisms

The GSI is considered the standard of grid security infrastructure, while the authorization is functioned through the grid-map file [90]. To provide data about the mapping between the identities of the grid and local users like the UNIX account grid-map file is used. After successful mapping, the local identity uses for enforcing the decisions of the local policy like the file access. Grid-map file authorization can be lightly implemented, but there are disadvantages like the absence of scalability and consistency [76].

To enable the delegated rights of users for proxies to other systems, the resource providers have needed authorization service for users' proxies to appear on their systems. The process of transferring users' rights to services or proxies is called delegation. The delegation of more rights may lead to the possibility of their abuse. When we delegate few rights may not happen fully execution by proxies. Therefore, it needs an authorization service that restricts user rights for proxies and resource providers that control the validity of using the delegated rights. The authorization restrictions can be overcome through alternative authorization solutions. These solutions result in two models of authorization: the push and pull model [76].

3.4.1. Push model

The push model has functioned when the authorization subject (a grid user) first contacts the authority to get its authorization rights. Then the authority releases and sends a token/message to the consumer (user) which included its access rights. A token is applied by the subject to request a specific resource/service. Service owner based on user token - the information received, it accepts or rejects its request and replies to it for the requested subject [76].

3.4.1.1. Community Authorization Service

The maintenance of the authorization data for community entities is usually an essential issue and this should be considered even strengthening fine-grained policies on access control. The server of the Community Authorization Service (CAS) is designed for the above-mentioned aspects (Figure 3.7). CAS stores the rights of the user in a backend database and it is available as a centralized, trusted, and third-party authorization system. To show which permissions are given for which users/resources, the

CAS server has the policy statements, which regulate and control permissions [91]. The delegation of permission from the CAS is done for all user rights groups according to roles in the community. The user's rights are determined by the overlapping rights granted by both the resource provider to the community and the community to the user [76].

The decisions of authorization are applied at the resource level when the user represents his rights for accessing a resource(s). The authentication of the public key and delegation instruments of the GSI are the main elements that CAS is implemented and designed to work. The community user first generates the proxy certificate marked with its credentials and then it can access a specific resource. Then, the proxy certificate of the user is represented by the user to the CAS server, while the CAS server issues the new certificate known as the CAS proxy certificate. This certificate includes the user rights and opportunities to access specific resources.

SAML is the standard for the policy assertions applied through the CAS server. After the user has its CAS certificate can provide the resource with it. Whereas, after analyzing the assertions of the CAS policy from the relevant resource, it is determined which operations are allocated to be performed by the user.

The number of users and resource providers is offered as scalable by CAS. The resource providers and users should be presented to the CAS server. They should be well-known and trusted it. Usually, CAS releases a certificate that is distinguishable from the GSI certificate. Therefore, user account mapping can be complicated. If numerous users try accessing the CAS server simultaneously, it may fail partially or totally [76].

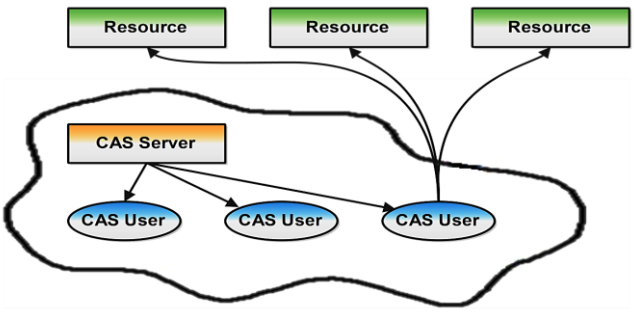

Figure 3.7. The CAS structure [76]

3.4.1.2. Virtual organization membership service

The functionality of the DataGrid project to be completed, it must be secured with authorization tools, and for this purpose initially the Virtual Organization Membership Service (VOMS) is used. The management of group memberships and their rights is done separately at VOMS. Group membership management is done by VOMS, while its rights are retained by the source site. Policy assertions are typically issued by VOMS for the user who has a role list or group memberships. Consequently, the assertion interpretation, and determination of the rights of user access are the responsibility of the resource provider. Such rights are relied on user memberships and local policies [76].

The proxy certificate generated for the user for accessing the grid resource(s) is considered the first operation to be performed. It is generated for the user(s) by the VOMS. The proxy certificate issued to the user contains the role and its memberships like the non-critical extension. Authorization assertions are extracted from the resource provider according to the user's proxy certificate combined with the local policies and in this way is achieved to the decision of the authorization. The authorization decision is made when the resource provider depended on two supplementary services: Local Centre Authorization System (LCAS) and Local Credential Mapping Service (LCMAPS). LCAS is presented as a service within the resource site for enforcing the policies of local security, whereas LCMAPS is responsible for associating a user with their corresponding local credentials [76].

The utilization of a normal proxy certificate with a non-critical extension that contains authorization data is considered a benefit of the VOMS system. Such certificates issued by the VOMS may also use for services that are not provided by it because their extra data is ignored and then certificates can use for other services. This simplifies the process of mapping the user account and the resource provider has possible to figure out the user through a proxy certificate. In the aspect of centralized authorization, VOMS does not enable the right solution because service providers must bind group memberships to source permissions. This situation can lead to consistent problems, in cases when policies change dynamically [76].

3.4.2. Pull model

In this model is enabled direct contact of the subject with the resource. Then it communicates by the authorizing authority to define the user access rights. The authority gets a decision about the authorization and sends back the response to the resource. The result influences that the resource should be allowed or denied service towards the subject [76].

3.4.2.1. AKENTI

Akenti is the authorization system infrastructure that implements fine-grained policies on access control in organizations with resources that are distributed and controlled by numerous pairs. This system uses the X.509 certificate to authenticate whole entities included in the authorization. The resource in the Akenti can consist of multiple actors from different domains, which independently define its policy authorization. The policy of resource is presented by the series of certificates digitally signed by the actors involved in the resource. Such certificates are distributed on many sites and remotely stored. To get a decision on the authorization, the policy engine collects all appropriate certificates for the user and resource, verifies them, and defines user(s) rights in relation to the service/resource [76].

Akenti uses XML as the format through which the resource authorization policy is maintained. The policy is stored according to these certificates' types: the use-condition certificates, the attribute certificates, and the policy certificates. The use-condition certificates represent certificates consisting of conditions that restrict access to resources and rights. They are issued if certain conditions are met. Attribute certificates relate to the user attributes necessary to respect the usage restrictions. A policy certificate enables the application of policies in the name of the resource and it contains its name. It also includes a URL list that requires use-condition certificates. The policy certificate is signed, stored, and protected in a secure location. The user who tries to access the resource must wait until the gatekeeper contacts the server of the Akenti for finding out if it can make the specific operation on the resource. The server of the Akenti recaptures the relevant certificates to see and verify if the signature on the certificates is made by the proper issuer. It then evaluates the certificates and sends back the authorization decision. It allows or denies the resource access.

The decentralized and distributed policy on authorization that enables fine-grained authorization is available through Akenti. This enables direct control of the user's action and what it makes in the grid. Regardless of the authorization data saved in the distributed and signed certificates digitally, several access management functions are difficult to accomplish. The Akenti mainly encountered some actions like the presentation of the users and the VO resources like a list or questionable (external) authorization for specific resources [76].

3.4.2.2. PERMIS

Privilege and Role Management Infrastructure Standards (PERMIS) is an authorization system based on RBAC [88]. Permissions and community user permissions have their origin in the roles and they are determined to users by the site administrator. The identity, roles, and policies of authorization of the users are put in certificates as the authorization data. PERMIS uses these features. Certificates are distributed across many sites. The decisions of authorization are mainly taken by PERMIS, and for such decisions, the PERMIS authorization engine collects and verifies all user certificates. It then evaluates them regardless of the local resource access policies.

The authorization policy of the PERMIS is consisted of two mechanisms: the Role Allocation Policy (RAP) and the Target Access Policy (TAP). RAP defines the trusted managers who have the authority to assign specific roles to particular users. TAP outlines the authorized roles that have the permission to perform specific actions on designated resources [76]. Whereas, the gateway of the application includes the Application dependent Enforcement Function (AEF) and the Application independent Decision Function (ADF). The AEF module of the resource is contacted through the user. It controls the user’s attribute certificates, regardless of the RAP as well as valid attributes that are transferred to the ADF. According to the authorization policies marked by TAP, the ADP takes a decision for the right access and sends back the response to the AEF. In the name of the resource, the AEF implements the decision.

Roles in the community are fewer than the volume of users; therefore, the PERMIS provides greater access control management capabilities, lower costs, and scalability. The PERMIS has limitations when many communities exist in a federation. This happens because every community

contains its attribute certificate repository and it has difficulties to be maintained consistency between communities [76].

3.5. Grid authentication infrastructures and technologies

Through the GSI is made early attempts to build the infrastructure for authentication and authorization of the grid in the Globus Toolkit. The GSI is the security framework, where authentication is applied using the PKI and X.509 identity certificates on which PKI is based. X.509 certificate is attached to a private key and together with it creates the uniform credential set, which a grid entity uses for authentication to further grid entities, as in the case of resource providers. The support of PKI for SSO based on proxy certificates is considered a benefit and the reason why it is widely accepted within the grid community. It also facilitates the user in holding the X.509 certificate released by the trusted Certificate Authority (CA). The user can access resources through all sites in the VO grid and specific sites in one grid in particular, where it should be authorized to access specific services [92].

PKI-based authentication is used in some security infrastructures in grid environments. Such an approach is criticized for its barriers in usability, scalability, and controllability. Getting an X.509 certificate from a trusted CA is a difficult process for users and it takes time. In this way is hampered usability that is considered essential to succeed in security technology. Trust is another fundamental issue related to PKI. It is imperative that sites in the grid trust the users, other sites, and CAs. If the trust among these parties does not exist or wavers, then it influences to aggravate the process. Finally, the PKI may cause privacy problems, because user information (name, organization, etc.) is used for the authorization decisions [76].

3.6. Grid authorization infrastructures and technologies

Authorization mechanisms are necessary for distributing resources to the VO grid. They define the resources and the way how the authenticated user is authorized for accessing the relevant resources. The GSI is an effort with the purpose to implement authorization in the grid environments respectively in the Globus Toolkit through a grid-map file. Behind a grid-map file lays the Access Control List (ACL) mechanism that stores and maps a Distinguished Names (DN) list of grid users (local user

account names) [93]. Such an authorization mechanism doesn't permit the local resource providers to put access control policies. This affects in restriction of the authorization functionality. The grid-map file medium is not scalable in the case of describing user rights in large-scale VOs operating at the grid [92], [94].

CAS is a grid authorization system with the purpose of the users' authorization management improvement. The allocation of the authorization rights is delegated to the administrator of the community (the administrator of a VO) by the resource provider, and then it permits the community administrator to decide who is able to use the relevant allocation. The whole process is achieved when a CAS server is possible for acting as a trusted mediator between the users and resources of the VO. The CAS server sets if the user has sufficient permissions to perform certain actions on the role-based, it has in the community. In this way, it applies access control based on role.

Scalability is enabled by CAS for users and VOs. All users and resource providers must be recognized and trusted by the CAS server. In certain cases, the number of resource access requests cannot be very scalable when using a CAS server. When the amount of users is large and trying to access simultaneously the single CAS server, its likelihood of will be potentially high to fail. The maintenance of all user specifications is done by the VO's administrator, so it is charged with a difficult task, i.e. to manage the authorization workload [92].

The Virtual Organization Membership Service (VOMS) is an authorization system same to the CAS. It enables the delegation of authorization permissions and privileges from users to the VO administrator. In Figure 3.8 are described the operations and VO's architecture by specifying how the VO administrator maintains the centralized database which includes user-related information in the VO grid. It enables the distribution of attributes needed for users who require access to resources through the VO. The attribute certificate is estimated as the VOMS basis for the authorization aspect. The VOMS certificate is different from the proxy certificate used in the Globus Toolkit's GSI. It holds information received from the VOMS server, i.e. the user role and the group that it belongs.

The VOMS is not limited like the CAS system; it exceeds and improves the limits of this system. The CAS did not allow attribute certificates to be issued, while in the VOMS the opposite occurs. It allows those certificates to be issued and adds them as a proxy certificate extension. The

focus of VOMS is on managing users and privileges. The rendition of the attribute certificates is implemented by the resource provider through the VOMS. These certificates are based on the final authorization and they are allocated by VOMS. The main administrator of the VO is allowed by VOMS for delegating the membership maintenance and users' attributes to other administrators. In this way, it reduces the burden of the VO's main administrator [76].

The PERMIS is an authorization system infrastructure that enables access control based on role through using attribute certificates. Attribute certificates released by an authority responsible for attributing reflect the attributes and roles associated with a user, while resource providers establish policies to apply access rights assigned to each specific role or attribute. The resource providers define the user access rights based on the user attribute certificates [92].

PREMIS consists of two main components: a Policy Enforcement Point (PEP) and a Policy Decision Point (PDP). The responsibility of the PEP falls on accepting requests of the user by ensuring that whole requests for accessing data sources are performed through the PDP, while the PDP deals with the execution of authorization decisions. Figure 3.9 shows the interactions between the components of PREMIS and its architecture. The PREMIS engine serves for authorization decisions and it consists of two components: the PDP and the Credential Validation Service (CVS). The request made by the user for accessing PREMIS's protected resources goes through the PDP where it submits the request to PEP for accessing through the required credentials [92].

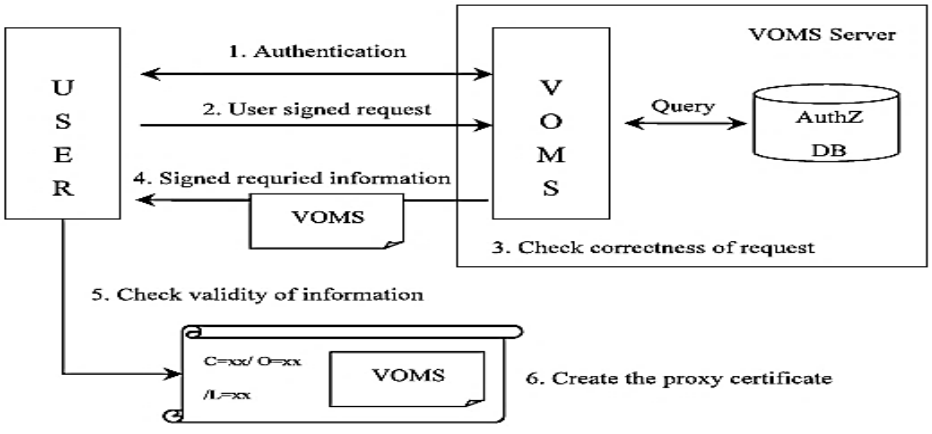

Figure 3.8. Operations and actions in the VOMS architecture [76]

The PEP then responds to the PDP and informs it about the user credentials found in the credential repository. After this step, the CVS validates the attribute tasks of the user according to its policy group by ascertaining that the authority enabled these attributes for this set of users. After the validation is completed, the user attributes are transferred to the PDP. It then decides based on its policies that users with this attribute set are allocated access to this resource with the condition they fulfill the required conditions [76].

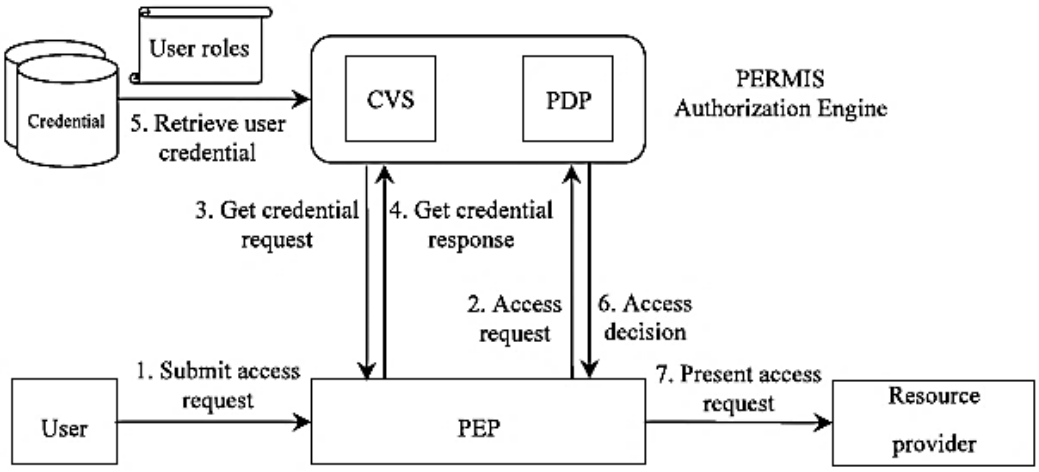

Figure 3.9. PREMIS architecture and interaction operations [76]

3.7. EGI

EGI [95] presents the e-infrastructure collaboration with advanced computing and data services for academic institutions and industry. Its services are provided to researchers, trainers, and innovators. This e-infrastructure includes the national e-infrastructures of many countries. It includes over 300 resource centers from Europe and further on.

Members involved in the EGI cooperate with each other and provide the following services:

a) *Compute services.* Clusters capable of high-throughput computing, IaaS based on the cloud, data computing support, and online hosting services beyond national institutions and mechanisms associated with EGI.

b) *Data services (storage services).* Opportunities and services for scientific communities such as storing, transferring, distributing, and archiving data about their community and the public. These services are provided in more than 300 countries and they allow their scientific communities to carry out the above-mentioned actions.
c) *Training services.* Training opportunities about EGI, including IT service management (FitSM) and ISO 27000 - information security standard. It provides the cloud infrastructure in the federation context required for the software training activities.
d) *Operational tools, processes, and protocols are employed to handle operational tasks efficiently in diverse and distributed infrastructures*. Such service is used by the members involved in the EGI infrastructure and scientific communities who must communicate and distribute their data as resources in a coordinated way [96].

EGI federation enables the distribution, allocation, computation, and storage of resources for scientific groups through VOs. In this case, the VO represents a scientific user community, in which the involved members work around the same field or similar areas of research, or are part of the scientific collaboration. They are involved in VO for research, applications, software services, and datasets or hardware resources. EGI has operated with a genuine VO structure since 2010. This infrastructure enables services and resources for large communities of users. There are teams capable of maintaining and providing IT support for EGI. They manage the functionality of VO services for researchers.

The integration and delivery of services to researchers, research teams, and members of research infrastructures was inevitable. Such actors, especially in the early stages of research, needed to use applications, computing, and similar services. In many cases, these elements were missing or impossible in their countries or partner institutions. This has happened due to the lack of sustainable e-infrastructure and institutions that provide such services in the countries they came from. Even those opportunities that existed were inadequate because of resource distribution and community policies. This has happened because from users were required:

- To get and use X.509 personal digital certificates from the certification authorities, which were recognized by the resource providers of EGI.

- To join an existing VO that is in line with the research/researcher's subject or purpose or create a new VO that is supported by other researchers and they are ready to support the user with their resources whether for computation or data storage.
- The integration of the scientific applications and their correlation with resources of VO to meet operational responsibilities, including managing VO membership, resource allocation negotiations, etc. [96].

To improve the above research problems, in the EGI is designed and developed an 'Application on Demand (AoDs)' service supported by FP7 funds of the EU. EGI AoDS fulfills the e-infrastructure dedicated to research and it serves research teams from all over Europe, without having the need to harmonize their work in any established community as a VO. It has no need for dedicated access for such purposes, to compute the resources for specific purposes of the community, the resource storing and Virtual Research Environments (VRE) or science gateways [76].

The AoDs service is created in order to patch up the existing EGI components. There are also developed new components that have enabled the integration of modules within a single service. AoDs consist of the components below:

1. *"The EGI Marketplace*
2. *The EGI Check-In service*
3. *The pool resources of the EGI Infrastructure*
4. *The Per-User Sub-Proxy (PUSP) certificate*
5. *Scientific applications*
6. *Applications hosting frameworks (VREs, science gateways)*
7. *Policies"* [96].

3.7.1. The EGI check-in service

Through the EGI check-in service is enabled access to the EGI services and resources using the authentication mechanisms. Check-in does an operation of the entity connector with the Identity Provider (IdPs), and it is mainly located outside the EGI ecosystem, while Service Providers (SPs) are part of the EGI or can be part of the associated partners. The user logs in to the service providers using the credentials and this is enabled through Check-in. The user attributes from different authorization

sources (Identity Providers and Attribute Providers) are aggregated by Check-in and it delivers these to the service providers. In this way, it also helps the different authorization sources to make authorization decisions based on the possession of information about authenticated users [76].

The benefit of this approach is that service and identity providers only need to establish and maintain a technological and trust relationship with one entity, EGI Check-in, rather than multiple relationships. Throughout this case, all hosting frameworks of the applications of the AoDs are organized as service providers in Check-in, whereas Google, Facebook, LinkedIn, ORCID, and EGI SSO are IdPs (Figure 3. 10).

The OpenID Connect Protocol (OIDC) or SAML 2 is used by Check-in to cooperate with SPs and IdPs. It uses the Per User Sub-Proxy mechanism to convert the username and password into short-term X.509 proxy certificates. They are utilized by computing and storage sites for authorization and authentication purposes [96].

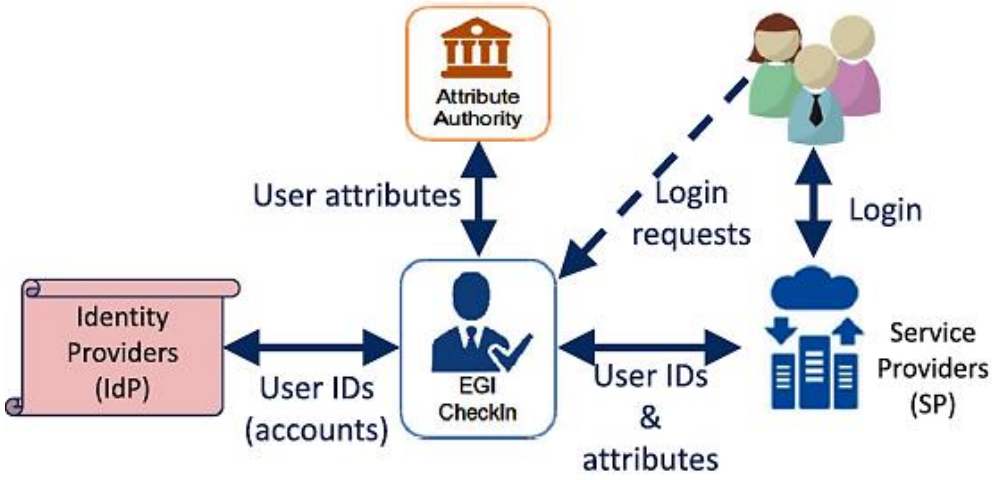

Figure 3.10. The EGI Check-in service high-level architecture [96]

3.8. Discussion

Grid computing is a distributed computing system that provides infrastructure for services based on heterogeneous resources distributed over geographical areas. It controls and coordinates the resources' distribution and uses them dynamically, scalable, and in distributed VOs. Therefore, the grid architecture works and the basis on web services and interconnection technologies. The grid must also have high-level security. Security is considered a serious grid issue because VOs enable the distribution of resources through members and organizations, which must be authenticated and authorized before accessing certain resources. Taking into consideration these challenging issues, authorization and authentication mechanisms, as well as the infrastructure and technologies in which they operate, are considered very important issues in grid systems.

Therefore, there have discussed the challenges faced by grid security, mainly control and security policies, their coordination and distribution using resources, and VO third party. Such challenges are addressed by grid security mechanisms through the domain policies decided in the VO. With such challenges are faced the grid authentication mechanisms, including credential confidentiality, flexibility, and encryption. There is studied in detail SSO, as the main challenge facing grid computing service authentication. This access control mechanism provides delegation of the rights/permissions to the authorized service(s) for enabling requests to other services in the name of the user. The authentication models and mechanisms enable the maintenance of security stability in grid computing.

The Kerberos mechanism, password mechanism, and certificate mechanism are represented schematically, and those schemes are presented in the authentication workflow using the above-mentioned mechanisms. Grid authorization models and mechanisms contribute to the functionality and security stability of grid computing. Here are discussed authorization mechanisms such as CAS and VOMS. CAS is the grid authorization and authentication system designed for authorization managing improvement and access control policies. The structure, operations, and interaction parties of this mechanism are presented schematically in this section. VOMS is an authorization system same to the CAS. It enables the delegation of authorization permissions and privileges for grid systems and services. The VOMS architecture, including workflow operations and actions, is schematically presented.

Akenti as the authorization system is discussed in this section. It is an authorization system infrastructure that implements fine-grained policies on access control in organizations with resources that are distributed and controlled by numerous pairs. PREMIS as an authorization system infrastructure that enables access control based on role through using attribute certificates, is discussed in this section. Here, its architecture and interactive operations are schematically presented. Finally, details are provided about the EGI system, which presents the e-infrastructure collaboration with advanced computing and data services for academic institutions and industry. The focus of this part has been the EGI Check-in service through which it enables access to EGI services and resources using authentication mechanisms. Authentication as a process through the EGI Check-in service is also presented schematically in this section.

3.9. Conclusion remarks

Authentication and authorization for services that operate on the grid, estimate as complex issues and they require effort and dedication in their implementation. In this section is surveyed the authentication and authorization in the services-oriented grid, respectively the challenges encountered in using and applying the grid authentication and authorization infrastructures by including technologies covering these two major grid security domains. It is also discussed grid authentication, authentication and authorization mechanisms and models, and authentication and authorization infrastructures in the grid. A recapitulation of existing authentication and authorization technologies and their applicability to service-oriented grid architecture is presented. Finally, we can say that no existing technologies solve authentication and authorization problems when acting alone.

In many cases, it is necessary to combine several technologies of grid security to resolve all issues associated with authorization and authentication. The integration of proper authentication and authorization mechanisms enables a stable security infrastructure that effectively fulfills all challenges. The authorization technologies like VOMS, PREMIS, and EGI enable inter-organizational authentication, role-based and fine-grained authorization. Such infrastructures offer promising authorization and authentication solutions for services in the grid architecture. In future work, an authentication and authorization management framework for grid services should be proposed. It will be used to manage and control unauthorized parties from being allowed access to specific resources or

services if previously they are not successfully authenticated and authorized in relevant grid system services.

4. AUTHENTICATION AND AUTHORIZATION IN SERVICE-ORIENTED CLOUD COMPUTING ARCHITECTURE

Cloud computing is a collection of various configurable computing resources such as networks, servers, storage, services, and applications, which can provide users with flexible as well as on-demand access [38], [52]. It also offers very cost-effective demand service and stability [38], [97], [98], highly efficient processing, and accessibility of resources. Cloud providers take responsibility for the optimization of resources [99]. The deployment models associate the aim and existence of cloud computing. There are three categories of the deployment model, respectively public cloud, private cloud, and hybrid cloud [50], [100]. Cloud computing essentially offers three separate service distribution models: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [50], [101], [102], [59], [103] as depicted in Figure 4.1.

In terms of cloud services, authorization, and authentication as the key issues should be ensured [52]. Cloud computing is needed robust mechanisms of authentication and authorization in order to protect and maintain its resources [38].

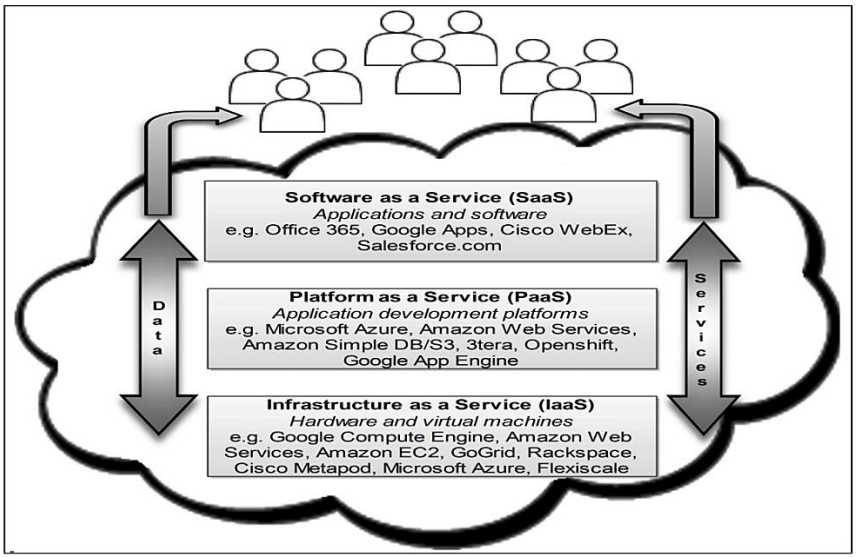

Figure 4.1. Cloud service distribution architecture [104]

In the cloud, security is considered a serious concern because the cloud service provider (CSP) enables the distribution of resources across members/organizations and it needs to define who has authorization and how it can have to access resources. Based on this issue we have two basic concepts of security, i.e., authentication is a process that enables confirmation of the claimed identity for systems that enables identifying their entities securely, while authorization is considered a systematized mechanism from the system that determines the access level of authenticated entities in secured resources. These two steps should be executed sequentially to provide the appropriate level of cloud security [105], [106], [107]. The convenient authentication and identification of cloud entities is essential in order to prevent unauthorized access. This is hard to maintain because many services as well as CSPs and their employees can access common resources [61].

The concern of the CSP is that its services and resources are accessible to authorized entities. If the cloud does not support and contain adequate access control policies, it will be often vulnerable to different threats and attacks [17]. Cloud computing can be attacked from conventional security attacks of systems: malicious code (viruses and Trojans), back door, Man-in-the-Middle, and Distributed Denial-of-Service (DoS) attacks, the API that is insecure, abuse and misuse of cloud computing, and malicious interference within the cloud. Because of these attacks, cloud services can be inaccessible and have bad impacts. For CSPs, ensuring that their services are completely accessible and usable at all times is an essential and main requirement. In such circumstances, resources can be faced with issues such as privacy and unauthorized access [108].

Even the confidentiality and integrity of resources should not be compromised in any way, in cases where services manipulate resources in the cloud, but also in cases when they are stored on servers of third-party public cloud. Therefore, an adequate mechanism for access management and control is essential and plays an important role in the above cases. It is directly related to the necessary characteristics such as authorization, availability, confidentiality, and integrity [104]. Cloud providers must provide essential features for controlling unauthorized accesses, such as ensuring secure access to the services, defending access to service's resources from other services, and controlling access to services according to their previously determined privileges, i.e. to maintain and manage the rules of access control constructively [61]. In cloud environments, CSPs are accountable for identity and other types of management. Nevertheless, because of the weaknesses in identity management systems, a

significant number of resource leakage problems are caused. The absence of an effective mechanism affects in many issues in the cloud computing environment, including identity management, service security, privacy, and resource leakage [38].

Except for an effective authentication and authorization mechanism that should be considered, cloud auditing also plays an important role in the authentication and access efforts of the service. The auditing mechanism should be kept track of all successful and failed activities related to the authentication and authorization of access from service to service and resources [38], [52], [61]. The audit is a crucial aspect for ensuring cloud computing and access control mechanisms is utilized on it. In the cloud access control mechanisms, the audit must track the actual state of a mechanism, record each failure and make a decision, i.e. it should report any effort about the violation of access policies or modification of privileges [108].

4.1. Cloud computing security issues

Cloud computing includes security issues related to Internet-based distributed services such as security technology of visualization, service availability, traffic handling, service security, access management and control, authentication, and authorization [52], [105], [61]. The security of cloud computing is based on a variety of infrastructures, applications, and security policies that are used for service security in the cloud environment. There are used different technologies, including intrusion detection systems, firewalls, and segregation of obligations on various cloud service models and layers provide security in cloud computing [62]. In the security of cloud services has contributed the mutual authentication and security standards of web services [38].

The architectural design, attack areas, preventives from different attack ways, and access controls are involved in cloud security [52], [61]. The services based on the cloud are accessible from the cloud users, but their security can be influenced by the APIs and protocols utilized, and they can also generate cloud computing insecurity [52]. The service provider must care that cloud services, resources, and infrastructure are secured in the cloud environment [104].

There are numerous security problems that jeopardize resources and services in the process of service access and resource storage in the cloud environment. One such issue is the case of resource

storage with the support of third-party organizations which can sometimes have a compromising role as a malicious attacker. The best practices enable the cloud service providers to overcome these security issues and to secure their network by updating it with the package of security requirements such as managing users, roles, and identities; ensuring the appropriate protection of services and resources, enforcing policies for ensure of services and resources; review the security facilities for cloud services; assess security mechanisms on cloud environment and provisions; etc. One of the best practices for the estimation of cloud services is identity and access management (IAM) [38]. Currently, it enables efficient security and identity management and access control to cloud resources and services for registered subjects in cloud systems. Consumers, software processes, or systems can be considered as subjects (entities) [38], [52], [61].

In addition to providing security which is essential for the cloud environment, IAM systems enable various security operations in cloud computing including authentication and authorization. The identity of each entity must be verified first, i.e. passing the authentication process that is followed by the authorization process, for having the appropriate access level to resources [38], [52], [61]. The IAM system ensures the security of the identities and attributes of users in cloud computing by enabling the right users to access the cloud systems. Another feature of the IAM system is access management rights. It can control if the assigned user/service with the right privileges accesses resources that are found or stored in the cloud. In order to provide better service security, access control, and management for resources stored in the cloud environment, many organizations actually are using IAM systems [38].

In the following, in each subsection of this section, a security property is included.

4.1.1. Security polices

Preventive measures through appropriate security policy rules like identification and authentication, logical service access control, protection services, secure service management, handling cloud security incidents, service access control and resource use, and protection of cloud systems against attacks are included in security policies [50], [109]. The relevant security policy rules should enable a secure operating environment, without affecting the performance and reliability of the cloud [50], [107]. Security policies operate and derive according to regulatory authorities. This encompasses

various aspects such as service-level agreements, management challenges between clients and services, and the establishment of trust based on prior conditions [50].

4.1.2. Identity and access management

Identity and access management (IAM) stands for security-related mechanisms that enable resource security and maintain the cloud service identity. It has the possibility to perform many functions such as identity management, maintenance, policy enforcement, resource exchange, authentication, and authorization. IAM verifies whether the correct identities are used for certain applications (services). It manages them and at the same time provides security for the respective identities. Through the IAM is enabled authentication of users, services, or other system resources. It allows or denies access right to resources. In the case of access to any application, the service does not request that its identity be stored or authenticated by the authentication mechanism. The identity verification as a process can be passed to a trusted identity provider in order to be reduced the service/application load. IAM facilitates identity and access management in distributed services [38], [62].

IAM can be used in organizations, enterprises, private enterprises, and cloud providers. It can be used in cloud computing to identify cloud objects, and entities, and control the access of services to resources based on predefined policies. There are numerous operational areas and multiple of them relate to IAM. These operational areas related to IAM include identity management and facilitating, authentication and authorization management, federated IAM, and compliance management. The aforementioned operational areas enable that the authorized services to be integrated securely and constructively into cloud computing. Authentication management through IAM enables usernames, passwords and digital certificates in the cloud to be managed properly and securely. It also provides the possibility of authentication of cloud services by utilizing identity providers. Once the authentication is successful, the authorization mechanism is considered, which determines whether the authenticated entity can carry out any operation within a certain service [38].

In the context of cloud security issues, the IAM has a major role. Two important issues in terms of security are privacy and interoperability as the existing IAM approaches, mainly in the context of the public cloud. IAM systems are currently effective mechanisms for minimizing risks related to the

cloud environment. The IAM system is provided by many organizations to protect resources by controlling and managing service access permissions. SailPoint, IBM, Oracle, RSA, and Core Security are the most widely recognized IAM system providers.

Four main solutions for cloud security are offered by Oracle IAM. The initial offering from Oracle IAM involves identity management, which encompasses self-service account requests, management of identity lifecycle, enterprise role management, and password management [38]. The management of authentication and trust of services is offered as the second solution by Oracle IAM by including specifications such as privacy, SSO, and identity federation. A third solution offered by Oracle IAM is an access control including features such as fine-grained rights/privileges, authorization based on risk, and security of web services. The identity and access administration is offered as a four solution by Oracle IAM by including features like tasks segregation, auditing, management of resolving conflicts, role mining and applying, certification, resource analysis for identity and embezzlement-prevention and directory services (virtualization of identity, database security for services, consistent storage, etc.) [38], [50].

4.2. Cloud computing authentication mechanisms

The mechanism that is used to verify the identity of the user/service is called authentication. In the cloud system, authentication means that the right user is getting access to the resources provided by the cloud provider [62]. Authentication is considered the mechanism, which enables one entity to approve another entity [59]. Its purpose is to ensure that the proper person or application is accessing certain resources [52]. Authentication as a process is performed through the software or as a part of it [38].

Cloud computing authentication is ensured if the resources stored in the cloud are accessed; in this situation, the identity of the user is offered to the cloud service provider [62]. The cloud provider is able to choose and provide different authentication mechanisms that have various strengths of security. Reliability and integrity determine the strength of these mechanisms [107].

Simple authentication is not considered an appropriate solution for customers who access and compose services from multiple cloud providers in the cloud environment. Such a problem is solved if

we use different authentication methods [50]. In Figure 4.2 is presented the general authentication scheme in cloud systems. Users using the services in the cloud server must be verified by the authentication server and then access the cloud server. Its strength has an impact on the cloud environment in terms of reliability and security [62]. Access permissions are granted to users when they introduce themselves with something that is convenient for the system, such as a card number or password, which is dedicated to users and defined by them [50].

In the following are presented and discussed several digital security authentication mechanisms [50], [38] that are very important and enable service-oriented cloud computing authentication.

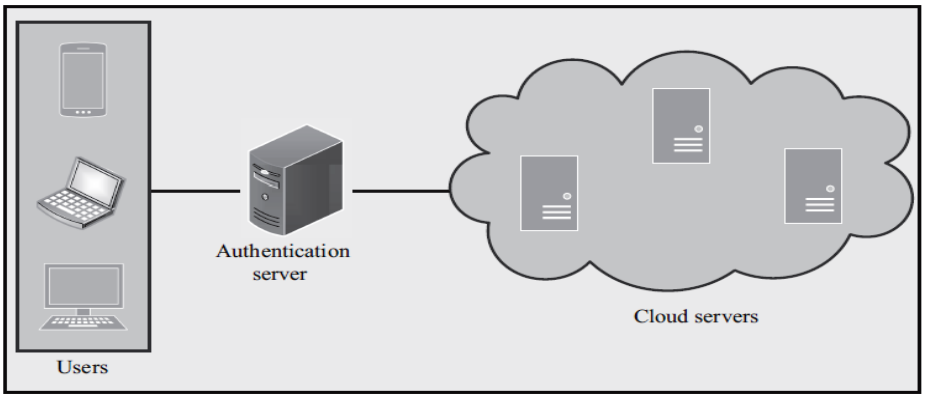

Figure 4.2. The general authentication scheme in cloud systems [62]

4.2.1. Password authentication

Password authentication is considered simple and not complex to use. It must have a complex composition and be regularly renovated to keep the possible degree of security. This authentication technology is known for its weaknesses, so even if the username and password are provided correctly, it is still hard to verify whether the owner of the given credentials is the rightfulness and the resource request is sent by the right owner. It often happens that users reuse passwords when they are authenticated in different cloud services. Usually, the high-security risks in user account information come mainly from fragile password practices. However, password authentication is used in more than 90% of transactions [107] and makes this widely used as an authentication technology. In the latest

cloud deployment, password authentication is the simple model of the challenge-response authentication protocol [110], [107].

The challenge-response authentication protocol is a set of protocols where one pair presents a "challenge" (to be responded to) and another pair must provide a valid "response" (the answer needs to be validated (checked)) for the question (challenge) in order to be authenticated. These protocols request of the claimant prove his identity to the verifier by presenting its information as a secret value, which only the claimant knows and they are not disclosed during the authentication process, as part of the authentication protocol [110], [111].

4.2.2. Public key infrastructure authentication

Public key infrastructure (PKI) authentication is an authentication method based on public key cryptography. This authentication mechanism allows users to authenticate other parties through a certificate without distributing their secret information [62]. The right level of trust in the cloud is achieved when a Trusted Third Party (TTP) is used, which provides the solution for maintaining confidentiality, integrity, resource authenticity, and communication.

Joining two authentication methods, respectively PKI and TTP, and their application in the cloud results in strong and effective authentication and authorization in that system [107]. In order to provide proper authentication, PKI is used to develop and design Secure Socket Layer (SSL), Transport Layer Security (TLS), and Secure Electronic Transaction (SET). SSL, TLS, and SET are security protocols [62], [110]. PKI's effectiveness lies in managing private key access, identical to other forms of encryption systems [62].

4.2.3. SSO and cloud federation authentication

Since the applications of SaaS need a centralized management system, which restricts the software policies, the traditional mechanisms of authentication are not always considered appropriate for remote authentication. Clients can use many services in the cloud, therefore, they cause many requests for logging and these bring different problems. This happens as a result that a single consumer should maintain a large amount of credentials. Based on the above-mentioned reasons, the SSO mechanism is considered the potential solution for such an issue [101].

Federated identity is really a valuable functionality for managing identity. The fundamental principles and protocols for federated authentication of the cloud service are considered OpenID, OAuth, and SAML [62]. SAML, OAuth, and OpenID offer SSO facilities by enabling the Identity Provider (IdP) to exchange information of authorization and authentication with the Service Providers (SPs). This workflow interaction is shown in Figure 4.3 [38].

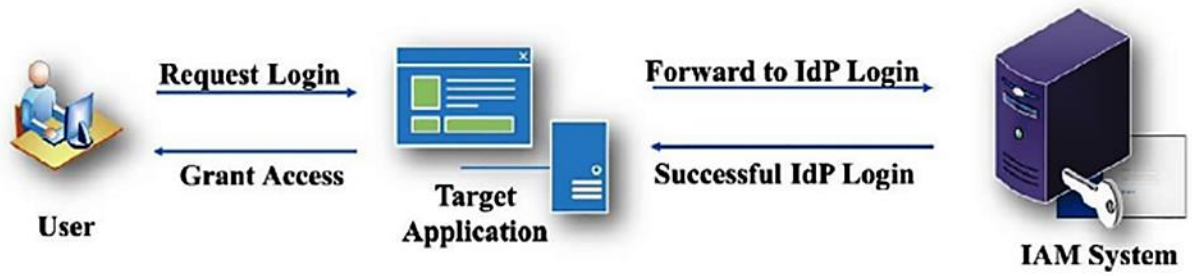

Figure 4.3. The mechanism of the SSO authentication workflow in the cloud environment [38]

4.2.3.1. SSO in cloud services

SSO is a system (a mechanism) for identity management in which a user can be authenticated through a single authentication. Then, the user can access to the specific resources by not repeating authentication (without logging in to an application (service) again). SSO [105] is considered something like a passport that is used for authentication only the first time. The user does not need to log in again, for other sites or a process in which is applied the authentication mechanism. Multiple clients are defined by this protocol. Such clients can access resources and applications (services). When a consumer (user) uses the SSO [112], it usually has the proof through which verifies its identity and access. Afterward, it uses that to access certain services [104].

Figure 4.4 shows the SSO architecture workflow in the cloud system (environment) context. Different expressions are used in systems where SSO protocols are included. All expressions are discussed in the following. The identity provider (IdP) has responsibility for the process of authentication. In addition, the user information stored as attributes in a token of identity is handled by the IdP. Once a user is authenticated, an object that holds the user information is created by IdP. When

the service provider requires user attributes, it uses the user information. Service provider (SP) mainly provides the specific service and it is secured from securing protection in general. If the user decides to use the service, the security protection requires the identification of its information. The authorization server is also considered with the term of the trusted authority. After the user is authenticated, the authorization server provides access tokens for it. There are various SSO structures, but the enterprise single sign-on and web single sign-on consider the most used structures of the SSO [62].

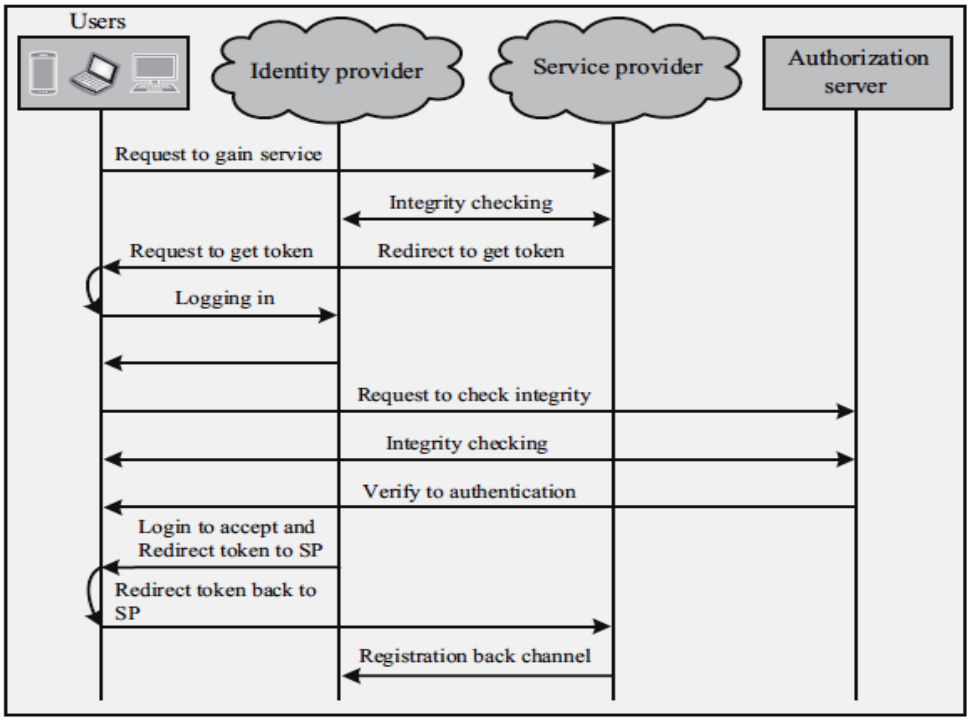

Figure 4.4. The SSO architecture workflow in the cloud environment context [62]

4.2.3.2. OpenID in cloud services

An authentication protocol that is also an open standard is called OpenID. It enables the authentication of users to the relying parties (RP) through the support of third-party organizations. A relying party (RP) refers to a resource provider, such as a website (application), that necessitates the verification of end-users [38]. OpenID [112] is based on SSO services' features, therefore this enables

the authentication to be performed using the credentials only once to authenticate and access multiple websites and web services [38].

It is a decentralized authentication system and its mechanism is based on this type of authentication. OpenID does not need administration by webmasters. The users' list is stored in the OpenID provider. Then users create their accounts with the support of the provider's list. The cloud users through their accounts can sign in (access) any website that enables OpenID authentication. OpenID Connect (OIDC) is the latest version of the OpenID mechanism. It is built as the identity layer based on the OAuth protocol. The authentication mechanisms for cloud and mobile applications are supported by the OpenID Connect protocol. In the case of communications between the cloud interaction parties, OpenID Connect enables their encryption and sign-in [104].

4.2.3.3. OAuth in cloud services

OAuth is an authentication mechanism used in cloud computing and it enables one-way or mutual authentication in the cloud environment. This alternative method of authentication is also an open standard that allows delegating and granting access to the user on another application/service without distributing the password by the authenticated service/application. OAuth 2.0[2] is the latest version of OAuth [38].

Using the OAuth authentication standard undoubtedly provides favorable authentication options, as in the case where users can exchange private resources which are usually stored on a secured resource server. In this case, the user's credentials are not exchanged. The OAuth's goal is to complement OpenID and delegate access for users/services to the protected resources. This part is enabled through the authorization server, which usually generates tokens that do not contain information about user credentials. The simplicity of OAuth [106] has influenced it to be the preferred solution for cloud providers [62].

OAuth offers a method of accessing the HTTP service from the application/service in the name of the resource owner that is commonly used as an authentication standard by Internet giant companies such as Google, Facebook, and Microsoft. The mechanism of OAuth is functioning on HTTP which

[2] OAuth 2.0 is available at: https://tools.ietf.org/html/rfc6749#page-4

assumes a cloud server, which has a role in adapting the OAuth mechanism that can mainly operate by utilizing HTTP [113]. There are four various types in OAuth 2.0 framework, which enable authorization grants. The first type is the authorization code which is considered the most widely used type. In this type, the communication is performed between the cloud and authorization server during which is generated the access token. The implicit type is the second type of OAuth 2.0 framework in which the consumer accesses the authorization server to get the token for accessing. The third type is resource owner password credentials. It is an authorization grant type in which the consumer provides the consumer ID and password to the cloud. Client credential is the last type that supposes the authorization server need to have trust in the cloud, so the authorization server authorizes all controls of the authorization to the cloud [63].

OAuth as an authorization framework enables third parties to access user/service resources without revealing the username and password to the third-party service. For example, the user uses HP's SnapFish service to print photos online, and he/she may authorize this service to access his/her Facebook account images without granting his/her SnapFish his/her Facebook account password. OAuth can execute the restrictive policies in the access domain and token expiration for restricting clients/services' access to specific resources and functions for a certain time period. The need to integrate APIs and cloud services and their prevalence in the cloud environment has made it necessary to use a common protocol for delegating authorization. This request is best fulfilled by OAuth [64].

OAuth is being utilized as a security layer and a standard protocol in various technology services including cloud computing. For example, we have a cloud image-storage service and an image printing service and we want to print the images (photos) that we have stored in the cloud storage service. An API is used for communication between the cloud printing service and the cloud storage service. The two services mentioned above operate in various companies, so our storage service account has no connection with our printing service account. In this case, OAuth is considered as a solution to this problem by enabling us to delegate access to our photos to various services, without providing our password to the photo printer. The OAuth system consists of four major actors such as the client, the authorization server, and the protected resource. Each of the four OAuth actors is accountable for various parts of this protocol. All components work together to enable the proper operation of the OAuth protocol. In this example, the OAuth client is considered the printing service,

while the photo storage site represents the protected resource. The end-user is identified as the resource owner, who wants to print its photos. For its protected resources, the photo storage site runs using its in-house authorization server [114].

4.2.3.4. SAML in cloud services

SAML [105] is an open standard mechanism that enables communication and exchange of authentication and authorization resources between two interacting parties. Its mechanism operates according to the request and response techniques based on the token. The service provider and identity provider are two interaction parties. SAML assures that user authentication with the service provider is performed securely. No user credentials are included in the tokens of the SAML. It also enables data communication to be encrypted and encoded between two interaction parties (the identity provider and the service provider) [38].

SAML takes care of and enables secure authentication of the user in the service provider. SAML supports the SSO specifications and it enables interoperability based on this mechanism. It includes several roles such as user, identity provider, and service provider. The user requests a web service from the service provider, which requests and receives assertions of authentication from the identity provider. The service provider decides about the access privileges/permissions based on the received assertion [104].

4.3. Cloud computing authorization mechanisms

Authorization is considered a systematized mechanism of the cloud system that determines the access level of authenticated entities in secured resources [52], [62]. Authorization presents a method that permits or prevents access to a specific resource based on the rights (permissions and privileges) of the authorized entity. There is a system administrator who monitors the access permissions in systems where entities (users/services) have permission to access. Since the cloud network is made up of various service providers, there are situations where the user can access different types of services at a specific time; each service that is provided by the specific service provider may have different levels of security [38].

There are cases when the authorization rights are managed (granted) by third-party organizations, which are authorized to have access to specific private information about services or applications as shown in Figure 4.6 [38]. For example, if the application (service) is authorized by the user, then the application (service) hosted in the cloud can be accessed outside of it. In this case, the authorization is performed through the delegation of access privileges or access control policies. The cloud service provider provides and applies (implements) access control policies for services and resources where their access is only available to the authorized users/services [38], [50]. The taxonomy of the authorization mechanisms is shown in Figure 4.5.

The protection of confidential information, minimization of management, and security tasks are several benefits of centralized authorization mechanisms. Nevertheless, there are authorization mechanisms like MAC, DAC, RBAC, and ABAC, and they are shown in Figure 4.6 [50], [38]. These mechanisms are discussed in the subsections of this section.

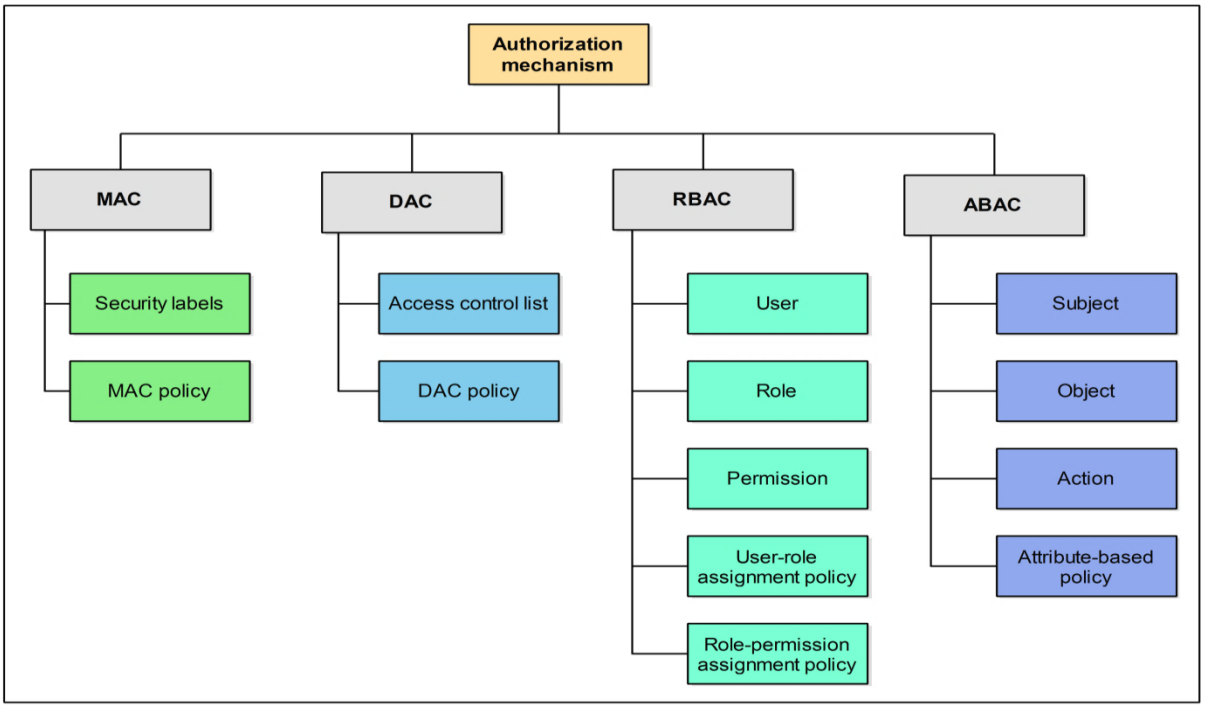

Figure 4.5. The taxonomy of the authorization mechanisms

4.3.1. Mandatory access control in cloud

Mandatory Access Control (MAC) [107], [58] is the conventional mechanism for determining users' access privileges (rights) as is shown in Figure 4.6. The permission for access is granted by the MAC via the operating system or security kernel. The MAC controls the capability of data owners for permitting or rejecting access rights to the customer for a file system [60]. In this mechanism are decided the access control rights by the system manager, while the operating system or security kernel enforces them. In MAC, the objects of the file system are classified according to sensitivity labels as secret, top secret, or confidential. Each device or client is also classified according to the above-mentioned levels [59].

The labeling process according to the classification of customers and resources is specified by the security kernel. The operating system or security kernel is responsible for controlling the username and password for persons or systems when they access specific resources. As we mentioned, they are also responsible for specifying the access rights for parties (persons or devices) who access or attempt to access. Although the MAC provides a lot of security in resource access, the right planning and frequent monitoring are key factors that must be considered to hold classification labels updated. The processing of the access rights is not appropriate in MAC because it has a not-so-flexible environment for this issue [38], [50], [61].

The MAC must have a central authority to define what resources will be accessible and who can access resources. For example, a company manager wants to access the resources of a company staff member. Full access to all staff member resources should not be allowed to the manager, because if s/he has full access, s/he can access and disclose sensitive resources such as the details of the bank accounts of the company staff members. While cloud computing uses web applications to provide its services, MAC needs to sophisticate semantic models for cloud security because there is a lack of them, especially in representing and communicating the privileges and restrictions that are enabled through access control policies [108].

4.3.2. Discretionary access control in cloud

Discretionary Access Control (DAC) [58] also known as Identity-Based Access Control (IBAC) [59], is a mechanism for security access control that controls the access permissions via the data owner, as is shown in Figure 4.6. In DAC, any user's access rights are completed through authentication by validating the credentials (username and password). DAC is considered discretionary since the owner specifies access rights. File/information or resources possess the owner in DAC, while the resource owner controls the policies of the resource access. Nevertheless, the DAC mechanism offers more flexibility than the MAC mechanism, while it has less security compared with MAC [38], [50], [58].

The DAC can also be used in cloud computing, but it presents side effects, despite the DAC being influenced by authorizing of objects' owners to check permissions of access to objects. For example, if there was any mechanism or technique to enable the administration of inappropriate rights (risk awareness) that object owners could provide to users, it would be an extraordinary element for the DAC. Unfortunately, such a feature does not exist in the DAC. Sometimes, users/services are requested to utilize privileges that disclose resources for entities to third parties. For example, when the employee has to read the contents of a file in the company where it works and then it copies the contents of the file to another file and transfers it to another employee. The DAC does have not the authority to control the flow of information or deal with malicious content that may be carried as a result of access permissions. Besides the above case, the possibility of transferring the rights of the user/service to another is presented as a problem in the DAC and as a consequence, the integrity and confidentiality of the objects can be violated. Based on the above characteristics, it means that the DAC is not scalable enough for use in cloud computing [108].

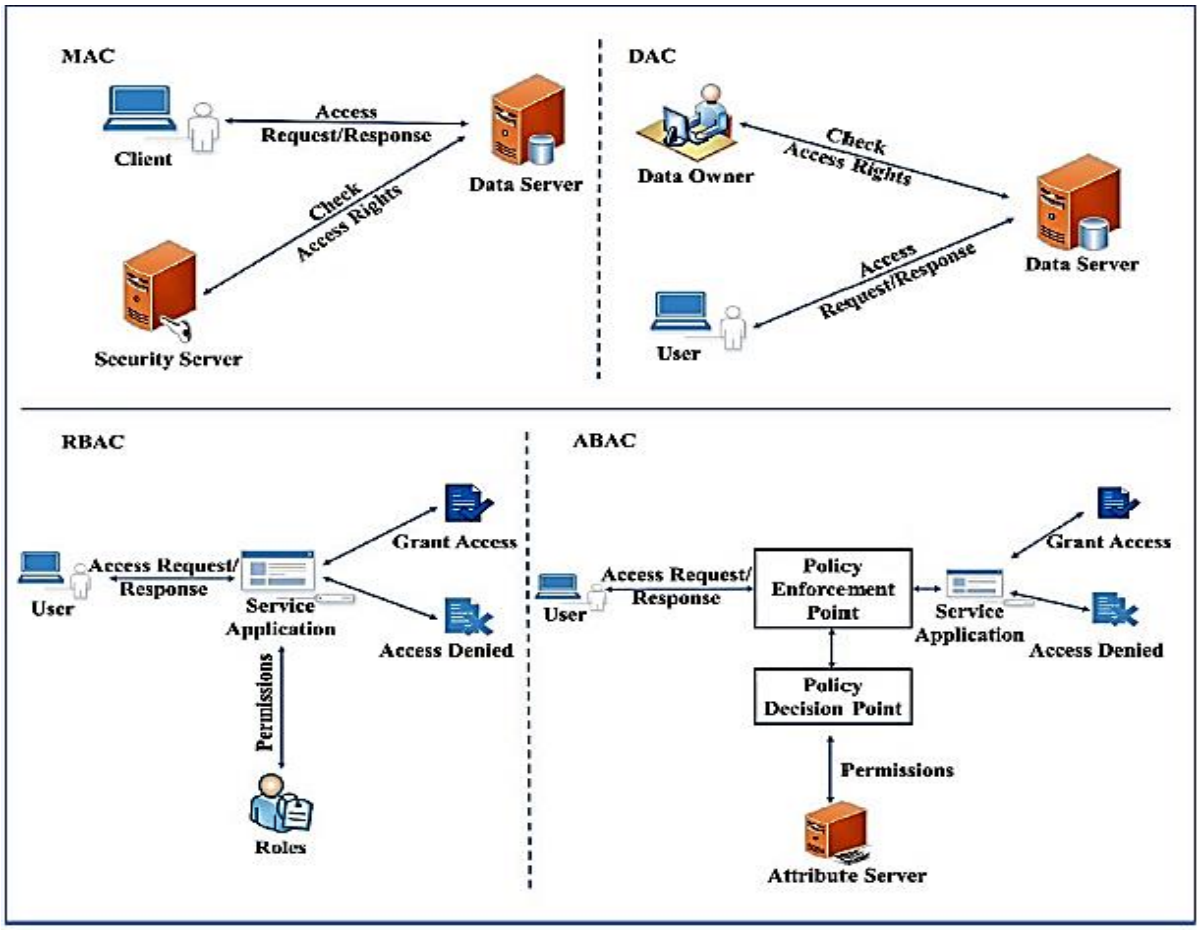

Figure 4.6. Comparison of the authorization mechanisms in cloud computing services [52]

4.3.3. Role-based access control in cloud

This is an access control mechanism briefly referred to as RBAC [58] that allows access rights for users according to roles and permissions (Figure 4.6). The policies of access control present rules that tell how the setup process authorization allows or denies users [59]. The user is defined as a human, process, machine, or network. A role presents the approval for performing an object operation, which can be an action, function, or duty that the user should be done. The object is another term that is included in RBAC, which refers to information containers such as files, directories, database tables, or resources (printers, PC, etc.). The permission is defined as a 'tool' that enables the user to assign

roles that are actually active along with user sessions. Permission as a part of the RBAC model has functions to analyze (review) a set of users, to which they want to assign a role or a set of roles [60].

In RBAC, user permissions are provided by different parameters and they can be as user roles, permissions of roles, and role-role relationships. Roles in RBAC are divided into two categories, namely application/technical roles and organizational/business roles. The application/technical role involves the combination of various application/service-specific rights or tasks that have elements of permissions, specifically based on permissions. Its domain is restricted to specific applications. The organizational/business role consists of various job functions and the rights of access allocated for employees. It consists of combining various application/technical roles. In organizations that have many users and required multiple permissions, RBAC is used to administrate their security [50], [38], [61].

RBAC primarily includes three rules for allocating permission to a specific user, like assignment of role, authorization of role, and authorization of permission. Based on such rules, users are given permission for accessing the resources. RBAC offers a secure environment for setting (allocating) access permissions. The disadvantage of the RBAC is in terms of changing roles, which they change from time to time and there is always a need for real-time environment. Therefore, certain changes need to be checked and verified in that environment [50], [38], [61].

The RBAC mechanism has a lot of advantages in comparison with the DAC and MAC mechanisms. Selecting the proper roles that represent the cloud system is not a simple duty, and categorizing entities based on roles can make things problematic. In the RBAC mechanism, roles categorize (rank) the subjects into a variety of categories. Therefore, each entity (subject) must have a role to enable access to the cloud system. Nevertheless, roles sometimes give the subject more rights than are necessary to it, so this can lead to the abuse and infringement of access security policies. In order to interpret and transmit privileges, the RBAC must deal with an absence of complicated semantic models. For example, a doctor in a remote environment may not be able for accessing a particular system through cloud computing because of the absence of syntactic and semantic assistance. This lack of semantic support also leads in facing to a semantic gap amongst the authentication and authorization mechanisms of the service [108].

It must ensure that access decisions are made within a reasonable period of time and based on system requests before using the RBAC in cloud computing. For instance, the response time is essential for a lot of applications including even the health care system. The system must be accessed in a timely way by a remote consultant from a hospital by ignoring an amount of RBAC and distance access requests. There may be critical infrastructure of service provider who wants to migrate to cloud computing. It can have many users, dozens of roles, and thousands of permissions. Such infrastructure can be faced with enormous tasks which cannot be centralized by small groups of security administrators [108].

The cloud system consists of a sequence of operations that must be controlled. For example, in the health care system, the doctor must examine the physical condition of the patient; check his /her medical history, and asks the patient for tests or scans around his/her in order to achieve the proper patient treatment. The doctor can request another doctor for support or send several data to another hospital. Each of the above operations (actions) requires various sets of permissions. Therefore, the RBAC may not be able to provide access to a series of operations in cloud computing [108].

4.3.4. Attribute-based access control in cloud

Attribute-Based Access Control (ABAC) [58] is considered a mechanism that is needed to control access permissions as shown in Figure 4.6. This access control mechanism uses its policies to define various sets of attributes that are needed to control the user access rights separately [59] as is shown in Figure 4.6. The policies are mainly created through various types of attributes. The system is based on these policies and decides the access permissions. Here includes a series of attributes which are subject attributes, object attributes, resource attributes, and environmental attributes. Under the ABAC mechanism, each user's roles and permissions/privileges are predetermined. This model enables the solution of many issues of authorization, ensures successful regulatory enforcement, and enables flexibility in implementation [50], [38], [115], [116].

The starting date of the user's work, his location, the user role, etc. can be attributes. The relation between the attributes may or may not exist. Attributes are first determined to be used in the system, and then each attribute is accounted as a discrete value. These values are then compared to the set of values from the policy decision level to allow or deny access. In addition, a subject does not

need to be identified preliminary to the system. It must once be authenticated in the system and then ensure its attributes. Nevertheless, an agreement must be reached to know what type of attributes should be utilized and how many of them are taken into consideration to create access decisions. Such an issue is considered quite complicated in cloud computing [108].

It is necessary to propose a security policy, which can work exactly with the ABAC model in cloud computing, due the security policy is accountable for picking the substantial attributes that are used to create access decisions.

4.3.5. The authorization models

Authorization is mainly needed to control the access request for resources. It is performed as a process after authentication. Authorization enables the complex access controls that are derived from resources and policies such as user/service attributes, user/service roles, actions taken, access channels, requests for resources, external resources, business logic, and rules [62].

There are three authorization models shown in Figure 4.7. They are discussed below:

- *User/service push model.* Initially, the user/service performs a handshake with the authorization authority, then with the resource site/provider along a specific sequence.
- *Resource-pulling model***.** Under this model, the resource is located in the middle. In the first step, the user/service initially controls the resource. In the second step, the resource request is verified in its authority, after the resource contacts the authority for this issue. Then, in the third step, the authority authorizes the resource. After this, the resource approves or refuses the request of the user/service at the final step.
- *Agent-based authorization model.* In this model, the authorization authority is put in the middle. In the first step, the user/service controls the authority, while the authorization authority decides about access to the required resources. In the last step, the authorization process is finally considered completed [104].

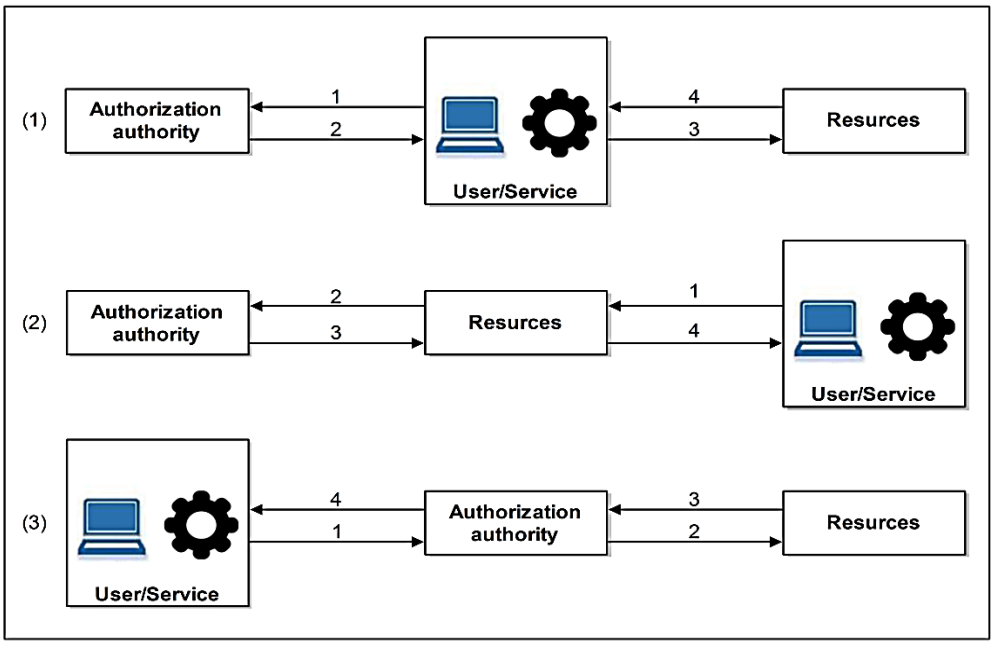

Figure 4.7. Three authorization models: (1) user/service-push model; (2) resource-pulling model and (3) agent-based authorization model

4.4. Cloud auditing mechanisms

Auditing is called the process of gathering information about the user/service for its access to specific resources or services or about actions performed by the user/service. All activities made on a certain resource/service must be recorded in the log file on the system. If any issue occurs, the log file can be checked to find such a user/service. The application of security auditing enhances the trust of cloud tenants in service providers by providing security in accordance with applicable laws, regulations, policies, and standards [52], [117].

Systematic auditing should be applied throughout the life cycle of the IT outsourcing process to control and identify the risks in resources and services. Cloud auditing is more complicated than regular IT auditing, therefore in cloud computing external organizations or partners are involved in helping in the auditing process [118].

Auditing in cloud computing can be internal or external. Internal auditing is carried out by internal auditors to analyze and evaluate the resources (data) and processes for improving the effectiveness and efficiency of the organization. External auditing is performed by auditing companies or specialist auditors. Auditing is the necessary element that organizations must apply and respect for demonstrating that they comply with applicable audit regulations, including processes, practices, internal controls, independent validation, or quality assurance related to relevant certifications [118].

Cloud auditing is considered an extraordinary function of assessing and identifying risks associated with the organization's resources and services. It is the main tool for helping the board and management of the organization in assessing, identifying, and reporting risks related to the organization's resources and services. Cloud-based auditing can be applied to different sectors like levels of entity, application systems, security, information systems, data centers, virtualized environments and web applications/services, and enterprise resource planning. The cloud auditing process is based on the International Standards of Auditing and Auditing Methodology. Its realization can be done by internal or external auditors. Auditing is a continuous process and it is performed in different stages and such is important and should be considered by the auditor [118].

4.4.1. Third-party cloud auditing mechanism

Since different users can use resources (data) at any time, the consistency of resources (data) is more crucial because unauthorized users can use, modify, alter, or remove them. When multiple users are simultaneously interacting with data, with one user writing and another user reading, it can potentially result in inconsistency. Therefore, it becomes crucial for the data owner to address and resolve this data inconsistency as a significant task. Third-party auditing (TPA) can be used as an intermediary party between the customer and the cloud service provider. TPA uses a resource (data) and can modify it as the data (resource) owner. It also receives information about the problems presented many times by the user, which is considered an important issue. Improving security control and data integrity enables the detection of problems that are security issues and third-party auditing services. Mostly, their management is enabled through third-party auditing [119].

TPA provides security and some main security elements [119]. TPA is an individual who has the knowledge and skills to perform all audit procedures are shown in Figure 4.8. The TPA scheme shown

in Figure 4.8 is used to check the security and integrity of data (resources). Since there can be many incidents and suspicious movements, cloud users depend on third-party auditors. Balusamy et al. [120] have proposed a framework that includes the data (resource) owner, which controls the integrity and security of the outsourced data (resource(s)) [99].

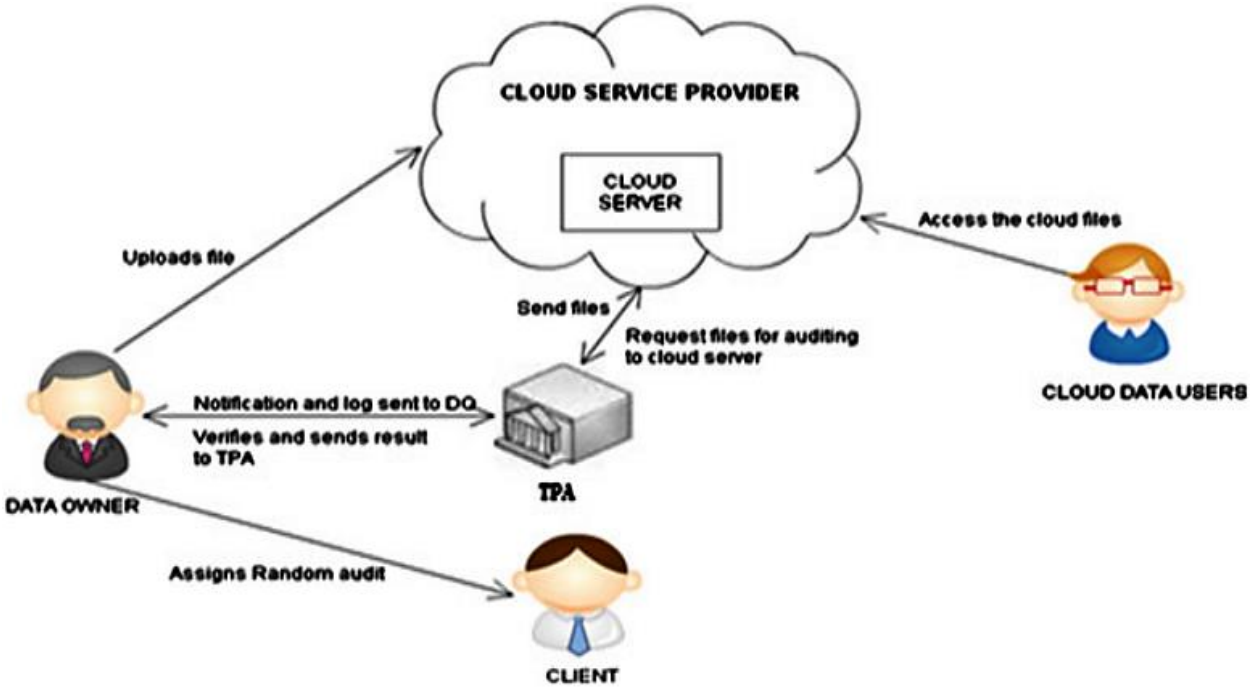

Figure 4.8. Third-party cloud auditing mechanism architecture [99], [120]

The proposed scheme enables data (resource) integrity and security, and it ensures the data (resource) owner (DO) in relation to data (resource) security. The owner possesses information about all its resources in the cloud. Thus, this framework ensures data integrity for all cloud owner resources. This scheme also includes the DO throughout the audit process. Initially, the TPA applies normal auditing processes. Upon discovering any data (resource) changes, then the owner is notified of those modifications. In the next step, the owner checks the log files of the auditing process to verify the relevant changes. If unusual actions occur in the owner's data, it can check them itself or delegate control to another auditor that it chooses. In this way, the owner is notified (followed) of any modification that occurs in his data (resources). There's also an appointed threshold value, which

shouldn't be exceeded as the response by a third-party auditor. All modifications less than or equal to this threshold are validated by the DO. The DO is supposed to perform a sudden control when the used time exceeds the threshold [99], [120]. The above-mentioned cloud auditing process is shown in Figure 4.9.

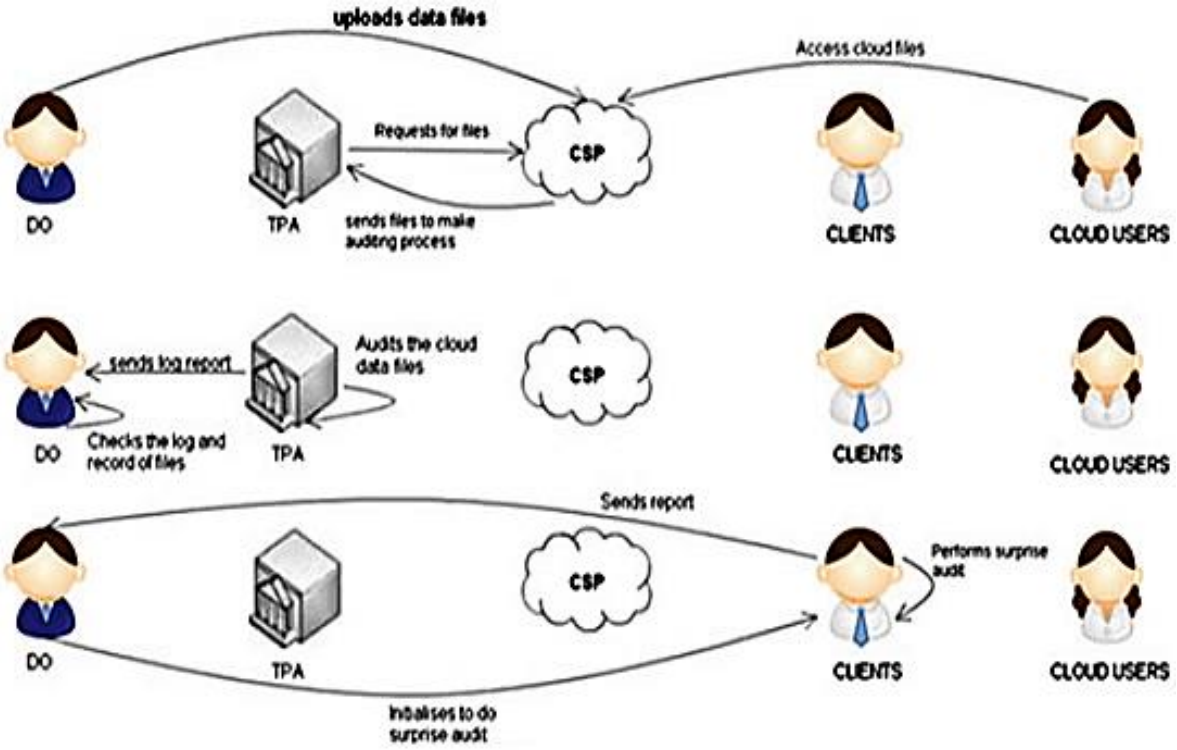

Figure 4.9. The comprehensive system architecture of third-party cloud auditing mechanism [99], [120]

4.4.2. The public cloud auditing mechanism

The system architecture of the public cloud auditing mechanism where intervenes the user revocation is shown in Figure 4.10. In the public auditing mechanism are included three entities: cloud, third-party auditor (TPA), and users. The cloud provides users much more important elements such as data (resource) storage and distributing services. The integrity checking of distributed data (resources) through public auditing for different users is enabled by the TPA. In a separate group, there is an original user and other users of the group. The original user is the data (resource) owner (DO). It can create or distribute data with other users in the group using the cloud. The original user and group

users can access; download and change the shared data (resource(s)). The shared data (resource) is then separated into blocks. The changing of the shared data block can be done by the user using one of the operations: insert, delete, or update that data block [121].

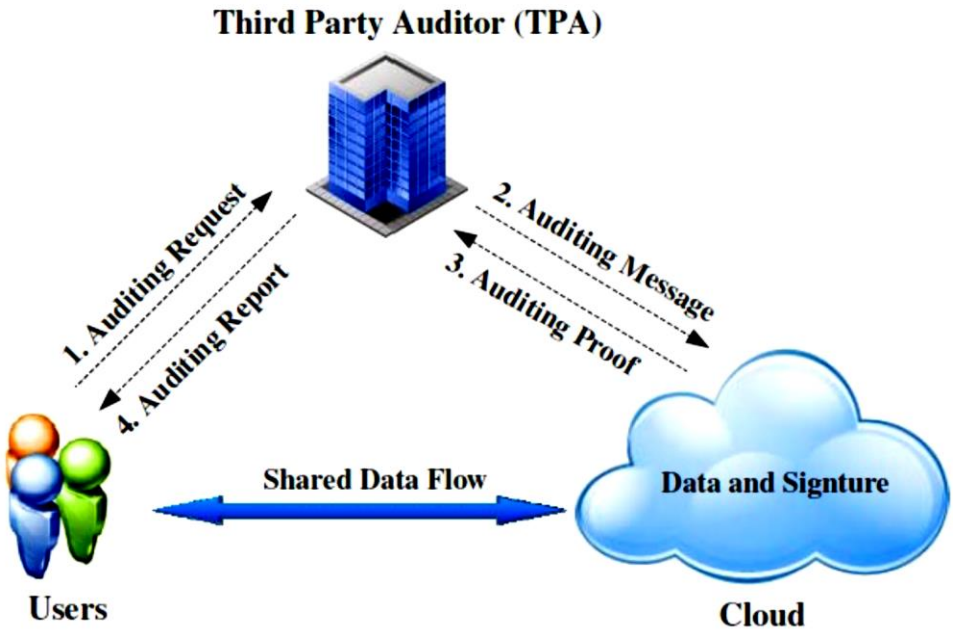

Figure 4.10. The public cloud auditing mechanism system architecture [121]

Another very efficient model of the cloud auditing mechanism (as in Figure 4.11) is introduced by Imad El Ghoubach et al. [122]. Four actors constitute this system model as below:

- *Cloud Service Provider (CSP)* provides the necessary space for data storage and computing power for the limited resources of users. It can delete or modify the stored data. CSP tries to generate a valid message of verification for providing the TPA and users that it stores with loyalty their data on its servers.
- *Data user* outsources its data storage to the cloud server, while the maintenance relies on the CSP. An individual or organization can be considered as the user.
- *The Third-Party Auditor (TPA)* is an integral part of the system model which can verify the integrity of the user data (resources) according to the submitted request. TPA is the correct and necessary entity for the system architecture that follows the given model. However, it

tries to recover the data from the responses received, if such actions are necessary for the system. These actions are performed during the verification process where TPA faithfully pursues and run the algorithm of this process.

- *Private Key Generator (PKG)* is an entity trusted by CSP, TPA, and users. The master key is generated by this entity, and then it is used to generate private and public keys for each user [122].

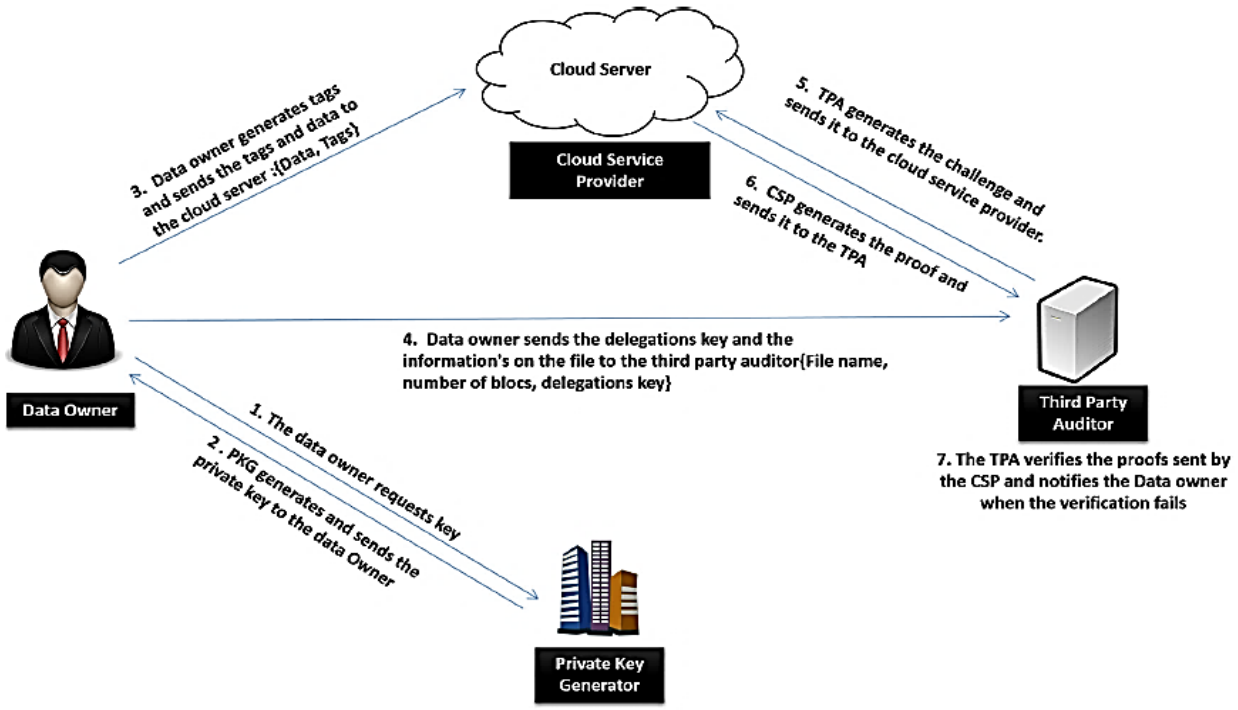

Figure 4.11. The secure public remote cloud auditing mechanism [122]

4.5. GDPR in cloud services

In May 2018, the European Union (EU) approved a new law on data protection, which has an effect in all EU countries and is called the General Data Protection Regulation (GDPR) [123]. This regulation has a great impact on the use and management of technologies related to data protection in the EU, including cloud computing, where the operability of this regulation is detailed in this section. The GDPR has aimed to clarify the concepts and procedures related to data protection in the digital

world that enormously intensify the risks of data breaches. Since many European companies use cloud applications, which encountered 608 such applications, their compliance with GDPR rules is a difficult task [124], [125], [126].

Cloud providers must comply with GDPR rules and compliance with the relevant rules is required from May 2018 for all cloud providers operating in the EU. The purpose of the GDPR is to enable a high and very stable level of personal data protection for all EU countries. With this approach, the GDPR enables EU citizens to control their personal data and simplify the environment that affects personal data in business terms. The regulation as a whole is applied to all areas where personal data is affected, including companies that store or process data about residents within the EU, respectively cloud providers, cloud consumers, and cloud sub-contractors. The rules for the protection of personal data included in the GDPR must be executed consistently in all EU countries. Particular emphasis is placed on how identifiable personal data are handled and protected by mechanisms within the EU and in specific cases outside the EU [125], [127].

Throughout the GDPR compliance process in cloud computing environments, some specifications of the regulation should be accounted for. In the cloud environment, GDPR is executed for cloud "controllers" who decide why and how to process personal data and cloud "processors" who process personal data on the name of the controller. Specifically, the cloud controllers and processors who are operating within the EU must comply with GDPR requirements. The clouds that have originated outside the EU, but provide services to the EU citizens must comply with the GDPR requirements. They should be constantly monitored by the EU institutions for the implementation of the GDPR. Therefore, for checking if each cloud meets the GDPR requirements, there is suggested to apply an audit warning mechanism that will continuously check if the user, service, or data placed (processed) in the cloud (IaaS) is in compliance with the GDPR regulation [127].

The GDPR principles such as lawfulness, righteousness and transparency, preciseness, data minimization, storage restriction, integrity, and confidentiality must be ensured by cloud providers. In cases where the requirements are specific for certain services, the data controller should adopt the privacy policies that are most appropriate to the data, and thus it is enabled the promotion of privacy rights. Data that is collected, stored, and processed by cloud providers should be used for specific and

legal purposes, while their unauthorized use outside of certain purposes is prohibited. In the GDPR is clearly specified that certain data may be further processed only in cases when they are used for public purposes or scientific research. Personal data must be kept reliable and up to date by cloud controllers. If certain data is used for specific purposes and should not be used further, then they should be deleted immediately and thus the principle of storage limitation is respected. Finally, in order to prevent unauthorized or illegal processing or/and improper disclosure, destruction, or harm, integrity, and confidentiality must be ensured. Data encryption, anonymization, and fragmentation techniques should be used to provide integrity and confidentiality. If reasonably possible, pseudo-anonymization as a privacy-boosting technique should also be applied by eliminating immediate data linkability to the data subjects [127].

Cloud providers must ensure information through requests they receive from data subjects or their privacy policies based on the following elements: identity and contact information related to the controller; the data included, reason (objective) of processing and legal basis, recipient or rank of the recipients; information about data transfer outside EU territory, data retention period and rights of entities (individuals) [125], [127].

Data subjects must receive confirmation from the cloud provider if the personal data of the particular individual is being processed; does s/he has access to additional data and information regarding their correction or provision, data source, and portability?! Cloud providers are obligated to take care of the owners' data by giving them the right to oppose the processing of their data whenever it is reasonable. In technical terms, this issue can be regulated through the application of the mechanism that enables opposition to the data subject automatically and other similar actions [127].

In case the data is placed in the public domain of the cloud provider, it needs to inform all other controllers that the data owner wants to restrict access or that his/her data must be removed. Cloud providers need to install special deletion software that ensures that data cannot be restored (recovered) from cloud storage. If the data must keep for certain periods of time, it is necessary to use restrictive mechanisms that will not allow the information to be available every time and to enable the blocking of data for other systems [127].

The data processors and controllers are the two main roles of cloud participants, which are specified as terms/roles in the GDPR. Cloud providers operate as data processors on behalf of clients/users that are considered the data controllers [125], [127]. A data processor refers to an individual or organization, including both public and private entities, that handles personal data under the authority and instructions of the data controller. The term "controller" refers to an individual or organization, including public authorities and agencies, that is responsible for deciding the purposes and methods of processing personal data. In cases where the processing purposes and methods are determined by the laws of the European Union or Member States, the controller may be designated by either Union or Member State law, or specific criteria for such designation may be outlined in the respective laws [127].

The PaaS cloud provider must have complied with the GDPR. In relation to organizational security obligations and GDPR guidelines and procedures, PaaS allows cloud users to secure and protect their data, as well as personal data. It is highly recommended to utilize some security measures of service level to assure the integrity, confidentiality, and availability of the data processed. Relevant measures should be included at multilayer levels, such as physical, logical, and data, and should be incorporated elements such as 24-hour access restriction to the data centers, authentication as an integral part of processes (via the smart cards or biometric scanners) to enable physical access, access control lists, security guards, host, and IP security policies, rules for restrictions and issues about hosts through firewalls, strict control of admin access to user data, isolation of data using active directories including authorization features, RBACs and special mechanisms for workload isolation within systems. Tools for identifying sensitive data and specific measures for classifying and protecting data should be used by PaaS cloud providers [127].

It is necessary to apply control-based rules that the cloud processor administrators to be secured and alerted when the PaaS user stores information classified as sensitive. In the PaaS providers are enabled with the right for opposing the processing of user data based on GDPR. These providers must provide tools that enable the data storage restriction, retention, or deletion of data after achieving the objectives of the data subject. Firewalls and network protection mechanisms must be included as security measures in the cloud provider [127].

Cloud providers of SaaS encounter responsibilities associated with the operations of data processing. Every action between SaaS providers and customers, whether in securing agreements or other activities, must be complied with the data protection regulation and other rules arising from this regulation (GDPR). If SaaS providers do not execute the data protection regulation for their clients, then the local authorities for personal data protection should impose amercements in order to obligate the relevant parties to respect the GDPR regulation [127].

The data controller and the data processor interact with each other; therefore they have duties, responsibilities, requirements, privileges, and rights. Data protection principles for SaaS cloud providers are applied to all tiers of the cloud including physical protection, infrastructure, and up to the data protection of software (applications). In the physical protection aspect, measures related to this layer should be taken and included such as against unauthorized access of personnel to data centers, and physical access control mechanisms from cameras to biometric devices. Network segregation and access control should also be applied [125].

Software must enable audit mechanisms in order to log and notify about data view, modification, and utilization based on warning customization rules. SaaS should not be profiling users according to their private (sensitive) information if they are collected directly by these providers while completing the user profile or obtaining information automatically from the software itself. Data Protection by Design and by Default is considered the best way to fulfill the above requirements. In this way, we build services that control which data is necessary to use and which is not and why to use them. SaaS providers should include security and data breach notifications and they need to notify data controllers about data leaks and personal information [128].

In order to ensure that the cloud provider complies with the data protection principles of the GDPR, we should control them and incorporate necessary security measures. The data minimization technique should also be applied to delete unnecessary data stored on certain cloud devices, to ensure the collected data, accurately define and document appropriate security policies against relevant data. The cloud provider must fulfill the requirements and legality of the GDPR-based data processing; apply appropriate audit mechanisms that mainly audit data, information, and activities around them by

including their sources, active directories, operating system logs, storage analytics, data processing and alerts about their security [125], [127].

4.6. Discussion

The section provides comprehensive information about service authentication and authorization as an essential part of the cloud computing service architecture. Based on the key performance indicators (KPIs) of the research model [129] related to the security of cloud computing services, it is recommended to use the RBAC and ABAC mechanisms because they best fulfill the cloud security parameters compared to MAC and DAC.

The DAC mechanism is typically used simplest with legacy services and requires significant management expenses in contemporary multi-users (tenant) and multi-application environments such as cloud-distributed systems. The MAC mechanism enables the mapping of user resources, and it is considered more suitable for distributed systems than the DAC. Multi-layer security systems mainly use the MAC. The RBAC mechanism is much more scalable compared to the DAC and MAC mechanisms. It is also considered very suitable for cloud environments because it enables non-tracking of the identity of the cloud service users. The disadvantage of the RBAC is in terms of changing roles, which they change from time to time and there is always a need for real-time environment. This lack of semantic support also leads in facing a semantic gap amongst the authentication and authorization mechanisms of the service.

DAC and MAC are the authorization mechanisms whose security is based on identity, therefore it is not preferable to use them in open clouds, because the resource nodes may not be friendly on the cloud environment or may not recognize interactive nodes. It is necessary to propose a security policy, which can work exactly with the ABAC model in cloud computing, due the security policy is accountable for picking the substantial attributes that are used to create access decisions. ABAC and RBAC models are used in the Amazon Web Services and such use cases are Repp Health[3], WeWork

[3] This is available at: https://aws.amazon.com/solutions/case-studies/repp-health-case-study/

Fieldlens[4], and McDonald's[5]. RBAC and ABAC mechanisms are also used in Microsoft Azure and such use cases are the University of Toronto[6], AkzoNobel[7], and Intercontinental Hotel Group (IHG)[8].

4.7. Conclusion remarks

The cloud service is considered an important digital solution because it reduces the capital and operational services of the organization. Security risks and threats are major concerns of cloud computing because its nature is multi-tenant and third-party delegation to maintain the environment of this technology. This section has analyzed and surveyed current security issues, access control mechanisms, and potential mitigations involved in cloud services, with particular emphasis on the technologies and mechanisms of authentication and authorization needed to manage access, security, and services in the cloud environment. It discusses different topics about authentication and authorization mechanisms and the main aspects related to each mechanism in cloud computing services.

The survey of the various authentication and authorization mechanisms, their cloud-related architecture, and the different services offered by this technology highlight the need to improve existing authorization and access management models and authentication services. Here is also studied the auditing of cloud computing, with particular emphasis on its mechanisms that enable us to analyze, and evaluate data (resources) and processes for improving the cloud services. In this regard, cloud auditing is discussed about its mechanisms for assessing cloud services in the security context, respectively the issues of authentication and authorization of cloud services.

According to the researched literature, we have found that the RBAC model is considered the most used authorization and access control mechanism. MAC and DAC are the most reliable mechanisms among the systems that are examined. They are not so preferred to be used as a single due to the flexibility of MAC and the low security that DAC has compared to MAC. Authentication

[4] This is available at: https://aws.amazon.com/solutions/case-studies/wework-fieldlens/
[5] This is available at: https://aws.amazon.com/solutions/case-studies/mcdonalds/
[6]This is available at: https://customers.microsoft.com/en-us/story/819757-university-of-toronto-sap-on-azure-higher-education-canada
[7] This is available at: https://www.mygreatlearning.com/blog/microsoft-azure/
[8] This is available at: https://customers.microsoft.com/en-us/story/ihg

models are also presented in this section. The described authentication technologies and mechanisms can be appropriately combined to provide better security, or a secure authentication method can be developed for effective authentication of cloud computing services.

The appropriate and secure authentication and authorization mechanisms and protocols for cloud systems are suggested to be designed and developed in the future to improve the effectiveness of these domains in terms of cloud service security. GDPR has an extraordinary role with its mechanisms in data protection of all technologies, especially for cloud computing services.

5. THE AUTHORIZATION MANAGEMENT FRAMEWORK FOR DISTRIBUTED RESTful SERVICES

The open standards communities that produced web services generated a variety of security standards that are valuable for web services, and they are frequently used in SOA implementations, to assure security in a loosely connected SOA context [22].

The WS-security specification describes improvements in messages sent over SOAP to ensure integrity, confidentiality, and authentication of them. The authorization of web service does not currently have a standard framework. Various research groups are attempting to create authorization frameworks and principles for web services. Following the authentication procedure, most web services-based applications use application-specific access control mechanisms to make authorization decisions [7]. This contributes to a tendency to frequently try to reinvent the right elements, encouraging us to look more closely at the SOA authorization demands.

It is absolutely essential to provide the security of the systems that operate on the web, the messages that are exchanged, and the communication channels should also be ensured, to enable cooperation within the service-based organization. Confidentiality, authentication, authorization, integrity, and availability are all basic security elements that must be met and guaranteed because they are also essential for the security of web services and are considered active subjects whose application is yet researched in academia and industry [20].

Google, Facebook, and other similar companies typically implement control and management of service authorization as part of their operations. These corporations utilize the OAuth 2.0 protocol for the purposes of authentication and authorization. The process begins by initiating a request from the client application, which seeks the access token from the authorization server. It then extracts the token from the response and sends the corresponding token to its API which it intends to access. Facebook primarily comprises numerous services that are accessible to users and commonly allow access to user data through the utilization of OAuth 2.0. Similar to Facebook, the Twitter REST API relies on the OAuth protocol to facilitate authentication and authorization, ensuring secure access for applications [65].

The proposed framework's purpose is to manage the authorization of RESTful services and their access to increase the flexibility of services and their usability, namely the use of certain resources by services being previously authorized for such purposes. This approach enables stability, flexibility, and better use of services and their resources securely. For this reason, several REST services have been developed, implemented, and tested through a number of experiments in the proposed framework.

5.1. Authorization issues

HTTP offers integrative methods for user authentication, while the authorization mechanism (AM) enables the reliance on underlying resources. It is important to provide detailed information about the characteristics of entities involved in the authorization process in order to avoid limitations in the functionality of REST-based systems. Enabling the AM in architecture promotes stability, enhances portability, and facilitates interconnectivity between services, thereby reducing their overall complexity. By incorporating "resource-aware" permissions or rights [12] into the AM, access to authorization rules and permissions within the resource mapping is facilitated, enabling the determination of entity structure. This approach can have an impact on the authorization workflow by aiming to eliminate any constraints associated with the dependency on various resources.

The services that build on REST should support the integration and design approach in the authorization architecture, where they are limited by permissions at the structure level. Our approach is based on an inclusive system that needs to be upheld in the following constraints:

- *There should be support and consideration for different ways of presenting underlying resources.* Despite the presence of various entities, it is imperative to address and give due consideration to this issue. This particular aspect should not impact the reduction of service functionality and its available resources.
- *Various authorization permissions should be allowed to be applied to the resources.* Authorization permissions need to be taken into account within URIs, even when there are actions from HTTP methods that might be influenced by separate sets of permissions. Different permissions should incorporate supplementary details about resources and approaches for modifying them.

- *The availability of REST's extendibility should be ensured.* Enabling adherence to this parameter allows for the smooth adaptation of the service architecture, even when incorporating new operations, services, or resources. Expanding the range of available resources (content types) is facilitated through its adherence [12]. Any adaptations occurring within the architecture should be consistent and in accordance with the architecture of the AM.

The authorization mechanism (AM) is commonly referred to as a structured unit that operates using a layered model. Its role encompasses the management and handling of requests. The entire process is achieved by substituting the regular resource access with the AM. The AM incorporates resource enrollment, with the presence of a resource provider and service model that encompasses specific services. It provides the capability to store content directly within its structure and facilitates the forwarding of requests via links. The paths of services (resources) are used to establish the rules of the AM. This facilitates the inclusion of resources in the interconnection with operations, in accordance with HTTP access rules. Within a defined set of rules, the potential combinations of connections among services are boundless. Support for resource-aware filtering [14] is provided when determining the rules, regardless of the HTTP verb demands or entity attributes.

The AM functions as a structure that enables the authorization of services along with their respective resources. Various mapping rules are used to define the authorization structure, taking into consideration HTTP operations, functionalities, resources, and "resource-aware" [12] filters. The AM performs an identification and evaluation of specific request(s) before interacting with the corresponding resource(s). Rules that pertain to a specific HTTP operation are associated with a unique URI. In the AM, each authorization is designed to identify and perform operations on specific resources that are assigned to it.

The foundation of the AM structure primarily relies on rules, permissions, and authorizations. Taking into consideration the above-mentioned issues, the respective structure encompasses four aspects:

- Through numerous rules the enrolled URIs can be asserted. A variety of authorization sets can be achieved by utilizing the URI manipulation options provided by GET, POST, PUT, PATCH, and DELETE operations [68].

- Each rule within the AM is assigned by mapping it to specific operations derived from the available sets of HTTP methods. As a result, distinctive REST awareness is guaranteed within every AM rule.
- Furthermore, resource-aware authorization filtering is available as an optional feature in addition to URI and REST-level authorization [12]. When the filtering is separate from the information retrieved in the AM, it is necessary for it to be conscious of the data characteristics and metadata.
- By using links, resources can be addressed and stored within the AM. In both cases, the content of the resources stays linked to the mapping rules and permissions.

5.2. The proposed authorization management framework of RESTful services

REST communication relies on the utilization of the HTTP data transfer layer and standardized operations to facilitate the retrieval and transmission of data from services through a web browser. REST provides numerous benefits, including simplicity, a unified interface, scalability, high performance, and flexibility. It ensures reliability, supports independent development of components, minimizes interaction latency, enforces security measures, and more [8].

The central element in the proposed framework is the authorization mechanism (AM), which allows services to be enrolled and interact with one another through a shared interface. The AM operates for individual services that are segregated both in physical aspects and processes. As a result, the framework facilitates stability protection, enhances portability, promotes interactivity, and reduces the complexity of services.

The AM maintains service keys and generates the tokens as required. It also includes a registered action list, which represents the available actions offered by each respective service. For example, in case an endpoint is provided, the AM can provide a range of possible URLs along with their corresponding HTTP methods. This pairing of URL and HTTP verbs can be employed to describe services and establish permissions. These rules facilitate the development of more advanced permissions for various systems. Whenever a request is made for different services or resources, the AM will allocate a unique token from the application service for each requested service to grant access.

It has the capability to verify whether a service possesses the appropriate authorization to perform a specific action. The services provide their identity to the AM to authenticate themselves. They communicate with the AM to authenticate their request(s) identity and verify authorization, which includes controlling the service access rights and permissions to restrict resource access (Figure 5.1.).

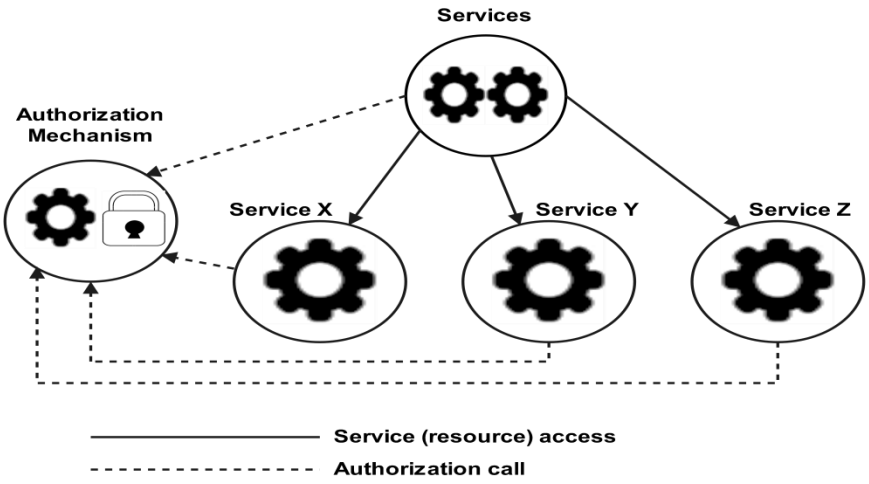

Figure 5.1. RESTful services with the authorization mechanism

The service application is required to generate a distinct token for the service seeking access. In the absence of such separation, there is a risk that the service may misuse the service token to gain unauthorized access to other services. Certain aspects necessitate support from services:

- Before granting any permission, it is important to offer various services the chance to update and enhance all applications (services). Testing should be conducted as the first step for the service, followed by the requirement of permission, and ultimately transitioning to the true endpoints.
- Endpoint versioning is essential for enhancing flexibility. During the deployment of new versions, it should be possible to ensure that the current services remain unaffected.
- In order to achieve more flexibility, services should take on the responsibility of load balancing.

A sequence diagram, shown in Figure 5.2, illustrates the execution sequence of the proposed framework:

- The service sends a request to the AM for acquiring a token to access the particular service.
- The AM generates a token, retains it, and subsequently sends it back to the service. Validation of a token should be performed for a single service, while for multiple services, a determination is made according to the list of roles in the AM.
- The service sends the request to another service using a token ensured by the AM for the respective service.
- Before permission is granted to the service, it checks the validity of a token by invoking the AM. A service consists of its unique identifier, the requested service, and the local state. After evaluating the request, the AM sends a response to the service, indicating whether access is allowed or denied.

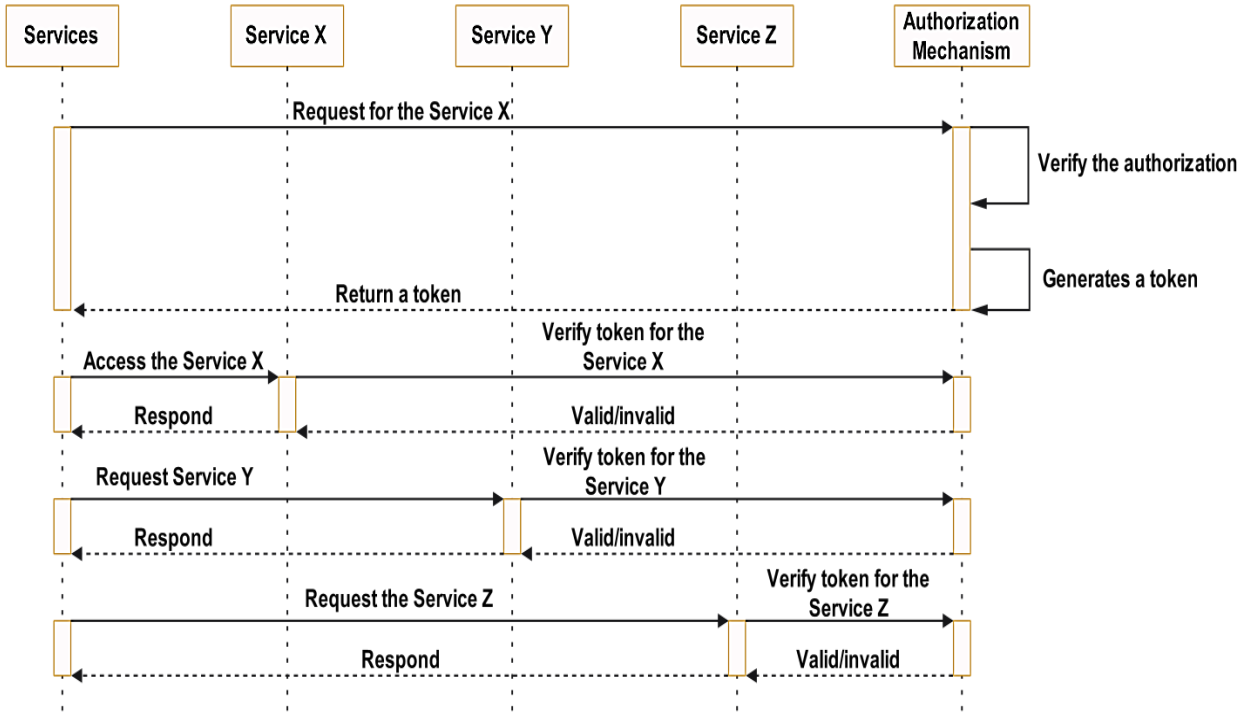

Figure 5.2. The proposed framework with RESTful services and the authorization mechanism

Table 5.1. The service enrollment to the authorization mechanism

Service	Service Key	Secret Code	URL (endpoint)
X	244	n85S15VbzT	https://webservice.domain.com/v1/courses
Y	356	tBj3841sY6	https://webservice.domain.com/v1/courses/1/students
			https://webservice.domain.com/v2/courses/2/students
Z	456	Fg9GygTy2t	https://webservice.domain.com/v1/courses/1/professors https://webservice.domain.com/v2/courses/2/professors

The AM grants the token lifespan, according to the preliminary setup of the service (the service priority enrollment) to the AM roles' list (Table 5.2) and in conformity with the authorization principles. It may use any compatible access and control features.

Table 5.2. The enrolled roles' list in the authorization mechanism

Service Key	URL (endpoint)	Token	Expiration date
244	https://webservice.domain.com/v1/courses	d4gc239d-3681-5244-8a38-754fc245b299	December 20, 2023 00:00:00
356	https://webservice.domain.com/v1/courses/1/students	eggc239d-3681-5244-8a38-754fc245b299	November 20, 2023 00:00:00

456	https://webservice.domain.com/v1/courses/1/professors	d4gc239d-3681-5244-8a38-754fc245b299	November 20, 2023 00:00:00
568	https://webservice.domain.com/v1/courses/1/announcements	7fbg4579-f787-5ela-058g-g5b35b6fgff2	October 20, 2023 00:00:00
673	https://webservice.domain.com/v1/courses/1/courseWorkMaterials	ce1fc841-282b-51c6-0944-gb6800d1d4fb	December 20, 2023 00:00:00

5.3. Experimental results and discussions

The occurrence of delays resulting from authorization checks can be regarded as a critical problem. In case the delay is excessive, then the AM can become an obstacle to slow down the whole work of the service (application). Therefore, understanding the delay of how it is being created and measuring it in practice is essential. The total time (T1) is calculated for verification of authorization request and response, and generation of the token from the AM while, after identifying the authorization in the AM, is calculated the total time (T2) for processing the service (resource) request and response. Therefore, the total time (T) for processing to access the service (resource) is T = T1 + T2.

The time intervals are depicted in Table 5.3. The first stage includes the time intervals T_AuthVerify and T_TokenGenerate as the total time T1 to verify the authorization request and generate the token from the AM for service (resource) manipulation by services based on the token that is requested in the AM. The second stage includes the time intervals T_S, SXandAMwithAuth, T_S,

SYandAMwithAuth, and T_S, SZandAMwithAuth that are presented as the total time T2. At this stage, the services request access to different services (for example, in service X, service Y, or service Z), and each time, their token is verified in the AM for accessing the specific service.

Table 5.3. The time intervals of services and resources' requests and responses processing based on authorization consideration

T_AuthVerify	The execution time of request, verification, and response between the services and the AM.
T_TokenGenerate	The execution time of request, verification, generates token and response between the services and the AM.
T_S, SXandAMwithAuth	The total execution time of requests for access and responses between the services, the service X, the AM, and protocol processing time (e.g. TCP/IP and HTTP) with token-based authorization.
T_S, SYandAMwithAuth	The total execution time of requests for access and responses between the services, the service Y, the AM, and protocol processing time (e.g. TCP/IP and HTTP) with token-based authorization.
T_S, SZandAMwithAuth	The total execution time of requests for access and responses between the services, the service Z, the AM, and protocol processing time (e.g. TCP/IP and HTTP) with token-based authorization.

We have developed a test setup to test and check the performance as well as measure the functionality of the authorization management system, including its components, processing, and interoperability between services. On a single machine are located some services, service X, service Y, service Z, and the authorization mechanism, as part of the test setup. Services are deployed on a single machine to minimize the effects of request and response (network) execution. The test was performed by running services on an I7 quad-core computer with a clock speed of 4.2 GHz, while 4 GB of

memory was dedicated only with the intention of testing the setup. The REST services presented in the authorization management framework are fully analyzed through the test setup. This means that we developed a RESTful API (services) in the Python programming language and created a meta-model example as in Table 5.4.

Table 5.4. Description of RESTful service resources as a metamodel for /courses/1

Resource	**HTTP Methods**
/courses/1	PUT
/courses/1/students	GET, POST, PUT, PATCH
/courses/1/students/1	GET, PUT
/courses/1/students/2	GET, PUT
/courses/1/students/1	DELETE

We have concluded that processing and interaction time does not depend on the volume of resources referenced in a depicted domain. As a result, a few resources are needed to test the authorization management system's performance. The course, as well as the specific parts, has been needed for testing reasons. We developed requests for specific resources that include various user categories (student or professor). Individual requests are executed a minimum of hundred times. In Figure 5.3 are listed the processing times (in milliseconds) calculated for the minimum, maximum and average times for a request which was observed in the case when it was sent to the authorization mechanism for verification of the authorized access to the specific service (resource). As remarked in the results presented in Figure 5.3, the processing times are small and differ from each other.

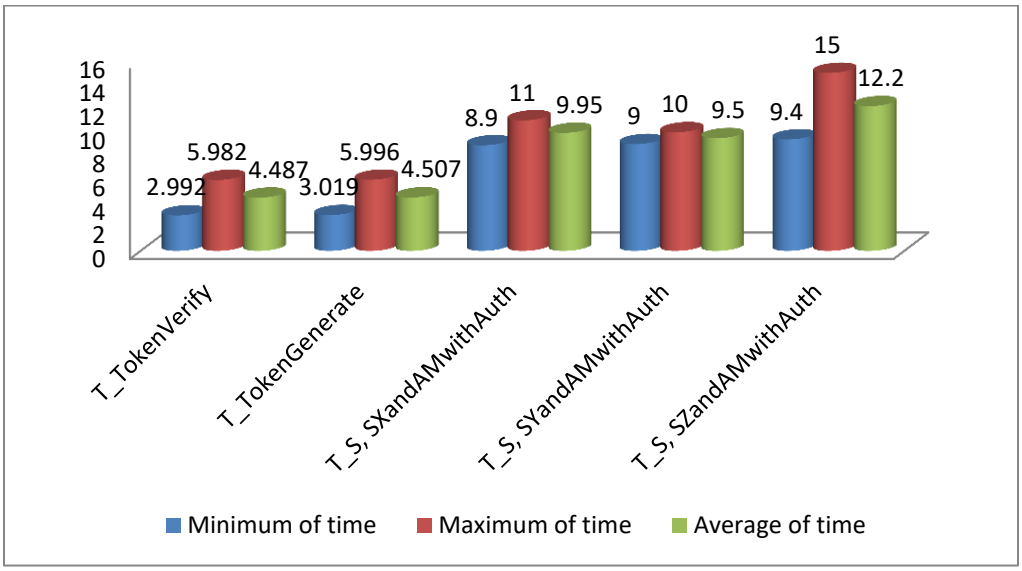

Figure 5.3. The results obtained during testing based on processing times and authorization consideration

According to the results depicted in Figure 5.4, the average delay, represented by Delta times, is 4.31 ms for service-to-service requests and 4.36 ms for service-to-authorization mechanism requests. This can also be verified by taking into account the average execution time of token generation/verification steps. Additional delays caused during the authorization management system processing time are not taken into consideration in the test setup. Furthermore, if the current resource representation should be adapted, the manipulation time may change significantly. If we should adapt the representation of the service resources for humans, then more computations will be required.

When we built APIs, we kept in mind the need for easy use, interoperability, and integration for our APIs. The implementation platform was built using the Python programming language, where for API and JSON Web Token (JWT) specifically, we used Flask and PyJWT. The Flask is a Python-based micro-framework used to build REST API services. The PyJWT is a Python library, which allows us to encode and decode JWT.

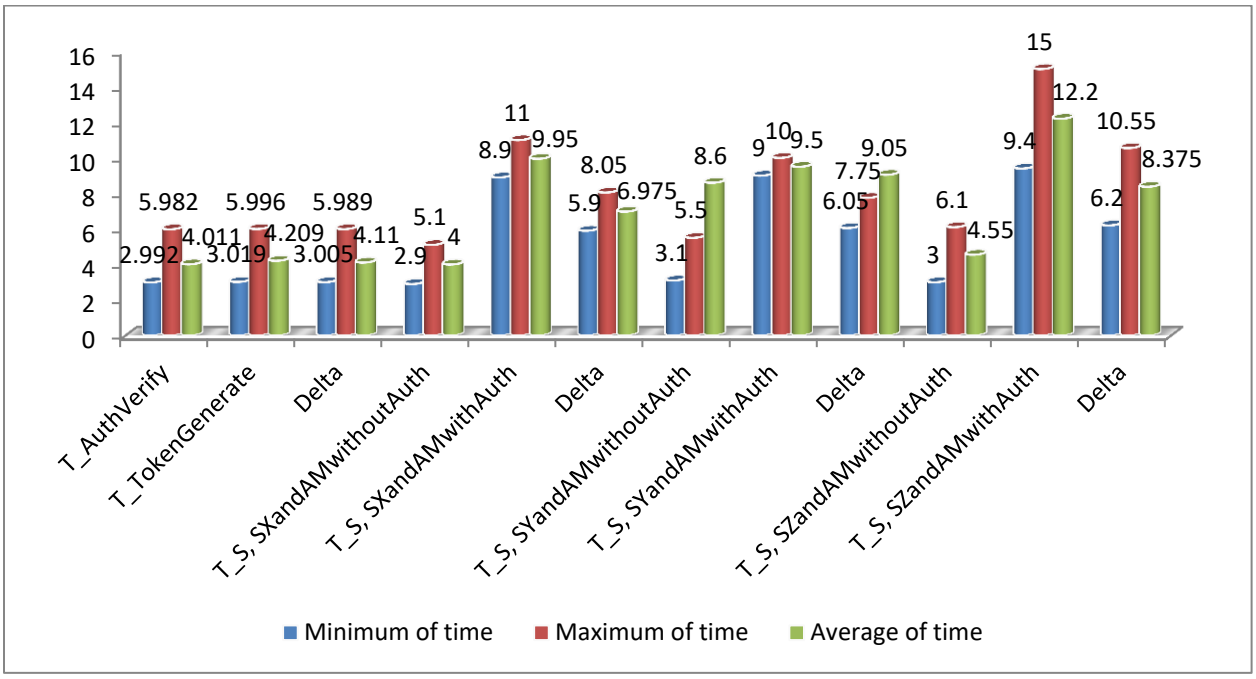

Figure 5.4. The results obtained during testing based on processing times with and without authorization consideration

6. SERVICE AUTHORIZATION MANAGEMENT BASED ON THE PROPOSED FRAMEWORK FOR RESTful SERVICES

Generally, the communication of REST services is enabled using the HTTP transport layer [12] and its standard methods, through which the presentation of resources from the services through the web browser is performed. The proposed authorization management framework focuses on service access authorization management and provides opportunities for resource manipulation in a system consisting of services that can be used in academic institutions or enterprises. In this system, services can interact with each other and can access each other's resource(s) being authorized to manipulate the respective resources, otherwise, the access will not be allowed if the authorization is not verified in advance. The services operate separately, while their authorization is verified by the authorization mechanism, which has the role of verifier of the service access to the other service if the service is authorized and can use certain service resources. This enables stability and security in the system, increases portability and interactivity, and minimizes the complexity of services.

The proposed framework for managing the authorization of services, except consisting of the services emphasized in the framework (Figure 6.1), also includes the authorization mechanism (AM) - authorization service which manages the authorization (access) and verifies it for specific services (resources). It generates tokens according to requests of services for access to services or resources. Usually, the service permissions and descriptions are defined based on the combination (service + HTTP method + URL) and they are provided by AM for certain endpoints. The token separated by AM to the service that requests access is controlled by this mechanism to verify whether the service is authorized to perform certain actions on the other service. Load balancing is left to the self-management of each service in order to have more flexibility in the system.

The authorization management framework is described step by step and is presented through the sequence diagram as shown in Figure 6.1:

- Service X requests access to service Y that is verified in the authorization mechanism (AM). Once AM verifies the authorization/access of the respective service, generates the token, and returns it as a response to service X *(case 1).*

- Service X attempts to access in service Y through the token received from the AM. Initially, service Y verifies the service X token in the AM for its validity. The AM responds to service Y, whether the token is valid or invalid depending on the request (service + URI + HTTP method) submitted by service Y. If the token is valid, service X is authorized to access service Y, if the opposite happens (invalid token), then service X is not allowed to access service Y *(case 2).*
- Service X requests a resource (URI + method) on the Resource Server (RS). The RS verifies the Service X request in the AM based on the Service X token presented. If the token is valid, the access and manipulation of the resource are allowed, and it means that Service X is authorized to access the relevant resource. If the token is invalid, the access to the RS is denied and Service X has no authorization to manipulate and access the specific resource *(case 3).*

The proposed framework increases the effectiveness of the authorization. It enables greater control of authorization, because each service must first be verified in the AM and receives a token, and then access the resources of the other service using the token issued by the AM. The second verification occurs in the RS, which based on the token presented by the service allows or denies access to resources of other services (this process is evidenced in the cycle of the proposed framework (Figure 6.1)). To speed up the process and reduce the number of invokes to the AM, the service can cache the token lifetimes locally. This allows for multiple requests from the service without calling the AM. Initially, services must be enrolled in the AM to obtain an authentication (access) token, as shown in Table 6.1.

Table 6.1. The service enrollment to the authorization mechanism

Service	Service Key	Secret Code	URL (endpoint)
X	244	n85S15VbzT	https://webservice.domain.com/v1/courses
			https://webservice.domain.com/v1/courses/1/students
			https://webservice.domain.com/v2/courses/2/students
Y	356	Fg9GygTy2t	https://webservice.domain.com/v1/courses/1/professors
			https://webservice.domain.com/v2/courses/2/professors

The AM grants the token lifespan, according to the preliminary setup of the service (the service priority enrollment) to the AM roles' list (Table 6.2) and in conformity with the authorization principles. It may use any compatible access and control features.

Table 6.2. The enrolled roles' list in the authorization mechanism

Service Key	**URL (endpoint)**	**Token**	**Expiration date**
244	https://webservice.domain.com/v1/courses	d4gc239d-3681-5244-8a38-754fc245b299	December 20, 2023 00:00:00
356	https://webservice.domain.com/v1/courses/1/students	eggc239d-3681-5244-8a38-754fc245b299	November 20, 2023 00:00:00
456	https://webservice.domain.com/v1/courses/1/professors	d4gc239d-3681-5244-8a38-754fc245b299	November 20, 2023 00:00:00
568	https://webservice.domain.com/v1/courses/1/announcements	7fbg4579-f787-5ela-058g-g5b35b6fgff2	October 20, 2023 00:00:00
673	https://webservice.domain.com/v1/courses/1/courseMaterials	ce1fc841-282b-51c6-0944-gb6800d1d4fb	December 20, 2023 00:00:00

As observed in the workflow, the AM has the capability to establish various policies for accessing the service. The AM has the ability to either grant a service full access to all resources on another service or implement a more precise policy in which specific resources are designated for

access by a particular service. This feature enables authorization management to be centralized by authorized individuals, freeing developers from the responsibility of managing access and only requiring them to reach out in the AM prior to access.

Another crucial point to note is that the flow is structured in nature. This enables the implementation of libraries for handling authorization within particular development tools and technologies. Through this approach, the RESTful services' development can be significantly streamlined and simplified to the point of merely declaring the import of a library. The utilization of libraries reduces the security risks associated with faulty implementations. However, there are advantages and disadvantages when using libraries.

Advantages: efficient coding: libraries allow developers to reuse pre-existing code rather than starting from scratch, resulting in more efficient coding; time-saving: utilizing libraries can save development time by allowing them to focus on other aspects of the code, leading to faster software development; security: using established and well-known libraries can help reduce security risks associated with faulty implementations.

Disadvantages: depending on third-party code: when using libraries, developers are reliant on third-party code, which can introduce security vulnerabilities that are beyond their control; compatibility issues: in some cases, using libraries can lead to compatibility issues with other parts of the software, resulting in bugs and security vulnerabilities. In conclusion, while utilizing libraries can bring a number of benefits, it is important to carefully consider the potential risks and ensure that the libraries being used are up-to-date, well-tested, and secure.

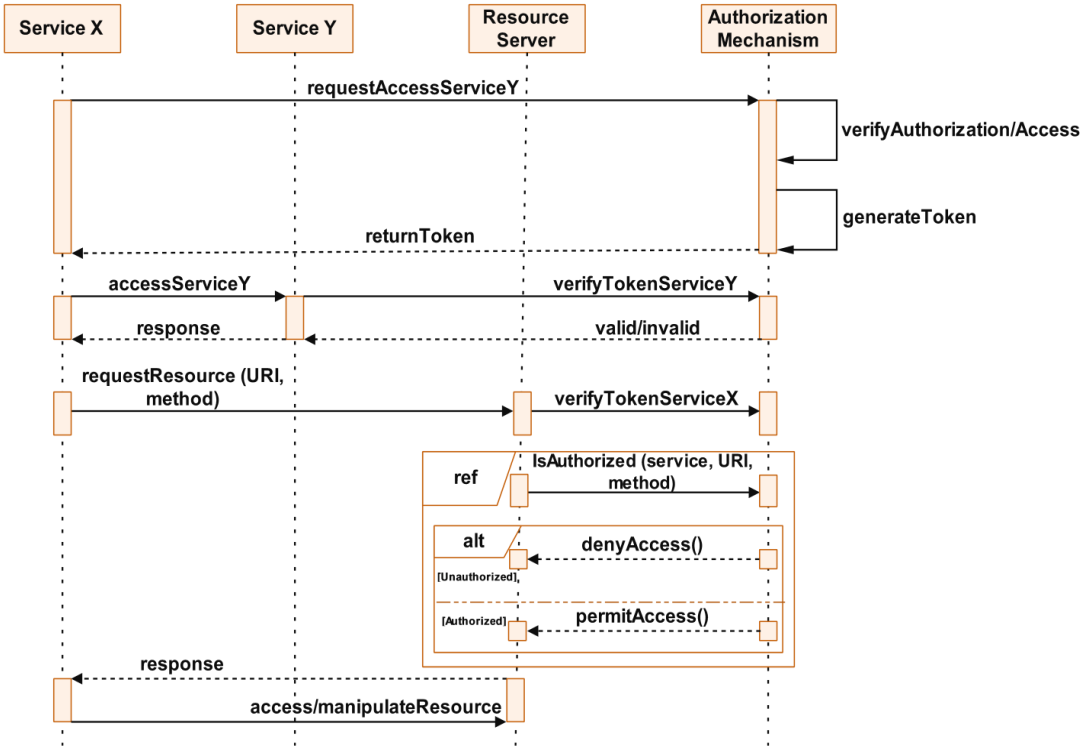

Figure 6.1. The proposed authorization management framework of RESTful services

The proposed framework enhances security and further strengthens the service access control to other service resources with the condition that the service be authorized for using specific service resources. This occurs due to the double control of the token for access by the responsible services for authorization issues. Therefore, this approach prevents misuse of the service resources and their access by unauthorized actors. The double authorization verifications cause delays in the system, especially when many requests are performed by different services to access other services or resources. This presents itself as a main challenge. Such delays are small, almost negligible in the case of applying the framework to an in-house system. This problem is confirmed in the following section of the study, in which the results related to the testing and application of the proposed framework are presented. Moreover, the proposed framework enhances the security and dependability of services by ensuring that the authorized services can access the resources within the service. This can help prevent unauthorized access or misuse of the service and its resources, particularly crucial for services that deal with critical or sensitive information.

Integration and interoperability are important in modern computing environments because they allow different systems and services to work together and share data seamlessly. This can improve efficiency and productivity, as well as provide new capabilities and functionalities [20]. However, achieving integration and interoperability can be challenging, as it requires careful planning and design to ensure that different systems and services are compatible and able to exchange data in a meaningful way. In the proposed framework, interoperability and integrity are evident in the following cases:

- Service X requests access to service Y. The request is verified by an authorization mechanism (AM). The AM is responsible for verifying whether service X has the necessary authorization or access to use service Y. If service X is authorized, the AM generates a token and returns it as a response to service X. The process described in this case is an example of integration and interoperability between the two services. In this context, service X and service Y are able to integrate and interoperate because they are able to exchange data and perform specific tasks on each other's behalf through the authorization mechanism. The token generated by the AM can be used as a means of authentication and authorization for service X to access the resources and functionality of service Y. The token serves as proof that service X has been authorized to access service Y and can be used to identify and verify the service's identity and permissions. The use of a token in this context helps to provide security, integrity, and interoperability between service X and service Y.
- Service X is attempting to access service Y using a token that it has received from an authorization mechanism. Service Y is responsible for verifying the validity of the token before granting service X access. To do this, service Y submits a request to the authorization mechanism, which includes information about the service (service X), the specific URI that service X is trying to access, and the HTTP method being used (e.g. GET, POST, etc.). The authorization mechanism responds to service Y with a message indicating whether or not the token is valid. If the token is valid, service Y grants service X access to its resources. If the token is invalid, service Y denies service X access. This process is important because it helps to ensure that only authorized parties are able to access the resources of service Y. By verifying the validity of the token, service Y can ensure that service X has the necessary permissions to access the requested resources. This helps to maintain the integrity and security of service Y's resources. Integration and interoperability issues are raised when service X and service Y work

together seamlessly. For example, if the AM used by service Y recognizes the token of service X, or if the request submitted by service Y includes all of the necessary information, service Y must verify the validity of the token and grant access in the AM. In these cases, service Y is properly integrated and it has interoperability with service X, ensuring smooth and seamless access to resources.

- Service X refers to the service that is attempting to access a resource on a resource server (RS). In order to access the resource, service X must present a token to the RS, which will then verify the token through the AM service. If the token is valid, it means that service X is authorized to access and manipulate the resource. If the token is invalid, it means that service X is not authorized to access or manipulate the resource, and access to the RS will be denied. This process helps ensure the security and integrity of the resource server and the resources it stores. It also helps ensure that only authorized services can access and manipulate the resources, which is important for maintaining the confidentiality, integrity, and availability of the resources. This process is used to ensure that only authorized services are able to access and manipulate resources on the RS. It helps to maintain the security and integrity of the resources by preventing unauthorized access and manipulation. The use of tokens and an AM service allows for integration and interoperability between different services, as it allows different services to access and use resources on the RS in a standardized way.

Overall, the described processes involve the integration and interoperability of different services in order to manage access to resources and ensure that only authorized entities are able to access them. The proposed framework can be integrated with other systems or platforms that may have different authentication and authorization mechanisms by implementing a token-based authentication and authorization protocol, such as OAuth2 or JWT. The goal is to address service authorization issues by exporting applications as fundamental services through OAuth 2.0 and REST services, effectively integrating the services. This protocol allows for the exchange of tokens between services and systems, and the verification of those tokens against a centralized authorization server. In case 1, service X can use OAuth2 or JWT to authenticate and authorize its access to service Y. Service Y can generate a token after verifying the authorization and access of service X in the AM. This token can be used by service X to access service Y's resources without needing to re-authenticate with the AM.

In case 2, service X can use OAuth2 or JWT to authenticate and authorize its access to service Y. Service Y can then verify the token against the AM for its validity before granting or denying access to its resources. In case 3, service X can present its token to the RS, which can then verify it against the AM for its validity. If the token is valid, service X is authorized to access and manipulate the requested resource. If the token is invalid, service X is denied access to the RS and its resources. In all cases, the proposed framework can be integrated with other authentication and authorization mechanisms by using a token-based protocol, such as OAuth2 or JWT, to facilitate the exchange and verification of tokens between services and systems.

Implementation of the proposed framework is performed using the Python programming language, where for services specifically, we used Flask. The Flask micro-framework [130] built on Python; is used to develop REST services. This means that we have developed RESTful services in the Python programming language as shown in the next section.

6.2. Authorization mechanism as an authorization service

The authorization mechanism (AM) is responsible for managing the service keys and generating tokens as needed. Its task is also to enlist the actions provided by each service. It offers a collection of available URLs with their corresponding HTTP methods for a particular endpoint. The URL and HTTP method pairing can function as a means of describing services and defining authorization permissions. A service will use tokens to access various services directly. The AM is responsible for verifying the tokens' authenticity when the service tries to access specific service(s) or resource(s). It may also confirm on the AM if the service has the required authorization permissions for accessing a particular action.

Below is the definition of the authorization process and mechanism that will manage (handle) the authorization between services that are communicating with each other, as it is proposed. It should be noted that in this particular architectural setup, the role of the trusted entity is assigned to the AM, and all the services involved rely on the AM to handle their authorization-related issues. Beforehand, it is necessary to register each service within the AM. The crucial point is that every service has its credentials, which are utilized to verify its identity when accessing the AM. We have demonstrated the authentication process using fundamental principles of basic authentication; however, any appropriate

authentication technique can be employed in this aspect. Additionally, tokens and other analogous data are shortened to enhance readability. In the following is illustrated the authorization procedure in a scenario where service X aims to access a resource within service Y:

a) To gain access to a particular resource, service X must obtain a token from the AM. Given that a distinct token is necessary for accessing each service, the request must specify the service for which the token is needed.

```
POST /tokens HTTP/1.1     # HTTP method (POST) and the target URL ("/tokens") with HTTP version (HTTP/1.1)
Host: amservice.example.com   # The hostname of the server to which the request is being sent
Authorization: Basic NyPpbneyXHZ4MnbyME...   # Basic Authentication with Base64-encoded credentials
Content-Type: application/json   # The type of the content in the request body (JSON)
...
# The JSON payload starts here
{
    "service": "X"   # JSON data with a single field "service" and its value "X"
}
```

b) Assuming that service X has the authorization to access the requested resource, the AM will generate, or provide an already existing token, along with the corresponding validity period.

```
HTTP/1.1 201 Created   # HTTP response status code (201 Created) with the HTTP version (HTTP/1.1)
...   # Other response headers may be present, but they are omitted for brevity
Expires: Wed, 27 Dec 2023 20:10:09 +0000   # The expiration date and time of the token
...
# The JSON response starts here
{
    "token": "913168gg87ge14847...",   # The token value, which is a string of characters
    "expires": "Wed, 27 Dec 2023 20:10:09 +0000"   # The expiration date and time of the token in human-readable format
}
```

c) In case service X lacks authorization for the particular requested service, the resulting response will be as shown below.

```
HTTP/1.1 403 Forbidden   # HTTP response status code (403 Forbidden) with the HTTP version (HTTP/1.1)
...   # Other response headers may be present, but they are omitted for brevity
# The JSON response starts here
{
    "reason": "Access to the service is denied"   # The reason for the 403 Forbidden status
}
```

d) Once a valid token has been obtained, service X can request a particular resource from service Y by utilizing the previously obtained token for authentication.

```
POST /courses HTTP/1.1
Authorization: Token 913168gg873gel4847...
...
# The JSON payload starts here
{
    "course_code": 2345,   # The code that identifies the course
    "course_name": "Distributed RESTful Services",   # The name of the course
    ...   # Other properties of the course
}
```

e) Service X verifies the token's validity by sending a request to the AM, along with information regarding the specific resource that is required.

```
GET /authorization HTTP/1.1
...
# The JSON payload starts here
{
    "token": "913168gg873gel4847...",    # The authorization token for verification
    "http_method": "POST",               # The HTTP method for which authorization is requested
    "resource": "/courses"               # The resource (endpoint) for which authorization is requested
}
```

f) The AM bears the responsibility of determining if service X has permission to access the designated resource within service Y. In the event of a lack of authorization, a response with a code of 403 will be returned.

```
HTTP/1.1 403 Forbidden   # HTTP response status code (403 Forbidden) with the HTTP version (HTTP/1.1)
...
# The JSON response starts here
{
    "reason": "Access to this resource is denied due to insufficient permission."
}
```

g) If authorization is not granted, service Y is required to forward the relevant response back to the service X.

h) When service X has been granted authorization, the AM will return a response with a code in the 20× range.

```
HTTP/1.1 200 OK   # HTTP response status code (200 OK) with the HTTP version (HTTP/1.1)
... # JSON response or other response content would follow here
```

i) In such a situation, service Y should proceed with processing the initial request and subsequently provide the appropriate response.

```
HTTP/1.1 201 Created   # HTTP response status code (201 Created) with the HTTP version (HTTP/1.1)
...
# The JSON response starts here

{
    "id": 78,                                       # The ID of the created resource
    "course_code": "2345",                          # The course code of the created course
    "course_name": "Distributed RESTful Services",  # The name of the created course
    ...
}
```

As illustrated in the preceding flow, the AM has the ability to establish various policies for accessing services. It has “the competency” to grant service permission to access wholly resources on another service or to utilize a more precise policy(s) where the AM relations a list of resources that can be accessed for a specific service. This feature enables authorized entities to centrally manage

permissions, thereby reducing the responsibility of developers to simply invoke the AM before accessing any resources (services).

To prevent the authorization mechanism from being a single point of failure in the proposed framework, we can implement a distributed authorization service (mechanism). This approach distributes the responsibility for authorization decisions across multiple nodes, which provides fault tolerance and improved performance. To ensure that the proposed framework can be extended without compromising security or introducing other concerns, it is important to design the system using modular components. Each component is designed to be as independent as possible so that new features and functionality can be added without affecting the overall security or stability of the system.

6.3. Potential limitations and challenges

One of the primary challenges in using the proposed framework is integrating it with existing systems or applications. The integration process may require modifications to the existing system, which could be time-consuming and potentially introduce new issues. The token design and management in the proposed framework may be challenging for some organizations. The management of the token requires a reliable and secure storage mechanism, and the token's size may affect the performance of the system. The design of the token used in the proposed framework has effectiveness in preventing tampering, forgery, and system security.

The additional overhead introduced by the authorization mechanisms in the proposed framework may impact the performance of the system. This impact needs to be carefully measured and managed to ensure that the system meets the required performance objectives. The adoption of the proposed framework may face challenges due to the need to modify existing systems or services to support the framework's authentication and authorization mechanisms. This challenge may require additional effort and resources to overcome. While the proposed framework is designed with scalability in mind, there may be limits to the number of requests and users that can be handled efficiently. As the system grows in size, additional resources and optimizations may be necessary to maintain performance.

6.4. The comparison and analysis of the proposed framework

In the study [14] is proposed and analyzed a framework for secure authentication and authorization of integrated REST services. It focuses on providing secure access control mechanisms to protect against unauthorized access based on OAuth. However, it does not specifically address RESTful services, and it does not provide detailed implementation or experimental results. The research study [18] describes an access control framework for REST services based on a role-based access control (RBAC) model. It provides detailed implementation and experimental results but does not specifically address RESTful services. The framework is based on the XACML policy decision point and XML, which is a widely used access control policy language.

In [131] is proposed a RESTful service access control framework based on tokens. It provides a detailed implementation and experimental results based on secure authentication and authorization RBAC mechanism for RESTful services. It used also a token-based authentication mechanism, to manage access control. The framework proposed in [32] presents security architecture for RESTful services, which provides a set of security mechanisms, including message signing and encryption, to protect against common attacks. It also presented a detailed analysis of the security mechanisms and evaluates their effectiveness in various scenarios. In comparison to our framework, it focuses more on message-level security and may require additional mechanisms for access control and authorization management.

Compared to these frameworks, our proposed framework provides a solution for managing service access authorization specifically for RESTful services. It utilizes a token-based authentication mechanism, similar to JWT, and provides detailed implementation and experimental results. Additionally, our framework provides a balanced approach for authentication and authorization management in RESTful services, with a focus on simplicity, security, and scalability.

To ensure that the proposed framework can be extended without compromising security or introducing other concerns, it is important that the system is designed using modular components. Each component is designed to be as independent as possible so that new features and functionality can be added without affecting the overall security or stability of the system.

For instance, what happens if service Y requires service Z? If service Y requires access to service Z, a similar authorization process would need to occur. Service Y would need to request access to service Z through the AM. Once the AM verifies the authorization/access of service Y, it would generate a token and return it as a response to service Y. Service Y would then attempt to access service Z using the token received from the AM. Service Z would verify the service Y token in the AM for its validity. The AM would respond to service Z, whether the token is valid or invalid depending on the request (service + URI + HTTP method) submitted by service Z. If the token is valid, service Y would be authorized to access service Z, and if the opposite happens (invalid token), then service Y would not be allowed to access service Z. This process ensures that only authorized services are able to access the requested resources, maintaining a secure and controlled environment.

What happens if the service is in another realm? If the service is in another realm, it means that the AM for that service may not be the same as the one used by service X. In this case, service X may need to obtain a new token for the authorization mechanism used by the other realm. This could involve authenticating with the AM of the other realm and requesting a new token with the appropriate permissions. Alternatively, if the two realms have a trust relationship, service X may be able to use its existing token to access the service in the other realm without needing to obtain a new token.

What happens if resources from two different resource servers are needed at the same time? If resources from two different resource servers are needed at the same time, service X would need to obtain valid tokens for each of the resource servers from the AM. Service X would then use these tokens to request the desired resources from each resource server. Each resource server would verify the validity of the respective token provided by service X through the AM and allow or deny access to the requested resource accordingly. If both tokens are valid, service X would be authorized to access and manipulate the resources from both resource servers. If either token is invalid, access to the respective resource server would be denied.

Does it work synchronously or asynchronously? Based on the cases described in the proposed framework, the system works asynchronously, as this depends on the details of each service and how they communicate with each other. In general, it is common for distributed systems like this to work asynchronously, where each service communicates with others through messages (requests) passing,

and events. This allows for better scalability, fault tolerance, and responsiveness, as services can work independently and concurrently without blocking each other.

What happens if one of the servers does not respond? If one of the servers involved in the authentication and authorization process does not respond, it can cause different scenarios depending on which server is not responding: if the AM does not respond, the authentication and authorization process cannot be completed, and access to the requested resource will be denied. If we assume that service Y does not respond after receiving the token from AM, service X will not be able to access the resource from service Y. If the RS does not respond after receiving the valid token from AM, service X will not be able to access the requested resource on the RS. In general, a lack of response from any server involved in the authentication and authorization process can lead to a denial of access to the requested resource. However, this also depends on the specific implementation of the system and the error-handling mechanisms in place.

In general, there may be a timeout set for the request to wait for a response, after which the request may be retried or an error message may be returned to the client. The behavior may also depend on the type of request and the criticality of the service being requested. For example, if the request is for a non-critical resource, the system may simply return an error message and the user can try again later. If the request is for a critical resource, the system may have a failover mechanism in place to redirect the request to a backup server to ensure the continuity of the service.

What is the protocol? Based on the fact that the system involves different services and resource servers communicating with each other, is used a web service architecture. In such architecture, services communicate with each other using REST. However, the OAuth 2.0 protocol is commonly used in scenarios like the ones described, where a third-party service (service X) needs to access a protected resource (service Y) on behalf of a user (or another service) using a token-based authentication and authorization mechanism. The communication between the different services and the authorization server typically occurs over HTTP.

How the access control matrix or its concretization is designed? To design an access control matrix or its concretization, we need to follow these steps: identify the subjects (services) that need access control; identify the objects (resources) that need access control; determine the access rights

needed for each subject on each object. Based on the cases described in the framework, an access control matrix (ACM) is shown in Table 6.3.

Table 6.3. ACM for the proposed framework services (PFS)

PFS	Service X	Service Y	AM	RS
Service X	N/A	R, W	N/A	R, W
Service Y	R	N/A	N/A	N/A
AM	N/A	N/A	R, W	N/A
RS	N/A	N/A	R	R, W

Service X needs read and write access to service Y, as it requests access and manipulates data on it (case 1). Service Y needs read access to the service X token in the AM, as it verifies the validity of the token and retrieves data from it (case 2). The AM needs read and write access to the service X token and authorization data, as it verifies the authorization of service X and generates and responds with the token (case 1 and case 2). The RS needs read access to the service X token in the AM, as it verifies the validity of the token and retrieves data from it (case 3). It also needs read and write access to the resource, as it allows or denies access and manipulation to the resource based on the validity of the token (case 3).

How is this framework different than, for instance, Kerberos? There are a few differences between the proposed framework and Kerberos. The framework uses a token-based approach for authorization, while Kerberos uses a ticket-based approach. Kerberos uses a ticket-granting mechanism, where a user or service obtains a ticket from a central authentication server and then presents that ticket to other services to gain access [132]. The framework, on the other hand, uses tokens that are generated by an authorization mechanism and presented by the requesting service to gain access to resources. The framework uses the AM to verify authorization/access, while Kerberos uses a key distribution center [76] to authenticate users and provide tickets. Kerberos is primarily an authentication protocol, whereas the framework described focuses on authorization. In other words, Kerberos is used to verify the identity of a user or service, while the framework is used to determine

whether a service is authorized to access a specific resource. This framework relies on HTTP-based protocols, while Kerberos is designed to work with any network protocol. Our framework can be used for services and cloud-based applications, while Kerberos was originally designed for use in the client-server architecture.

6.5. Experimental results and discussions

6.5.1. Implementation of REST services cases based on the proposed framework

Based on the proposed framework for managing the authorization of REST services, specifically the cases (case 1, case 2, and case 3) from section 6, the REST services have been implemented. Implementation of the proposed framework is performed using the Python programming language, where Flask is used for developing services. The Flask micro-framework [130] is built on Python. REST services from case 1 of the framework proposed in section 6, where service X requests access to service Y by first verifying its state in the AM, are shown as follows.

```
from flask import Flask, request
from flask_restful import Resource, Api
app = Flask(__name__)
api = Api(app)
class AccessToken(Resource):
    def post(self):
        # Get the request data
        data = request.get_json()
        # Extract the service names from the request data
        service_x = data['service_x']
        service_y = data['service_y']
        # Verify the authorization / access of service_y using the authorization mechanism (AM)
        access_granted = verify_access(service_y)
        if access_granted:
            # Generate the token
            token = generate_token()
            # Return the token as a response to service_x
            return {'token': token}, 200
        else:
            return {'error': 'Access denied'}, 403
api.add_resource(AccessToken, '/access_token')
if __name__ == '__main__':
app.run(debug=True)
```

A flask-based REST service with an endpoint/access_token is implemented using the AccessToken resource. The POST method of the AccessToken resource handles the request from service X, verifies the authorization/ access of service Y using the AM, and generates a token if the access is granted. The generated token is then returned as a response to service X.

REST services from case 2 of the framework proposed in section 6, where service X tries to access service Y using the token received by AM, where first the service verifies its state in the AM

and the validity of the token used, are shown below.

```
from flask import Flask, request
from flask_restful import Resource, Api
app = Flask(__name__)
api = Api(app)
class AccessServiceY(Resource):
    def post(self):
        # Get the request data
        data = request.get_json()
        # Extract the token and request details from the request data
        token = data['token']
        service_x = data['service_x']
        uri = data['uri']
        http_method = data['http_method']
        # Verify the token for service_x in the authorization mechanism (AM)
        token_valid = verify_token(service_x, uri, http_method, token)
        if token_valid:
            # Service X is authorized to access service Y
            # Perform the necessary actions to allow access to service Y
            allow_access()
            return {'success': 'Access granted'}, 200
        else:
            # Service X is not authorized to access service Y
            return {'error': 'Access denied'}, 403
api.add_resource(AccessServiceY, '/access_service_y')
if __name__ == '__main__':
app.run(debug=True)
```

A flask-based REST service with an endpoint/access_service_y is implemented using the AccessServiceY resource. The post method of the AccessServiceY resource handles the request from service X to access service Y, verifies the token for service X in the AM, and allows access to service Y if the token is valid. If the token is invalid, access to service Y is denied. The REST services from case 3 of the framework proposed in section 6, where service X requests a resource in the RS and RS verifies the request of service X in the AM according to the presented token, are shown as follows.

```
from flask import Flask, request
import requests
app = Flask(__name__)
@app.route('/resource', methods=['GET', 'POST'])
def access_resource():
    # Get the token from the request header
    token = request.headers.get('Authorization')
    # Send a request to the AM to verify the token
    verify_url = 'http://am.example.com/verify'
    headers = {'Authorization': token}
    verify_response = requests.get(verify_url, headers=headers)
    # If the token is valid, allow access to the resource
    if verify_response.status_code == 200:
        if request.method == 'GET':
            # Handle GET request to retrieve the resource
            resource = retrieve_resource()
            return resource
        elif request.method == 'POST':
            # Handle POST request to manipulate the resource
            manipulate_resource()
            return 'Resource manipulated successfully'
    # If the token is invalid, return a 403 forbidden error
    else:
        return 'Forbidden', 403
def retrieve_resource():
    # Retrieve the resource from the resource server
    resource_url = 'http://rs.example.com/resource'
    resource_response = requests.get(resource_url)
    return resource_response.text
def manipulate_resource():
    # Manipulate the resource on the resource server
    resource_url = 'http://rs.example.com/resource'
    resource_data = request.json
    resource_response = requests.post(resource_url, json=resource_data)
if __name__ == '__main__':
    app.run()
```

When a request is made to this route, it retrieves the token from the request header and sends a request to the AM to verify the token. If the token is valid, the code allows access to the resource and handles the request accordingly (either retrieving the resource or manipulating it). If the token is invalid, it returns a 403 forbidden error. The retrieve_resource and manipulate_resource functions handle the actual communication with the resource server to retrieve or manipulate the resource.

6.5.2. Implementation of REST services - university scenario

The framework has been also implemented in the university environment, and we have taken into consideration three university management system services, namely the administrator service, learning service, and e-library service, as cases.

6.5.2.1. RESTful services based request implementation

Table 6.4 outlines the service requests with the HTTP methods, resource collections, operations, and action description according to the proposed architecture.

Table 6.4. Requests of REST service

Request	Source service	Destination service	HTTP method	Resource collection	Operation	Action description
a	X	Y	GET	study_programs	Retrieve	Retrieve a collection of programs for studying.
b	X	Y	POST	faculties	Create	Create a new academic unit (faculty).
c	X	Y	GET	faculties	Retrieve	Retrieve a collection of faculties.

d	Z	Y	GET	books	Retrieve	Retrieve a collection of books.
e	X	Y	GET	students	Retrieve	Retrieve a collection of students.

As outlined in the proposed framework, the initial step is to present the service request based on functions (according to requests a and c), which includes the "insert_courses()" function.

```
# Define a function named insertCourses.
def insertCourses():
    # Call a method named 'getdirections' on the 'AdminModel' class and store the result in the 'result' variable.
    result = AdminModel.getdirections()
    # Make an HTTP request to 'https://smu.dev' and store the response data in the 'faculties' variable.
    faculties = RestService.requestServiceData('https://smu.dev', 'https://smu.dev/faculties')
    # Make an HTTP request to 'https://smu.dev/faculties/1/programs' and store the response data in the 'study_programs' variable
    study_programs = RestService.requestServiceData('https://smu.dev', 'https://smu.dev/faculties/1/programs')
    # Create a dictionary containing data to be passed to a view.
    # The 'json.dumps(result)' part serializes the 'result' variable to a JSON string.
    # The resulting dictionary has three key-value pairs: "data", "faculties", and "programs".
    return View.make('admin/insert-courses', {"data": json.dumps(result), "faculties": faculties, "programs": programs})
```

The service then sends a request to the AM with a specific URL.

```
# Define a class named 'RestService'.
class RestService:
    # Define a class-level variable 'appServiceID' with a default value.
    appServiceID = "db0654253d86a9cb7b8dde8d54548ef2d6cf4e12"
    # Define a static method 'requestServiceData' that takes 'service' and 'requestUrl' as parameters.
    @staticmethod
    def requestServiceData(service, requestUrl):
        # Get the access token from the session if it exists.
        token = Session.get('access_token')
        # If the access token is not available, retrieve it using the 'getAccessToken' method.
        if token is None:
            token = RestService.getAccessToken()
        # Check if an access token is available.
        if token:
            # Create a 'Service' instance.
            service = Service()
            # Make an HTTP POST request to 'http://asm.dev/authenticator/request-service' with specific data.
            response = service.post('http://asm.dev/authenticator/request-service', data={
                'serviceID': RestService.appServiceID,
                'service': service,
                'request_url': requestUrl,
                'access_token': token
            })
            # Parse the JSON response and return it.
            return json.loads(response.text)
    # Define a static method 'getAccessToken'.
    @staticmethod
    def getAccessToken():
        # Create a 'Service' instance.
        service = Service()
```

```
    # Make an HTTP POST request to 'http://asm.dev/oauth/access_token' with specific data.
    response = service.post('http://asm.dev/oauth/access_token', data={
        'grant_type': 'service_credentials',
        'service_id': 'db0654253d86a9cb7b8dde8d54548ef2d6cf4e12',
        'service_secret': '2df7388773ce88113915e45f524e93c37cf8263f'
    })
    # Parse the JSON content of the response.
    content = json.loads(response.text)
    # Extract the 'access_token' and 'expires_in' values from the JSON content.
    token = content['access_token']
    expires_in = content['expires_in']
    # Store the access token and its expiration in the session.
    Session.put('access_token', token)
    Session.put('expires_in', expires_in)
    # Return the access token.
    return token
```

The service request is managed (handled) by the AM, which is responsible for controlling both the request and the response as presented below.

```
# Import necessary modules and classes.
from typing import Dict, Union
from flask import request, jsonify
from flask_restful import Resource

# Define a class named 'AuthenticateController' that inherits from 'Resource'.
class AuthenticateController(Resource):
    # Constructor for the class.
    def __init__(self):
        pass
    # Define a POST method for handling HTTP POST requests to this resource.
    def post(self):
        # Get the 'serviceID' from the JSON data in the request.
        serviceID = request.json.get("serviceID")
        # Get the 'service' from the JSON data in the request.
        serviceId = request.json.get("service")
        # Get the 'request_url' from the JSON data in the request.
        requestUrl = request.json.get("request_url")
        # Get the 'faculty_name' from the JSON data in the request.
        data = request.json.get("faculty_name")
        # Query the 'Service' model to find a service with the specified 'serviceID'.
        service = Service.query.filter_by(serviceid=serviceID).first()
        # Check if a service with the specified 'serviceID' exists.
        if service:
            # Query the 'Permissions' model to find an endpoint with the specified 'serviceURL'.
            endpoint = Permissions.query.filter_by(endpoint=serviceURL).first()
            # Get the HTTP method allowed for this endpoint.
            method = endpoint.method
            try:
                # Make a request to a specific 'requestUrl' using the specified HTTP 'method' and 'data'.
                response = self.make_request(method, requestUrl, data)
                # Check if the response status code is 200 (OK).
                if response.status_code == 200:
                    # Return a JSON response with the response data and HTTP status code 200.
                    return jsonify(response.json()), 200
```

```
            else:
                # Return a JSON response indicating that the request is not allowed with HTTP status code 405 (Method Not Allowed).
                return jsonify({"message": "not allowed", "code": "405"}), 405
        except BadResponseException as e:
            # Handle the case where the response from the request is not as expected.
            # Return a JSON response with an error message and HTTP status code 405 (Method Not Allowed).
            return jsonify({"message": str(e), "code": "405"}), 405
    else:
        # Handle the case where no permissions are found for the specified serviceID.
        # Return a JSON response indicating that there are no permissions with HTTP status code 405 (Method Not Allowed).
        return jsonify({"message": "No Permissions", "code": "405"}), 405
```

Subsequently, the function that stores the REST request (“storeRest”) for the service Y is shown below.

```
# Define a function named 'store_rest' that takes 'self' (assuming this is a method in a class) and 'request' as parameters.
def store_rest(self, request):
    # Get the 'name' parameter from the input of the 'request'.
    faculty_name = request.input('name')
    # Call the 'request_service_data' function with specific URLs and 'faculty_name' as data.
    response = self.request_service_data("http://smu-upz.dev", "http://smu-upz.dev/addFaculty", faculty_name)
    # Return the 'response' obtained from the 'request_service_data' function.
    return response
```

The function is presented below, related to the fourth request (“d”) of the service.

```
# Define a function named 'dashboard' (possibly part of a class) without any parameters.
def dashboard(self):
    # Call the 'userdata' method of the 'HomeModel' class and store the result in the 'result' variable.
    result = HomeModel.userdata()
    # Construct a URL by appending the 'userid' from the session to the base URL.
    requestUrl = 'http://elibrary.dev/student/' + session.get('userid') + '/books'
    # Make an HTTP request to the constructed 'requestUrl' and store the response data in 'serviceData'.
    serviceData = RestService.requestServiceData('http://elibrary.dev', requestUrl)
    # Initialize a boolean variable 'hasServiceData' as True.
    hasServiceData = True
    # Check if 'serviceData' is a dictionary and if it contains a 'code' key with the value "405".
    if isinstance(serviceData, dict) and serviceData['code'] == "405":
        # If the conditions are met, set 'hasServiceData' to False.
        hasServiceData = False
    # Render an HTML template named 'student/dashboard' with the following data:
    # - 'result': The serialized 'result' data as JSON.
    # - 'serviceData': The data obtained from the HTTP request.
    # - 'hasServiceData': A boolean indicating whether service data is available.
    return render_template('student/dashboard', data=json.dumps(result), serviceData=serviceData, hasServiceData=hasServiceData)
```

Finally, it is shown the function related to the fifth request (“e”) of the service.

```
# Import necessary modules and classes.
from flask import request, render_template, jsonify
from werkzeug.exceptions import BadRequest
from .models import ProfessorModel, Course

# Define a function named 'my_course'.
def my_course():
    # Decode a base64-encoded value from the request's view arguments and store it in 'c_id'.
    c_id = base64.b64decode(request.view_args['c_id'])
    # Attempt to convert 'c_id' to an integer, raising a 'ValueError' if it's not a valid integer.
    try:
        int(c_id)
    except ValueError:
        # If 'c_id' is not a valid integer, raise a BadRequest exception with a message.
        raise BadRequest("Invalid ID")
    # Call the 'get_course_data' method of the 'ProfessorModel' class with 'c_id' and the user's 'userid'.
    response = ProfessorModel.get_course_data(c_id, session.get('userid'))
```

```
    # Check if the 'result' attribute of the 'response' object is truthy.
    if response.result:
        # Query the 'Course' model to find a course with the specified 'course_id'.
        course = Course.query.filter_by(id=request.view_args['course_id']).first()
        # Construct a URL based on the 'course.id' and make a request to it using 'RestService'.
        course_service = RestService.request_service_data('http://smu.dev', f'http://smu.dev/courses/{course.id}')
        # Render an HTML template named 'professor/my-course.html' with the following data:
        # - 'int(c_id)': The integer value of 'c_id' serialized as JSON.
        # - 'course_name': The name of the course.
        # - 'course': The data obtained from the HTTP request.
        return render_template('professor/my-course.html', data=json.dumps(int(c_id)), course_name=course.course_name, course=course_service)
    else:
        # If the 'result' attribute of 'response' is falsy, return a JSON response with the 'msg' attribute and HTTP status code 500 (Internal Server Error).
        return jsonify(response.msg), 500
```

6.6. Evaluation of REST services

6.6.1. Evaluation of REST services based on testing implementation cases

Delays as a consequence of authorization checks can be presented as a critical problem. In case the delay is excessive, then the authorization mechanism can become an obstacle to slow down the whole work of the application (service). Therefore, understanding the delay of how it is being created and measuring it in practice is essential. The total time for processing (T) to access the resource, after identifying the authorization in the authorization mechanism, is calculated as the total time (T1) for processing the service/resource request and T2-time to generate the authorization response from the authorization mechanism: T = T1 + T2.

The time intervals are depicted in Table 6.5. The first stage of the authorization process (in the case when the request for access from service X to service Y is submitted after the verification of the authorization by the authorization mechanism) is presented in T1 as T_XYwithoutAuth/ T_XRSwithoutAuth, while T2 describes the second stage, through which the verification case of authorization for resources' manipulation by service X on resource server is described.

Table 6.5. The time intervals of services and resources' requests and responses processing

T_XYwithoutAuth	The total execution time of requests and responses between service X and service Y and the protocol processing time (e.g. TCP/IP and HTTP) without any additional authorization expenses.

T_XYwithAuth	The total execution time of requests and responses between service X and service Y and the protocol processing time (e.g. TCP/IP and HTTP) with token-based authorization.
T_XRSwithoutAuth	The total execution time of requests and responses between the service X and the RS and the protocol processing time (e.g. TCP/IP and HTTP) without any additional authorization expenses.
T_XRSwithAuth	The total execution time of requests and responses between the service X and the RS and the protocol processing time (e.g. TCP/IP and HTTP) with token-based authorization.
T_TokenGenerate	The execution time of request, verification, generates a token and response between the service/resource server and the AM.
T_TokenVerify	The execution time of request, verification, and response between the service/RS and the AM.

We have developed a test setup to test and check the performance as well as measure the functionality of the authorization management system, including its components, processing, and interoperability between services. On a single machine are located the service X, the service Y, the resource server, and the AM, as part of the test setup. Services are deployed on a single machine to minimize the effects of request and response (network) execution. The test was performed by running services on an I7 quad-core computer with a clock speed of 4.2 GHz, while 4 GB of memory was dedicated for the intention of the setup testing. The REST services presented in the authorization management framework are fully analyzed through the test setup. This means that it has been developed a RESTful API (services) in the Python programming language and created a metamodel example as in Table 6.6.

Table 6.6. Description of RESTful service resources as an illustration of a metamodel for /courses/id

Resource	HTTP Methods
/courses/1	PUT
/courses/1/students	GET, POST, PUT, PATCH
/courses/1/students/1	GET, PUT
/courses/1/students/2	GET, PUT
/courses/1/students/1	DELETE

We have concluded that processing and interaction time does not depend on the volume of resources referenced in a depicted domain. As a result, a few resources are needed to test the authorization management system's performance. The course, as well as the specific parts, has been needed for testing reasons. There are developed requests for specific resources that include various user categories (student or professor). Individual requests are executed a minimum of hundred times. In Figure 6.2 are listed the processing times (in milliseconds) calculated for the minimum, maximum and average times for a request which was observed in the case when it was sent to the authorization mechanism for verification of the authorized access to the specific service/resource. As remarked in the results presented in Figure 6.2, the processing times are relatively small and alter little from each other.

According to the results depicted in Figure 6.2, the average delay, represented by Delta times, is 4.427 ms for service-to-service requests and 4.318 ms for service-to-resource server requests. This can also be verified by taking into account the average execution time of token generation/verification steps, which are almost equal to the total delta times. Additional delays that were caused during the authorization management system processing time were not taken into consideration in the test setup. Furthermore, if the current resource representation should be adapted, the manipulation time may change significantly. If we should adapt the representation of the service resources for humans, then more computations will be required. When we built APIs, we kept in mind the need for easy use, interoperability, and integration for our APIs. The implementation platform was built using the Python programming language, whereas API and JWT specifically, it is used Flask and PyJWT [41]. The Flask micro-framework built on Python; is used to develop REST API services. The PyJWT is a Python library, which allows us to encode and decode JWT.

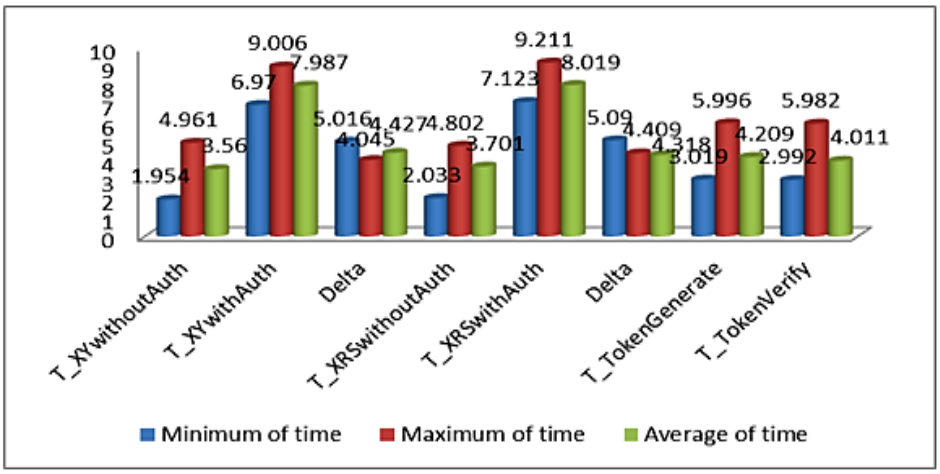

Figure 6.2. Results obtained during testing based on processing times

6.6.2. Evaluation of the proposed framework compared with other approaches

To contrast the proposed framework with other frameworks, let's consider two common authorization frameworks: OAuth 2.0 and JWT. We also consider factors such as security, ease of implementation, and scalability (as shown in Table 6.7). In terms of security, the proposed framework is secure because it uses a centralized authorization mechanism to verify the validity of tokens. This prevents unauthorized access to resources and ensures that only authorized services can access the resources that need. OAuth 2.0 is also secure, but it is more complex than the proposed framework because it involves multiple parties and requires additional steps for authorization. JWT is secure, but it can be vulnerable to attacks if the key used to sign the token is compromised.

In terms of ease of implementation, the proposed framework is straightforward because it only involves a single authorization mechanism. OAuth 2.0 is more complex because it requires multiple parties to be involved in the authorization process. JWTs are also relatively easy to implement, but they require additional steps to ensure the security of the token. In terms of scalability, the proposed framework is highly scalable because it can handle a large number of requests for multiple services. OAuth 2.0 and JWT are also scalable, but they may require additional resources to handle a large number of requests.

Table 6.7. The proposed framework compared with other frameworks (approaches)

Framework/approach	Security	Ease of implementation	Scalability
Proposed framework	High	Easy	High
OAuth 2.0	High	Complex	High
JWT	High	Relatively easy	High

In another evaluation, we consider the traditional approaches for authentication and authorization, such as basic authentication or cookie-based session management (as shown in Table 6.8). In a basic authentication approach, service X would need to send its credentials (e.g., username and password) with each request to service Y or the RS. This approach can be less secure since the credentials are sent in plaintext and can be intercepted. Additionally, managing credentials for multiple services can become cumbersome.

In a cookie-based session management approach, service X would need to maintain a session with service Y or the RS after initially authenticating. This approach can also be less secure since cookies can be intercepted and manipulated. Additionally, maintaining sessions for multiple services can become complex and can lead to issues with scalability. In contrast, the proposed token-based approach (proposed framework) can offer several advantages:

a) *Improved security.* Tokens can be encrypted and signed, making them less vulnerable to interception and manipulation compared to plaintext credentials or cookies.
b) *Scalability.* Tokens can be cached and validated by the AM, reducing the need for repeated authentication requests and improving performance.
c) *Ease of use.* Tokens can be easily exchanged between services and systems, making it easier to manage authentication and authorization for multiple services.

Table 6.8. Evaluation of different approaches for authentication and authorization

Framework/approach	Security	Scalability	Ease of use
Basic authentication	Low	Low	Low
Cookie-based session	Medium	Medium	Medium

management			
Token-based (proposed framework)	High	High	High

Overall, the proposed framework has the advantage of being a simple yet secure authorization mechanism that can be integrated with existing services and is highly scalable. While other approaches have their own advantages and disadvantages, the proposed framework offers a good balance of security and simplicity for most use cases.

6.6.3. Evaluation of REST services in local and cloud server

The primary focus of our study, the proposed framework, is also implemented in the platform based on cloud computing. For this purpose, we have used Amazon Web Services (AWS). It provides a comprehensive range of global services for computing, storage, databases, analytics, applications, and deployment, which enable companies to accelerate their operations, reduce IT expenses, and scale applications (services). The big corporations and most popular start-ups rely on these services to handle a diverse range of tasks such as web and mobile service and application development, data analysis and storage, data archiving, data processing, and more [133].

The response time of the implemented framework has been selected as the basis for evaluating its utility and effectiveness in the cloud environment. Furthermore, Table 6.9 illustrates a comparison between service response time (in milliseconds) in a local server and an AWS cloud server. The values in the table presented below indicates that the response time for individual service request in the AWS cloud server is nearly half of that in the local server.

Table 6.9. A comparison between service response time (in milliseconds) for service requests in a local server and an AWS cloud server

Request	**Source service**	**Destination service**	**HTTP method**	**Action description**	**Response time (in ms) in the local**	**Response time (in ms) in the AWS cloud**

					server	**server**
a	X	Y	GET	Retrieve a collection of programs that are provided for studying.	127 ms	64 ms
b	X	Y	POST	Create a new academic unit (faculty).	377 ms	190 ms
c	X	Y	GET	Retrieve a collection of faculties.	130 ms	60 ms
d	Z	Y	GET	Retrieve a collection of books.	177 ms	88 ms
e	X	Y	GET	Retrieve a collection of students.	157 ms	79 ms

The implemented framework is evaluated in Figure 6.3, which presents a comparison of the response time for service requests between the local server, and the cloud computing server.

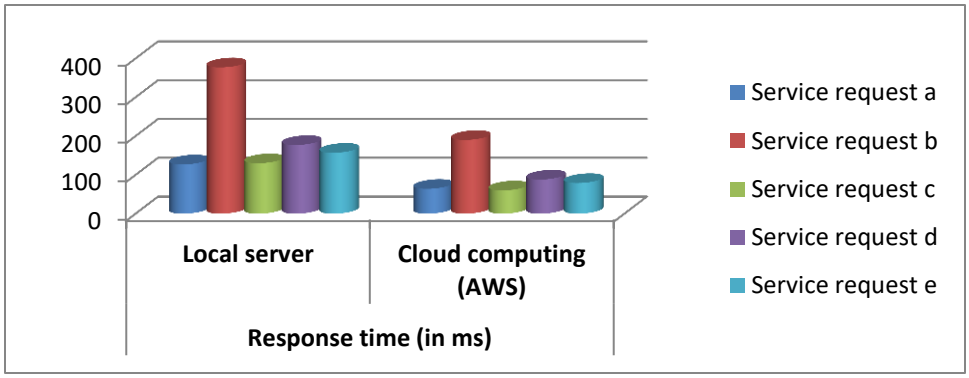

Figure 6.3. Comparison of the response time for service requests in the local server and cloud computing (AWS)

7. CONCLUSION AND FUTURE WORK

REST is an architectural style rooted in SOA that serves as a basis for designing and building contemporary web services (applications). REST embodies a set of features for developing web services that are simple, varied, and provided in a format that is compatible with the web. The stateless nature of REST, which prevents the service from keeping any records of prior requests, is one of its fundamental features. This study has delved into the realm of building distributed RESTful services and resources with a specific focus on addressing authorization issues. The proliferation of distributed systems and the widespread adoption of RESTful architectures have necessitated robust and secure authorization mechanisms. Through our research and implementation endeavors, we have made significant strides in tackling this challenge and providing valuable insights into the development of secure distributed systems.

Our investigation has emphasized the criticality of devising effective authorization mechanisms that align with the fundamental principles of RESTful architecture. By capitalizing on resource-oriented design, statelessness, and the utilization of HTTP methods, we have demonstrated the potential for improved access control and enhanced security within distributed environments. A key finding of our research is the paramount importance of robust authentication and authorization protocols in upholding the integrity and confidentiality of data in distributed systems. Moreover, we have highlighted the necessity for resilient authentication mechanisms, including token-based systems, to verify service identities and mitigate unauthorized access risks.

Furthermore, we have shed light on the challenges associated with managing and enforcing authorization policies in distributed RESTful services. We have explored various approaches, such as RBAC, ABAC, and policy-based access control, in order to establish and enforce granular authorization rules. Our emphasis has been on flexible and scalable authorization frameworks capable of adapting to the dynamic nature of distributed systems and evolving access requirements.

In our implementation, we have leveraged proposed authorization frameworks to provide secure mechanisms for service resource access grants. By integrating identity providers, we have simplified the authentication process and facilitated interoperability among REST-distributed services. Another important contribution of our research lies in the exploration of techniques for managing

service resources and ensuring their availability and consistency. We have examined the utilization of caching to enhance the performance and scalability of RESTful services. Additionally, we have investigated strategies for handling access control and resource consistency in distributed environments.

This research study further expands the functionality and authorization management of RESTful services based on the proposed framework. Service access is verified through the AM if the relevant service has been authorized to perform certain actions or access specific service resources. The authorization verification is based on the token generated by the AM and received by the service for accessing to another service. This enables the customization of the request depending on the service that requests the specific service/resource. It is implemented and demonstrated the functionality of authorizing access to services on the basis of the execution of the REST services developed according to the proposed framework and an elaborate example with HTTP resources and methods. The proposed framework improves the security and reliability of services by ensuring that only authorized services can access the services and resources. This can help to prevent unauthorized access or misuse of the service (resource), which can be especially important for services that handle sensitive or critical data.

Integration, interoperability, and scalability are important in modern computing environments because they allow different systems and services to work together and share data seamlessly. These functionalities are applied successfully in the proposed framework based on the authorization approach. The accessing delay of service to another service when there is and there is no authorization to manipulate resources and service is presented in this work. Additional delays during the implementation of services and their processing time have been almost negligible, and have not been taken into consideration. It can be concluded that processing and interaction time is independent of the volume of resources referenced in a depicted domain. Consequently, a number of resources have been utilized to test the authorization management system’s performance.

It will work in the future to integrate more services and resources into the authorization framework proposed in this study by analyzing in detail large amounts of resources to examine further the aspect of the authorization and handling of large data volumes in integrated services. It is expected the authorization management system to perform well for large amounts of resources in practice.

Furthermore, if the resource representation needs to be adapted, the manipulation time can change significantly. Moreover, if we need to adapt the representation of the service resources for humans, then more computations will be required. Therefore, such issues will also be considered in future work. Future research endeavors can also focus on exploring novel authorization mechanisms, improving the scalability and efficiency of distributed resource management, and addressing emerging security concerns in distributed RESTful environments.

REFERENCES

[1] K. Meridji, K. Al-Sarayreh, A. Abran and S. Trudel, "System Security Requirements: A Framework for Early Identification, Specification and Measurement of Related Software Requirements," *Computer Standards & Interfaces,* vol. 66, no. 103346, pp. 1-20, 2019.

[2] R. T. Fielding, "Architectural Styles and the Design of Network-Based Software Architectures," Univeristy of California, Irvine, California, 2000.

[3] A. Beshiri, A. Mishev and I. Chorbev, "Security Issues in the RESTful API (Service) using OAuth 2.0 for Authentication and Authorization," in *International Conference on Engineering Technologies,* Konya, Turkey, 2021.

[4] M. . G. d. Almeida and E. . D. Canedo, "Authentication and Authorization in Microservices Architecture: A Systematic Literature Review," *Applied Sciences, MDPI,* vol. 12, no. 6, pp. 1-20, 2022.

[5] T. Fredrich, "RESTful Service Best Practices - Recommendations for Creating Web Services," Pearson eCollege, Colorado, 2013.

[6] B. Varanasi and S. Belida, Spring REST - REST and Web Services Development using Spring, Apress, 2015.

[7] T. Erl, Service-Oriented Architecture: Analysis and Design for Services and Microservices, London, United Kingdom: Pearson Education, 2017, pp. 1-416.

[8] C. Pautasso, "RESTful Web Services: Principles, Patterns and Emerging Technologies," in *Web Services Foundations*, New York, Springer, 2014, pp. 31-53.

[9] A. Sunyaev, Internet Computing - Principles of Distributed Systems and Emerging Internet-Based Technologies, 1 ed., Springer, Cham, 2020, pp. 1-426.

[10] K. M. Dhara, M. Dharmala and C. K. Sharma, "A Survey Paper on Service Oriented Architecture Approach and Modern Web Services," Governors State University, Illinois, USA, 2015.

[11] S. Aruna, "Security in Web Services- Issues and Challenges," *International Journal of*

Engineering Research & Technology (IJERT), vol. 5, no. 9, pp. 243-248, 2016.

[12] A. Beshiri, "Resource Description Language for Distributed RESTful Service," *Technology, Education, Management and Informatics (TEM) Journal,* vol. 5, no. 4, pp. 538-549, 2016.

[13] A. Beshiri, "Description of RESTful Service Resources for Authorization Purposes," South East European University, Tetovo, Macedonia, 2016.

[14] A. Beshiri, "Analyzing of REST services: Integration Services and Authorization Management," in *Sixth International Conference for Information Systems and Technology Innovation: Inducing Modern Business Solutions,* Tirana, Albania, 2015.

[15] B. Costa, F. D. Pires, F. C. Delicato and P. Merson, "Evaluating REST Architectures-Approach, Tooling and Guidelines," *The Journal of Systems and Software,* vol. 112, pp. 156-180, 2016.

[16] L. L. Iaconoo and H. V. Nguyen, "Towards Conformance Testing of REST-based Web Services," in *11th International Conference on Web Information Systems and Technologies - WEBIST,* Lisbon, 2015.

[17] P. Giessler, M. Gebhart, D. Sarancin and R. Steinegger, "Best Practices for the Design of RESTful Web Services," in *The Tenth IEEE International Conference on Software Engineering* , 2015.

[18] S. Graf, V. Zholudev and L. Lewandowski, "Hecate, Managing Authorization with RESTful XML," in *ACM*, 2014.

[19] M. Richards, Microservices vs. Service-Oriented Architecture, 1st ed., O'Reilly Media, Inc., 2016.

[20] D. Barry and D. Dick, Web Services, Service-Oriented Architectures, and Cloud Computing, 2nd ed., Burlington, Massachusetts, Massachusetts: Morgan Kaufmann, 2013, pp. 1-248.

[21] V. Rafe and R. Hosseinpouri, "A Security Framework for Developing Service-Oriented Software Architectures," *Security and Communication Networks, Wiley,* vol. 8, p. 2957–2972, 2015.

[22] A. K. Dwivedi and S. K. Rath, "Incorporating Security Features in Service-Oriented Architecture using Security Patterns," *ACM SIGSOFT Software Engineering Notes,* vol. 40, no. 1, pp. 1-6, 2015.

[23] A. Buecker, P. Ashley, M. Borrett, M. Lu, S. Muppidi and N. Readshaw, Understanding SOA

Security Design and Implementation, 2nd ed., Riverton, New York, USA: IBM Redbooks, 2007, pp. 1-502.

[24] A. Miede, N. Nedyalkov, D. Schuller, N. Repp and R. Steinmetz, "Cross-Organizational Security - The Service-Oriented Difference," in *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, vol. 6275, Stockholm, Sweden, Springer, 2009, pp. 72-81.

[25] N. Niknejad, A. R. C. Hussin and I. S. Amiri , "Literature Review of Service-Oriented Architecture (SOA) Adoption Researches and the Related Significant Factors," in *The Impact of Service Oriented Architecture Adoption on Organizations*, Cham, Springer, Cham, Switzerland, 2019, pp. 9-41.

[26] M. I. B and M. A. Shanavas, "Severe SOA Security Threats on SOAP Web Services – A Critical Analysis," *IOSR Journal of Computer Engineering (IOSR-JCE),* vol. 16, no. 2, pp. 135-141, 2014.

[27] S. Alanazi, N. Abdullah, M. Anbar and O. Al-Wesabi, "Evaluation Approaches of Service Oriented Architecture (SOA) - A Survey," in *2nd International Conference on Computer Applications & Information Security (ICCAIS)*, Riyadh, Saudi Arabia, 2019.

[28] G. Kołaczek and J. M. Pietraszko, "Security Framework for Dynamic Service-Oriented IT Systems," *Journal of Information and Telecommunication*, vol. 2, no. 4, pp. 428-448, 2017.

[29] D. Chakroborti and S. S. Nath, "Web Service Performance Enhancement for Portable Devices Modifying SOAP Security Principle," in *20th IEEE International Conference of Computer and Information Technology (ICCIT)*, Dhaka, Bangladesh, 2017.

[30] M. I. Beer and M. F. Hassan , "Adaptive Security Architecture for Protecting RESTful Web Services in Enterprise Computing Environment," *Service Oriented Computing and Applications Journal (Springer)*, vol. 12, pp. 111-121, 2018.

[31] E. A. Setyawan and F. Hidayat, "Web Services Security and Threats: A Systematic Literature Review," in *IEEE International Conference on ICT for Smart Society (ICISS)*, Bandung, Indonesia, 2020.

[32] K. Rajaram and C. Babu, "WS-SM: Web Services - Secured Messaging Framework with Pluggable APIs," in *Computational Intelligence in Data Science*, Switzerland, Springer, 2020,

pp. 233-247.

[33] C. E. Cirnu, C. I. Rotuna, A. V. Vevera and R. Boncea, "Measures to Mitigate Cybersecurity Risks and Vulnerabilities in Service-Oriented Architecture," *Studies in Informatics and Control,* vol. 27, no. 3, pp. 359-368, 2018.

[34] W. Williams, Security for Services Oriented Architectures, 1st ed., Florida: CRC Press Taylor & Francis Group, 2014.

[35] D. Fett, R. Küsters and G. Schmitz, "The Web SSO Standard OpenID Connect: In-depth Formal Security Analysis and Security Guidelines," in *IEEE 30th Computer Security Foundations Symposium (CSF),* Santa Barbara, CA, USA, 2017.

[36] M. . I. Mohamed, M. F. Hassan, S. Safdar and M. Q. Saleem, "Adaptive Security Architectural Model for Protecting Identity Federation in Service Oriented Computing," *Journal of King Saud University - Computer and Information Sciences, Elsevier,* vol. 33, no. 5, pp. 580-592, 2021.

[37] A. Kogan, "Web Services Security - Focus on SAML and XACML," The Open University of Israel, Computer Science Division, 2015.

[38] I. Indu, P. R. Anand and V. Bhaskar, "Identity and Access Management in Cloud Environment: Mechanisms and Challenges," *Engineering Science and Technology, an International Journal, Elsevier,* vol. 21, no. 4, pp. 574-588, 2018.

[39] OASIS, "JSON Profile of XACML 3.0 Version 1.1," OASIS, 2019.

[40] OASIS, "Extensible Access Control Markup Language (XACML) Version 3.0," Organization for the Advancement of Structured Information Standards (OASIS), 2013.

[41] L. . L. Iacono , H. V. Nguyen and P. L. Gorski, "On the Need for a General REST-Security Framework," *Future Internet,* vol. 11, no. 56, pp. 1-33, 2019.

[42] L. Sungchul, J. Ju-Yeon and K. Yoohwan, "Authentication system for stateless RESTful Web service," *Journal of Computational Methods in Sciences and Engineering,* vol. 17, no. 1, pp. 21-34, 2017.

[43] M. Hüffmeyer, F. Haupt, F. Leymann and U. Schreier, "Authorization-Aware HATEOAS," in *8th International Conference on Cloud Computing and Services Science*, Funchal, Madeira, Portugal, 2018.

[44] R. T. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content (RFC 7231)," Internet Engineering Task Force (IETF), 2020.

[45] F. R. Medeiros , "Security and Usability in the HeadREST Language," University of Lisbon, Lisbon, 2020.

[46] I. Al-Rassan, "XML Encryption and Signature for Securing Web Services," *International Journal of Computer Science & Information Technology,* vol. 12, no. 4, pp. 13-26, 2020.

[47] A. Shashwat, D. Kumar and L. Chanana, "An End to End Security Framework for Service Oriented Architecture," in *IEEE International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS)*, Dubai, United Arab Emirates, 2017.

[48] A. Singhal, "Web Services Security: Challenges and Techniques," in *Eighth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'07)*, Bologna, 2007.

[49] C. Rongqiang, Y. Wang, X. Chi and R. He, "Authentication and Authorization for RESTful WEB API in Scientific Computing Environment," in *International Symposium on Grids & Clouds 2019 (ISGC2019)*, Taipei, Taiwan, 2019.

[50] H. Tabrizchi and M. K. Rafsanjani, "A Survey on Security Challenges in Cloud Computing: Issues, Threats, and Solutions," *The Journal of Supercomputing, Springer,* pp. 1-40, 28 February 2020.

[51] M. Hüffmeyer and U. Schreier, "RestACL: An Access Control Language for RESTful Services," in *ABAC '16: Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*, New Orleans, LA, USA, 2016.

[52] I. O. Ayo, O. Ajayi and S. Misra, "Cloud Computing Security: Issues and Developments," in *Proceedings of the World Congress on Engineering*, London, U.K., 2018.

[53] S. Ahmed and Q. Mahmood, "An Authentication Based Scheme for Applications Using JSON Web Token," in *IEEE 22nd International Multitopic Conference (INMIC)*, Islamabad, Pakistan, 2019.

[54] I. Darmawan, R. Gunawan, A. . P. A. Karim, D. Pramesti and A. Rahmatulloh, "JSON Web Token Penetration Testing on Cookie Storage with CSRF Techniques," in *IEEE International*

Conference Advancement in Data Science, E-learning and Information Systems (ICADEIS), Bali, Indonesia, 2021.

[55] M. Haekal and E. , "Token-Based Authentication using JSON Web Token on SIKASIR RESTful Web Service," in *International Conference on Informatics and Computing (ICIC), IEEE*, Mataram, Indonesia, 2016.

[56] X. He and X. Yang, "Authentication and Authorization of End User in Microservice Architecture," *Journal of Physics: Conference Series*, vol. 910, no. 2017, pp. 1-10, 2017.

[57] L. V. Janoky, J. Levendovszky and P. Ekler, "An Analysis on the Revoking Mechanisms for JSON Web Tokens," *International Journal of Distributed Sensor Networks*, vol. 14, no. 9, pp. 1-10, 2018.

[58] A. S. M. Kayes, R. Kalaria, I. H. Sarker, S. Islam, P. A. Watters, A. Ng, M. Hammoudeh, S. Badsha and I. Kumara, "A Survey of Context-Aware Access Control Mechanisms for Cloud and Fog Networks: Taxonomy and Open Research Issues," *Sensors*, vol. 20, no. 9, pp. 1-34, 27 April 2020.

[59] R. E. Sibai, N. Gemayel, J. B. Abdo and J. Demerjian, "A Survey on Access Control Mechanisms for Cloud Computing," *Transactions on Emerging Telecommunications Technologies (ETT), Wiley*, vol. 31, no. 2, pp. 1-21, 15 July 2019.

[60] F. Sifou, A. Kartit and A. Hammouch, "Different Access Control Mechanisms for Data Security in Cloud Computing," in *International Conference on Cloud and Big Data Computing (ICCBDC 2017), ACM*, London, United Kingdom, 2017.

[61] S. Harnal and R. K. Chauhan, "Efficient and Flexible Role-Based Access Control (EF-RBAC) Mechanism for Cloud," *Endorsed Transactions on Scalable Information Systems*, vol. 7, no. 26, pp. 1-10, 18 November 2019.

[62] B. Rashidi, "Authetication Issues for Cloud Applications," in *Authentication Technologies for Cloud Computing, IoT and Big Data*, vol. 9, Y. Alginahi and M. N. Kabir, Eds., Herts, The Institution of Engineering and Technology, 2019, pp. 209-240.

[63] S. H. Chung, J. H. Kim and Y. J. Kim, "Pragmatic Approach Using OAuth Mechanism for IoT Device Authorization in Cloud," in *International Conference on Advances in Computing,*

Communication Control and Networking (ICACCCN), IEEE, Greater Noida (UP), India, 2018.

[64] E. Ferry, J. O. Raw and K. Curran, "Security Evaluation of the OAuth 2.0 Framework," *Information and Computer Security Journal*, vol. 23, no. 1, pp. 73-101, 2015.

[65] J. Richer and A. Sanso, OAuth 2 in Action, Shelter Island, New York: Manning Publications Co, 2017, pp. 1-360.

[66] I. Rauf, "Design and Validation of Stateful Composition RESTful Web Services," Akademi University, Abo, 2014.

[67] I. E. T. F. (IETF), "IETF," 2012. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6749.

[68] A. Neumann, N. Laranjeiro and J. Bernardino, "An Analysis of Public REST Web Service APIs," *IEEE Transactions on Services Computing*, pp. 957-970, 2018.

[69] P. Siriwardena, Advanced API Security: OAuth 2.0 and Beyond, 2nd ed., San Jose, California: Springer-Verlag and Apress , 2020, pp. 1-449.

[70] M. Argyriou, N. Dragoni and A. Spognardi, "Security Flows in OAuth 2.0 Framework: A Case Study," in *International Conference on Computer Safety, Reliability, and Security*, Trento, Italy, 2017.

[71] D. Fett, R. Küsters and G. Schmitz, "A Comprehensive Formal Security Analysis of OAuth 2.0," in *CCS '16: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria, 2016.

[72] R. Yang, G. Li, W. C. Lau, K. Zhang and P. Hu, "Model-based Security Testing: An Empirical Study on OAuth 2.0 Implementations," in *ASIA CCS '16: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, Xi'an, China , 2016.

[73] A. Chaturvedi, "Subset WSDL to Access Subset Service for Analysis," in *IEEE 6th International Conference on Cloud Computing Technology and Science*, Singapore, 2014.

[74] I. Toma, J. . M. García, I. Larizgoitia and D. Fensel, "A Semantically Enabled Service Delivery Platform: An Architectural Overview," in *Handbook of Research on Architectural Trends in Service-Driven Computing*, Hershey, Pennsylvania, Igi-Global, 2014, pp. 181-196.

[75] J. Robie, R. Cavicchio, R. Sinnema and E. Wilde, "RESTful Service Description Language (RSDL): Describing RESTful Services Without Tight Coupling," in *Proceedings of Balisage:*

The Markup Conference 2013, Montréal, Canada, 2013.

[76] A. Beshiri and A. Mishev, "Authentication and Authorisation in Service Oriented Grid Architecture," *International Journal of Grid and Utility Computing,* pp. 1-15, 2020.

[77] B. Wood, B. Watling, Z. Winn, D. Messiha, Q. Mahmoud and A. Azim , "Remote Method Delegation: a Platform for Grid Computing," *Journal of Grid Computing, Springer,* vol. 18, p. 711–725, 2020.

[78] T. . B. Rehman, "Cloud Computing: A Juxtapositioning with Grid Computing," in *IEEE International Conference on Recent Innovations in Signal processing and Embedded Systems (RISE)*, Bhopal, India, 2017.

[79] S. V. Shrihari, K. Pawan, M. Vishnu and S. Mahajan, "Technological Aspects of Grid Computing," in *IEEE 2nd International Conference on Electronics and Communication Systems (ICECS)*, Coimbatore, India, 2015.

[80] B. Krašovec and A. Filipčič , "Enhancing the Grid with Cloud Computing," *Journal of Grid Computing, Springer*, vol. 17, p. 119–135, 2019.

[81] A. Anugurala and A. Chopra, "Securing and Preventing Man in Middle Attack in Grid using Open Pretty Good Privacy (PGP)," in *IEEE Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, Waknaghat, India, 2016.

[82] M. Ibrahim, H. Ibrahim, A. Abdullah and R. Latip, "A High Performance UCON and Semantic-based Authorization Framework for Grid Computing," *Journal of Information and Communication Technology (JICT),* vol. 15, no. 1, pp. 183-202, 2016.

[83] R. Aron, I. Chana and A. Abraham , "A Hyper-Heuristic Approach for Resource Provisioning-based Scheduling in Grid Environment," *The Journal of Supercomputing, Springer*, vol. 71, pp. 1427-1450, 2015.

[84] M. Rebbah, M. E. A. Yemres, M. Khaldi and M. Debakla, "Hybrid Distribution for Association Rules Extraction on Grid Computing," in *International Conference on Image Processing, Production and Computer Science (ICIPCS'2015)*, Istanbul, Turkey, 2015.

[85] M. Singh, "An Overview of Grid Computing," in *IEEE International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, Greater Noida, India, 2015.

[86] J. C. Patni, P. Rastogi, V. K. Jayant and M. S. Aswal, "Methods and Mechanisms of Security in Grid Computing," in *IEEE 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India, 2015.

[87] A. Farouk, A. Abdelhafez and M. Fouad, "Authentication Mechanisms in Grid Computing Environment: Comparative Study," in *IEEE International Conference on Engineering and Technology (ICET)*, Cairo, 2012.

[88] S. Namane and N. Ghoualmi, "Grid and Cloud Computing Security: A Comparative," *International Journal of Computer Networks and Applications (IJCNA)*, vol. 6, no. 1, pp. 1-12, 2019.

[89] W. Qiang and A. Konstantinov, "The Design and Implementation of Standards-Based Grid Single Sign-On Using Federated Identity," in *IEEE 12th International Conference on High Performance Computing and Communications (HPCC)*, Melbourne, VIC, Australia, 2010.

[90] S. Kavecký, "Grid Security and Trust Management Overview," *International Journal of Computer Science Issues (IJCSI)*, vol. 10, no. 3, pp. 225-233, 2013.

[91] S. K. M. Morgan, "Comparative Analysis of Cloud and Grid Computing Paradigms," *Egyptian Computer Science Journal*, vol. 41, no. 3, pp. 53-76, 2017.

[92] W. Jie, J. Arshad and P. Ekin , "Authentication and Authorization Infrastructure for Grids - Issues, Technologies, Trends and Experiences," *The Journal of Supercomputing, Springer*, vol. 52, pp. 82-96, 2009.

[93] S. Namane, "Parallel Access Control Model in Cross-Domain Grid Computing Environment," *International Journal of Embedded and Real-Time Communication Systems (IJERTCS)*, vol. 9, no. 1, pp. 1-16, 2018.

[94] M. S. Alzboon, A. . S. Arif and M. Mahmuddin, "Towards Self-Resource Discovery and Selection Models in Grid Computing," *ARPN Journal of Engineering and Applied Sciences*, vol. 11, no. 10, pp. 6269-6274, 2016.

[95] EGI, "EGI: Advanced Computing for Research," EGI, Amsterdam, Netherlands, 2020.

[96] G. Sipos, G. L. Rocca, D. Scardaci and P. Solagna, "The EGI applications on Demand service," *Future Generation Computer Systems, Elsevier*, vol. 98, pp. 171-179, 2019.

[97] M. S. Mahmoud and Y. Xia, "Cloud Computing," in *Networked Control Systems - Cloud Control and Secure Control*, Butterworth-Heinemann & Elsevier, 2019, pp. 91-125.

[98] S. Basu, A. Bardhan, K. Gupta, P. Saha, M. Pal, M. Bose, K. Basu, S. Chaudhury and P. Sarkar, "Cloud Computing Security Challenges and Solutions - A Survey," in *8th Annual Computing and Communication Workshop and Conference (CCWC), IEEE*, Las Vegas, NV, USA, 2018.

[99] S. Aldossary and W. Allen, "Data Security, Privacy, Availability and Integrity in Cloud Computing: Issues and Current Solutions," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 4, pp. 1-14, 2016.

[100] W. T. Tsai, X. Sun and J. Balasooriya, "Service-Oriented Cloud Computing Architecture," in *Seventh International Conference on Information Technology: New Generations*, *IEEE & ACM*, Las Vegas, NV, USA, 2010.

[101] B. Chaimaa, N. Elkamoun and R. Hilal, "Cloud Computing: Overview and Risk Identification Based on Classification by Type," in *Cloud Computing and Big Data: Technologies, Applications and Security, Springer*, 1 ed., vol. 49, M. Zbakh, M. Essaaidi, P. Manneback and C. Rong, Eds., Rabat, Springer International Publishing, 2019, pp. 19-34.

[102] I. O. Ayo, M. Ananya, F. Agono and R. G. Worlu, "Cloud Computing Architecture: A Critical Analysis," in *18th International Conference on Computational Science and Applications (ICCSA), IEEE*, Melbourne, VIC, Australia, 2018.

[103] M. A. Khan, "A Survey of Security Issues for Cloud Computing," *Journal of Network and Computer Applications, Elsevier*, vol. 71, pp. 11-29, August 2016.

[104] A. Beshiri, "Authentication and Authorization in Service Oriented Cloud Computing Architecture," in *ICT Innovations 2021*, Skopje, North Macedonia, 2021.

[105] B. Chaimaa, E. Najib and H. Rachid, "A Secure Authentication Model for Cloud Federation," *International Journal of Computer Science and Network Security*, vol. 17, no. 10, pp. 89-94, 2017.

[106] M. Hasan, H. Riaz and A. Rahman, "Authentication Techniques in Cloud and Mobile Cloud Computing," *International Journal of Computer Science and Network Security*, vol. 17, no. 11, pp. 28-39, 2017.

[107] S. Y. Lim, M. L. M. Kiah and T. F. Ang, "Security Issues and Future Challenges of Cloud Service Authentication," *Acta Polytechnica Hungarica - Journal of Applied Sciences, IEEE Hungary Section,* vol. 14, no. 2, pp. 69-89, 2017.

[108] Y. A. Younis, K. Kifayat and M. Merabti, "An Access Control Model for Cloud Computing," *Journal of Information Security and Applications, Elsevier,* vol. 19, no. 1, pp. 45-60, 2014.

[109] D. Georgiou and C. Lambrinoudakis, "A Security Policy for Cloud Providers - The Software-as-a-Service Model," in *ICIMP 2014 - The Ninth International Conference on Internet Monitoring and Protection*, Paris, 2014.

[110] M. Babaeizadeh, M. Bakhtiari and A. M. Mohammed, "Authentication Methods in Cloud Computing: A Survey," *Research Journal of Applied Sciences, Engineering and Technology,* vol. 9, no. 8, pp. 655-664, 15 March 2015.

[111] R. Zuccherato, "Challenge-Response Protocol; Identity Verification Protocol," in *Encyclopedia of Cryptography and Security*, Second Edition ed., H. V. Tilborg and S. Jajodia, Eds., Boston, MA, Springer Science and Business Media, 2011, pp. 627-628.

[112] M. H. Rana, H. Riaz and A. Rahaman, "Authentication Techniques in Cloud and Mobile Cloud Computing," *International Journal of Computer Science and Network Security,* vol. 17, no. 11, pp. 28-39, 2017.

[113] M. Darwish, A. Ouda and L. F. Capretz, "A Cloud-Based Secure Authentication (CSA) Protocol Suite for Defense Against Denial of Service (DoS) Attacks," *Journal of Information Security and Applications, Elsevier*, vol. 20, pp. 90-98, February 2015.

[114] J. Richer and A. Sanso, OAuth in Action, I. Glazer, Ed., Shelter Island, New York, New York: Manning Publications Co., 2018, pp. 1-360.

[115] M. Joshi, K. P. Joshi and T. Finin, "Attribute Based Encryption for Secure Access to Cloud Based EHR Systems," in *11th International Conference on Cloud Computing, IEEE*, San Francisco, CA, USA, 2018.

[116] J. Shen, T. Zhou, X. Chen, J. Li and W. Susilo, "Anonymous and Traceable Group Data Sharing in Cloud Computing," *Transactions on Information Forensics and Security, IEEE,* vol. 13, no. 4, pp. 912-925, April 2018.

[117] S. Majumdar, T. Madi, Y. Jarraya, M. Pourzandi, L. Wang and M. Debbabi, "Cloud Security Auditing: Major Approaches and Existing Challenges," in *Foundations and Practice of Security, Springer*, N. Z. Heywood, G. Bonfante, M. Debbabi and J. G. Alfaro, Eds., Montereal, QC: Springer International Publishing, 2019, pp. 61-77.

[118] M. Moghadasi, S. M. Mousavi and G. Fazekas, "Cloud Computing Auditing - Roadmap and Process," *International Journal of Advanced Computer Science and Applications,* vol. 9, no. 12, pp. 467-472, 2018.

[119] K. S. Reddy and M. Balaraju, "Comparative Study on Trustee of Third Party Auditor to Provide Integrity and Security in Cloud Computing," in *International Conference on Processing of Materials, Minerals and Energy, Elsevier*, Ongole, 2018.

[120] B. Balusamy, P. Venkatakrishna, A. Vaidhyanathan, M. Ravikumar and N. D. Munisamy, "Enhanced Security Framework for Data Integrity Using Third-party Auditing in the Cloud System," in *Artificial Intelligence and Evolutionary Algorithms in Engineering Systems, Springer*, 2015.

[121] Y. Ming and Y. Wang, "On the Security of Three Public Auditing Schemes in Cloud Computing," *International Journal of Network Security,* vol. 17, no. 6, pp. 795-802, 2015.

[122] I. E. Ghoubach, B. R. Abbou and F. Mrabti, "A Secure and Efficient Remote Data Auditing Scheme for Cloud Storage," *Journal of King Saud University - Computer and Information Sciences, Elsevier,* pp. 1-7, 2019.

[123] D. Georgiou and C. Lambrinoudakis, "Compatibility of a Security Policy for a Cloud-Based Healthcare System with the EU General Data Protection Regulation (GDPR)," *Information,* vol. 11, no. 12, pp. 1-19, 2020.

[124] B. Russo, L. Valle, G. Bonzagni, D. Locatello, M. Pancaldi and D. Tosi, "Cloud Computing and the New EU General Data Protection Regulation," *IEEE Cloud Computing,* vol. 5, no. 6, pp. 58-68, November/December 2018.

[125] M. Š. Vidović, "EU Data Protection Reform: Challenges for Cloud Computing," *Croatian Yearbook of European Law and Policy,* vol. 12, no. 1, pp. 171-206, December 2016.

[126] B. Duncan, "Can EU General Data Protection Regulation Compliance be Achieved When Using

Cloud Computing?," in *Cloud Computing 2018: The Ninth International Conference on Cloud Computing, GRIDs, and Virtualization*, Barcelona, Spain, 2018.

[127] Z. Georgiopoulou, E. L. Makri and C. Lambrinoudakis, "GDPR Compliance: Proposed Technical and Organizational Measures for Cloud Provider," *Journal of Information and Computer Security*, vol. 28, no. 5, pp. 665-680, 21 April 2020.

[128] L. Elluri and K. P. Joshi, "A Knowledge Representation of Cloud Data Controls for EU GDPR Compliance," in *IEEE World Congress on Services*, San Francisco, CA, USA, 2018.

[129] S. Al-Shammari and A. Al-Yasiri, "Defining a Metric for Measuring QoE of SaaS Cloud Computing," in *15th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNET 2014)*, Liverpool, United Kingdom, 2014.

[130] M. Grinberg, Flask Web Development: Developing Web Applications with Python, 2nd ed., O'Reilly Media, 2018, pp. 1-474.

[131] E. Brachmann, G. Dittmann and K. D. Schubert, "Simplified Authentication and Authorization for RESTful Services in Trusted Environments," in *ESOCC 2012: Service-Oriented and Cloud Computing (Springer)*, Bertinoro, Italy, 2012.

[132] R. Yaduvanshi, S. Mishra, A. . K. Mishra and A. Gupta, "A Security Framework for Service-Oriented Architecture Based on Kerberos," in *International Conference on Computational Intelligence, Communications, and Business Analytics (Springer)*, Kalyani, India, 2019.

[133] S. Hashemipour and M. Ali, "Amazon Web Services (AWS) – An Overview of the On-Demand Cloud Computing Platform," in *Springer International Conference for Emerging Technologies in Computing (iCETiC 2020: Emerging Technologies in Computing)*, London, UK, 2020.

БИОГРАФИЈА

Арбер Бешири е роден во 1990 година во Згатар (Драгаш), Косово. Во 2010 година завршил средно образование во Општата гимназија во СОУ „Ружди Бериша“ во Драгаш. Во меѓувреме, во 2013 година ги завршува дипломските студии по компјутерско инженерство на Факултетот за современи науки и технологии на Универзитетот на Југоисточна Европа во Тетово, Северна Македонија. Во февруари 2016 година ги завршил магистерските студии по компјутерски науки - Софтверско инженерство и телекомуникации на Факултетот за современи науки и технологии при Универзитетот на Југоисточна Европа во Скопје, Северна Македонија. Во 2017 година ги започнува докторските студии по компјутерски науки и инженерство на Факултетот за компјутерски науки и инженерство при Универзитетот „Свети Кирил и Методиј“ во Скопје, Северна Македонија. Арбер моментално работи како редовен асистент, предавач и раководител на наставата на Факултетот за компјутерски науки на Универзитетот „Укшин Хоти“ во Призрен, Косово. Има објавено значителен број трудови на национални и меѓународни конференции и списанија. Некои од неговите публикации поврзани со областа на студии и докторски студии се наведени подолу:

1. A. Beshiri, “Authentication and Authorization Management in SOA with the Focus on RESTful Services,” in *International Journal of Software Engineering and Knowledge Engineering*, vol. 33, no. 8, pp. 1293-1326, 2023. (WoS-JCR IF = 1.47; SJR = 1.31, Q1).
2. A. Beshiri and A. Mishev, "Authentication and Authorisation in Service-Oriented Grid Architecture," in *International Journal of Grid and Utility Computing*, vol.15, no. 5, pp. 422-435, 2023. (WoS IF = 0.5; JCI = 0.15; SJR = 1.036).
3. A. Beshiri, “The Authorization Management Framework for Distributed RESTful Services” in *11th Annual Conference of the Faculty of Computer Science and Engineering*, University “Ss. Cyril and Methodius”, pp. 1-16, 2022.
4. A. Beshiri, "Authentication and Authorization in Service Oriented Cloud Computing Architecture," in *ICT Innovations 2021*, Skopje, North Macedonia, pp. 104-119, 2021.
5. A. Beshiri, A. Mishev and I. Chorbev, "Security Issues in the RESTful API (Service) using OAuth 2.0 for Authentication and Authorization," in *International Conference on Engineering Technologies*, Konya, Turkey, pp. 46-53, 2021.
6. A. Beshiri, “The Flexible RESTful Services and Schemas for Consuming Resources of RESTful Services,” in *8th Annual Conference of the Faculty of Computer Science and Engineering*, University “Ss. Cyril and Methodius”, pp. 1-15, 2019.
7. A. Beshiri, “An Authorization Management Framework for Flexible Distributed RESTful Services,” in *6th Annual Conference of the Faculty of Computer Science and Engineering*, University “Ss. Cyril and Methodius”, 2018.

BIOGRAPHY

Arbër Beshiri has born in 1990 in Zgatar (Dragash), Kosovo. In 2010, he finished high school in the General Gymnasium at the High Secondary School "Ruzhdi Berisha" in Dragash. Meanwhile, in 2013 he completed his bachelor's studies in Computer Engineering at the Faculty of Contemporary Sciences and Technologies at the South East European University in Tetovo, North Macedonia. In February 2016, he completed his master's studies in Computer Science - Software Engineering and Telecommunication at the Faculty of Contemporary Sciences and Technologies at the South East European University in Skopje, North Macedonia. In 2017, he began his doctoral studies in Computer Science and Engineering at the Faculty of Computer Science and Engineering at the University “Saint Cyril and Methodius” in Skopje, North Macedonia. Arbër currently works as a full-time teaching assistant, lecturer, and head of teaching at the Faculty of Computer Science at the University “Ukshin Hoti” in Prizren, Kosovo. He has published a significant number of papers in national and international conferences and journals. Some of his publications related to the field of study and doctoral studies are listed below:

1. A. Beshiri, “Authentication and Authorization Management in SOA with the Focus on RESTful Services,” in *International Journal of Software Engineering and Knowledge Engineering*, vol. 33, no. 8, pp. 1293-1326, 2023. (WoS-JCR IF = 1.47; SJR = 1.31, Q1).
2. A. Beshiri and A. Mishev, "Authentication and Authorisation in Service-Oriented Grid Architecture," in *International Journal of Grid and Utility Computing,* vol.15, no. 5, pp. 422-435, 2023. (WoS IF = 0.5; JCI = 0.15; SJR = 1.036).
3. A. Beshiri, “The Authorization Management Framework for Distributed RESTful Services” in *11th Annual Conference of the Faculty of Computer Science and Engineering*, University “Ss. Cyril and Methodius”, pp. 1-16, 2022.
4. A. Beshiri, "Authentication and Authorization in Service Oriented Cloud Computing Architecture," in *ICT Innovations 2021*, Skopje, North Macedonia, pp. 104-119, 2021.
5. A. Beshiri, A. Mishev and I. Chorbev, "Security Issues in the RESTful API (Service) using OAuth 2.0 for Authentication and Authorization," in *International Conference on Engineering Technologies*, Konya, Turkey, pp. 46-53, 2021.
6. A. Beshiri, “The Flexible RESTful Services and Schemas for Consuming Resources of RESTful Services,” in *8th Annual Conference of the Faculty of Computer Science and Engineering*, University “Ss. Cyril and Methodius”, pp. 1-15, 2019.
7. A. Beshiri, “An Authorization Management Framework for Flexible Distributed RESTful Services,” in *6th Annual Conference of the Faculty of Computer Science and Engineering*, University “Ss. Cyril and Methodius”, 2018.