UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Design and implementation of reference software for MPEG-G genomic information protection

Master Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Èric Monné Mesalles

In partial fulfillment
of the requirements for the master in
Cybersecurity **ENGINEERING**

Advisor: Jaime Delgado Merce
Barcelona, 22th October 2021

Contents

List of Figures

Listings

Acronyms

ABAC Attribute-based access control

ACL Access Control List

AEAD Authenticated Encryption with Associated Data

AES Advanced Encryption Standard

API Application programming interface

BAM Binary Alignment Map

CTR Counter Mode

DNA Deoxyribonucleic acid

GA4GH Global aliance of genomic and health

GCM Gallois Counter Mode

HTTP Hypertext Transfer Protocol

IEFT Internet Engineering Task Force

JSON JavaScript Object Notation

JWT JSON Web Token

KVL Key length Value

MAC Message authentication code

MPEG Moving Picture Experts Group

PAP Policy administration point

PDP Policy decision point

PEP Policy enforcement point

PIP Policy information point

PRP Policy retrival point

RBAC Role-Based Access Control

REST Representational state transfer

RuBAC Rule-Based Acces Control

SAM Sequence Alignment Map

UDL Universitat de Lleida

UPC Universitat Politècnica de Catalunya

URL Uniform resource locator

XACML eXtensible Access Control Markup Language

XML Extensible Markup Language

XSD XML Schema Definition

YAML YAML Ain't Markup Language

Revision history and approval record

Revision	Date	Purpose
0	15/10/2021	Document revision
1	20/10/2021	Document revision
2	22/10/2021	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Èric Monné Mesalles	eric.monne@estudiantat.upc.edu
Jaime Delgado	jaime.delgado@upc.edu

Written by:		Reviewed and approved by:	
Date	22/10/2021	Date	22/10/2021
Name	Èric Monné Mesalles	Name	Jaime Delgado Merce
Position	Project Author	Position	Project Supervisor

Abstract

Genetic data is the most accurate definition of a human being, giving a complete image of a body. It is a very critical information which offers a new field of study, a great opportunity to improve our knowledge in the health arena. This sensitive data can be exchanged between researches and doing it in a secured and protected way is a must.

There are different available standards to help protecting genomic data and enabling it to be securely transmitted and exchanged. Examples include MPEG-G and Crypt4GH.

The project aims to implement a reference software for the MPEG-G standard, with a focus on the security part of the standard. The software is implemented using a microservice architecture, that enables: scalability, improvement and the addition of new functionalities. The resulting service mesh consists on four core services: the authentication service to manage the users, the protection box service to manage the access rules of the resources, the service to encrypt and decrypt the files and the Balana service which acts as a checker of the permissions to the resource.

During the software development will be generated a proposal to insert the Crypt4GH standard inside the MPEG-G. From this process a guide explaining how to install, improve and deploy the reference software will be defined.

1 Introduction

The genome is all the genetic material of an organism [8]. If we focus on human genetics, it is composed of 23 pairs of chromosomes. Counting all of them, we obtain 3200 million pairs of Deoxyribonucleic acid (DNA) bases.

Certain parts of the DNA sequence are traduced to amino acids that will compose the proteins, and these proteins will execute all the functions of the human organism: human development, system correct functioning, among others. In other words, they allow us to be ourselves.

The genetic code is replicated every time a cell is duplicated. The replication machine is not perfect and may sometimes introduce some errors, making each human different from each other. There are some errors which require special attention as they can produce a faulty sequence of the DNA that is translated into a dysfunctional protein which in turn may result in the onset of a disease. By understanding this process details we could tackle or fight against some of these diseases.

The current level of knowledge in the field of biotechnology allows us to carry out a complete reading and subsequent reconstruction of human DNA, enabling to read and process it faster than ever. Thanks to these capabilities, since a few years ago, we have been able to identify a greater number of diseases linked to erroneous DNA mutation. A disease will disrupt the expected functioning of the system and will consequently produce a bad function of the organism.

All these genetic data can be considered as the most accurate, low-level representation of a human being, resulting in critical information which needs to be stored and distributed safely. Any misuse of this information can have serious consequences on multiple levels, be it privacy or even patient health. The amount of genetic data currently available is very large, and in many cases, for various reasons, a single team or institution does not have all the data necessary for its studies. In these situations, the missing data is requested from other teams or institutions. To enable this transmission of critical data, we need some protocol that ensures the security of the transaction and its later storage.

At the time being, to solve part of this security issue, there exist two standards, the MPEG-G and the Crypt4GH. The first one was developed by the working group of Moving Picture Experts Group (MPEG). The second one is defined by the Global aliance of genomic and health (GA4GH). MPEG-G defines a complete standard to solve a group of problems, such as encrypting, storing, sending or compressing the data. Crypt4GH is mainly focused on the encryption of the data.

This thesis is organized into 5 chapters, starting with the current one, used to present the project. This structure aims to facilitate the understanding of the content and the objective we want to approach. Thus, it starts with an introduction of the theoretical concepts, moving towards the project methodology and ending with the results and the conclusions.

The second chapter presents the two standards commented above, MPEG-G and Crypt4GH. They are presented following this guideline: the file structure they use and the cypher ap-

plied in each case. Within this second chapter it is also presented a set of access policies methodologies with particular emphasis on the eXtensible Access Control Markup Language (XACML). This specific focus on the XACML is driven by the fact that is the methodology used in the MPEG-G standard.

The third chapter introduces the methodology used during the development process. The project consists of the implementation of a reference software of the MPEG-G standard. It presents the components and the implemented architecture used to achieve the objective together with some use cases used to test and evaluate the obtained results.

Next, chapter four details the results achieved. It is composed of three subsections, the first one will present a proposal to insert the Crypt4GH standard inside the MPEG-G. Then a second subsection devoted to the reference software with an introduction to it and a later dive in to explain how it works. Hereafter, an installation and first steps guide will be provided to enable the deployment and improvement of the referent software. The last part of the chapter is the test and validation of all the use cases presented in the methodology section. Those use cases are used for testing purposes. If the software passes all of them, it is considered to have worked correctly. Otherwise, it indicates that it needs to be further improved.

The last chapter presents the conclusions of the thesis as well as an analytic overview of the development process realized. It is also shown a brief analysis of the fulfilment or not of the objectives set. Based on the obtained results a line of future line of work is introduced.

1.1 Objectives

This chapter presents the objectives that are the main focus of the project:

- Define and implement a proposal to integrate the Crypt4GH standard inside the MPEG-G.
- Build a reference software of the MPEG-G standard.
- Define a guide on how to deploy, install and improve the reference software

2 State of the Art

This chapter reviews the two core standards which enable medical record sharing and collaboration between scientific teams in an encrypted and secure way. The two standards specified are MPEG-G and Crypt4GH.

On one hand, MPEG-G is a standard that defines how to encrypt one or more than one medical study in a single file. Is defined as a tree structure grouping the medical data [1], the study data [2] and the raw data[3], in an ordered way to enable the random access.

On the other hand, Crypt4GH is more focused to store the raw data of a study, more than one study can be stored on the file, but just the raw data sections one after the other.

The MPEG-G standard is composed of six parts, all parts except part six have a first published version, the sixth is not published yet. The Subchapter of the standard have a main focus on parts one and three, they are the ones that are relevant to the development of this project and to avoid adding extra complexity by explaining unused parts.

Concerning the Crypt4GH side, the cypher suite and the document structure are going to be defined to offer a better understanding of the standard.

After the subchapters of the standards will be introduced the concept of Access control policies. Offering an overview, and explaining the different existing approaches. The eXtensible Access Control Markup Language (XACML) protocol will be the core approach and will be presented more in deep due to is the one used by the MPEG-G standard to define the protection rules.

2.1 MPEG-G

The MPEG-G standard is developed by the Moving Picture Experts Group (MPEG). MPEG is WG11 of ISO/IEC JTC1/SC29, developing standards for media coding, MPEG has been re-structured into several WGs, being WG8 the one in charge of further development of MPEG-G.

This MPEG-G standard is registered under ISO/IEC 23092. It focuses on the storage of genomic data in a hierarchical structure.

The standard presents many benefits [16]. Among these benefits, we can highlight, for example, the functionality of**selective access**, which enables embedded tools to jump directly to the desired information and **data streaming**, which allows the receiver to start reading the data before all the file is transmitted.

Concerning data storage, the standard defines the capability of **genomic study's aggregation**, enabling the storing of multiple genomic data in the same MPEG-G file. This feature is so important because it permits the execution of queries over the file and obtaining data of differents genic records, offering a transversal vision of the information.

[1] Data extracted from any medical area, can contain a group of medical studies
[2] Group of data which composes an entirely medical study.
[3] Primary data, those extracted directly from the source.

The MPEG-G has a strong **enforcement of privacy rules**, the standard contains a specific section for this purpose called *rules*.In this section, a set of rules are stored, within the so-called *protection box*, defining which users can access the file and what actions, permissions or privileges they have over the concerned resource.

From an encryption point of view, it is permitted the **selective encryption of sequencing data and metadata**, which defines a way to encrypt the different file levels of the hierarchy.

Many other benefits are defined, for example, the **interoperability**. That property enables the conversion to other medical data structures as FASTQ, SAM or BAM. All the other benefits can be consulted on the MPEG-G white paper, but are not going to be related because are not used during the project.

To achieve all the explained benefits and so many others, the standard is defined by six parts. Each part is aimed at solving one or more than one property and adding extra functions to the MPEG-G those are:

1. File structure
2. Codification
3. Metadata and protection Box
4. Reference software
5. Conformance
6. Coding of genomic annotations

All the parts, except the sixth, have a stable and published version and are on revision and continuous improvement, but part six is still in development.

2.1.1 Part 1

Part one is registered under the code ISO/IEC 23092-1:2019. This part is focused on defining the document structure for storage and transport.

The storage format is used to validate and consequently save the document on the machine or resource server. The transport structure is dedicated to the sending of the document allowing data streaming.

Concerning the storage format, an overview of how data is structured within a MPEG-G file can be seen in figure 1. The file is organized in a hierarchical structure consisting of the next encapsulation levels: File format, dataset group, dataset, access unit and blocks.

The file format is the first box in the hierarchical level, and must always be placed before any variable-length box [14]. All the boxes need to follow the hierarchical structure represented in the figure 1, so, starting by the dataset group, it is allocated inside the file format parent section.

The dataset group is used to store a collection of datasets of a particular study. But the data is not set as raw data inside the dataset, it is stored on a layer under the dataset called

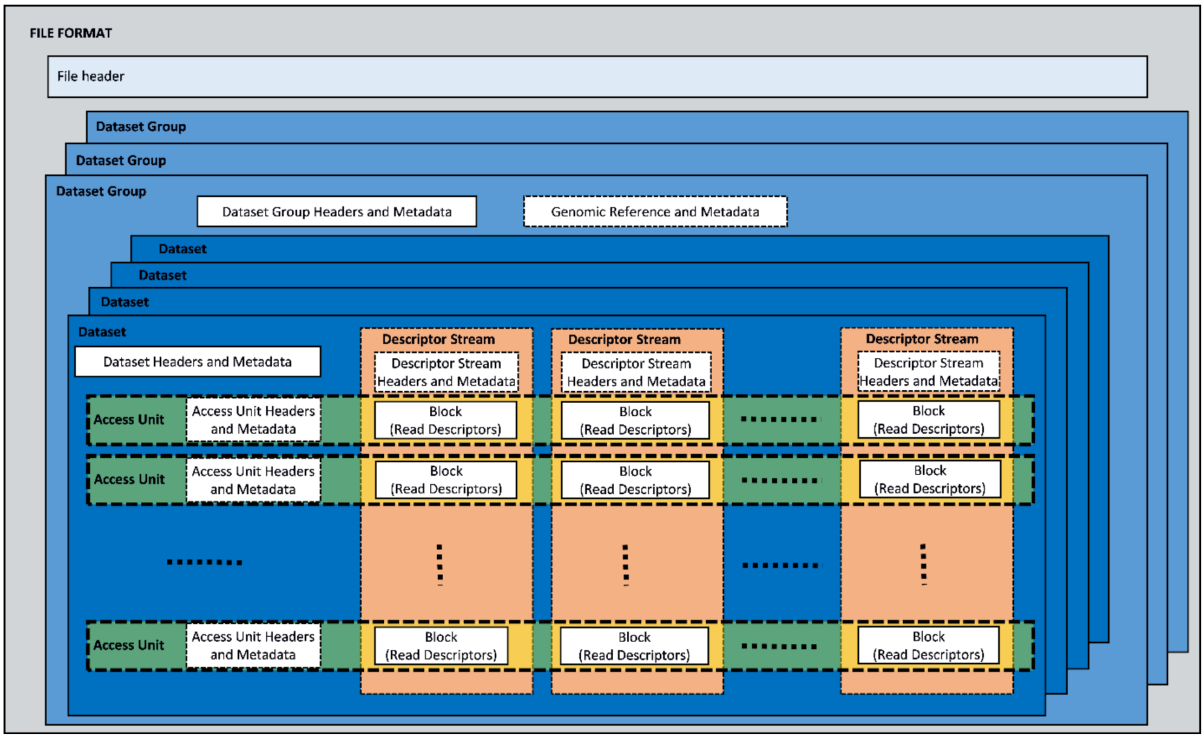

Figure 1: MPEG-G file format [16]

access unit. The access unit is composed of blocks. This block contains the genomic data, and the aggregation of the blocks enables the access and the inspection of the genomic records separately.

The format to transport the document still uses the hierarchical format but with a little variation, as can be seen on the figure 2.

The transport mode encapsulates, inside one or more packets, the MPEG-G file. If we compare the structures, we see that the transport mode deletes encapsulation levels defined on the storage mode.

On the definition of the ISOIEC standard, it is explicitly recommended the decoding of the transport mode started by the *dataset_mapping_table_list* to organize the document as in the storage mode.

To read the data correctly, the encapsulation levels or the type of the file is represented by a Key length Value (KVL) data structure. Is used to advise the reader which data is stored on it. The KVL syntax is defined by the structure presented on listing 1

Listing 1: MPEG-G KLV Structure

```
1 struct gen_info
2 {
3     c(4) Key;
4     u(64) Length;
5     u(8) Value[];
```

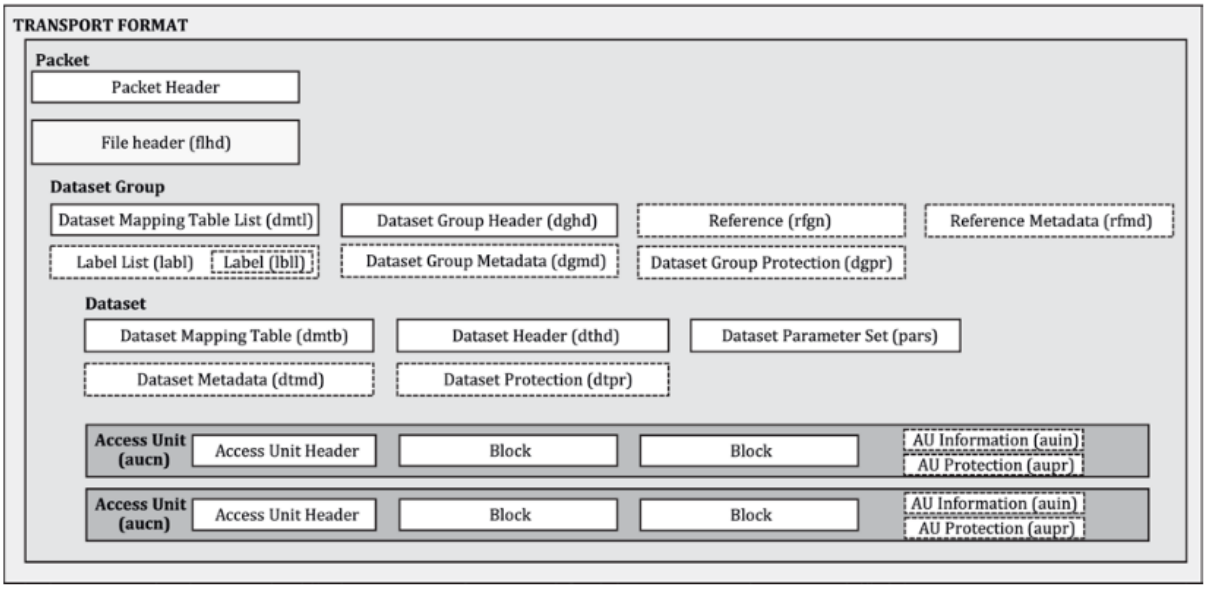

Figure 2: MPEG-G file transport format [14]

6 }

This structure is found in the first bytes of the file. Is used to inform the type of the file, the length and the data. The content type is encoded on **key** field. This field is composed of four chars. The list of possible values can be found on the ISO document. The content is used to mark which kind of file is going to be analized. Also, help to find the hierarchy level on the document.

The **length** is used to set how many bytes can be found on the value field, to parse correctly the document. Is encoded in little-endian. All the files, except the file format, have this variable.

Finally, can be found the **value**, on this field is where goes the data. The data will be encoded in an Extensible Markup Language (XML) file, following the schemas defined by the protocol.

2.1.2 Part 3

As seen before, part one defines how the data is stored and transported. Part three is related to some components inside the file, which is the protection box and metadata. The part three is defined in the standard ISO/IEC 23092:3. This part also defines how to set the backwards compatibly with other formats for example SAM files. The metadata is used to add extra information on every hierarchy level to improve the comprehension of the data.

The protection box is a XML file which defines all the elements related to the security and privacy of the content. This XML stores the required data to decrypt the file, defines a policy section to set the rules and the actions permitted, and also contains the signature

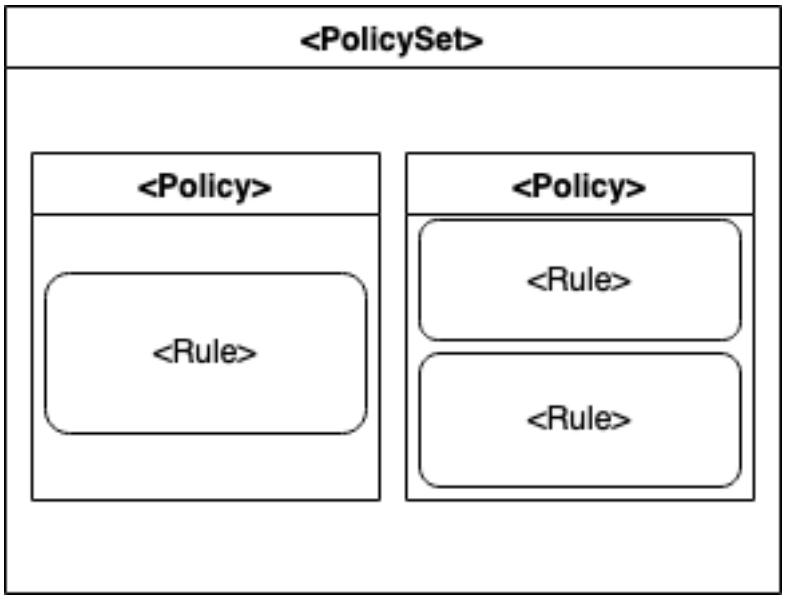

Figure 3: Top-level Policy Elements

section used to implement a digital signature to verify the document integrity and prove that the file was written by a true user.

The policy section is used to store all the rules to define which actions are permitted and which users can perform them. The policies are defined to be very grained rules, for example, a rule can define which role can access the resource, but just the role with determinate email domain, and just from a determinate IP range.

The structure of these polices are defined on the eXtensible Access Control Markup Language (XACML)[7] standard, because is the architecture used to verify the permissions over the resources, the XACML structure is going to be reviewed on next chapters.

The policies, which are encapsulated within a XML file, define a hierarchical structure, composed by three elements (figure 3). :

- `<Rule>` , this tag defines a boolean expression to be evaluated, the result of this expression defines if the users can perform the wanted action.
- `<Policy>` contains the group of rules to be evaluated, is the element used by the XACML system is the one which can be defined as the element which contains the bases of the authorization process.
- `<PolicySet>`, this tag can contain another policy set, or a group of `<Policy>` elements

The signature section stores the values of the digital signature applied over the all the document, and it is used to prove the authenticity of the file and demonstrate that the file was generated by a true user.

Listing 2: XML Signature

```
1  <Signature
2          xmlns="http://www.w3.org/2000/09/xmldsig#">
3          <SignedInfo>
4              <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/
                   REC-xml-c14n-20010315"/>
5              <SignatureMethod Algorithm="http://www.w3.org/2009/xmldsig11#
                   dsa-sha256"/>
6              <Reference URI="">
7                  <Transforms>
8                      <Transform Algorithm="http://www.w3.org/2000/09/xmldsig
                           #enveloped-signature"/>
9                  </Transforms>
10                 <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#
                       sha256"/>
11                 <DigestValue>bjolcVqOKaLWbqfLdV9vfV979ntUQUur4dKLInPuD/Y=</
                       DigestValue>
12             </Reference>
13         </SignedInfo>
14         <SignatureValue>hlhZhm/
               gEYAz0ZUXDp5HRoERhKlgjCZJ7QOuHxcSGaBJ0cqJzsn5YLExR58C0gQ9Gozy7+
               WfIHc=</SignatureValue>
15         <KeyInfo>
16             <X509Data>
17                 <X509SubjectName>CN=Eric Monne,OU=Cybersecurity,O=UPC,L=
                       Barcelona,ST=Barcelona,C=ES</X509SubjectName>
18                 </X509Certificate>
19                   ...
20                 </X509Certificate>
21             </X509Data>
22         </KeyInfo>
23     </Signature>
```

On the code section 2, is presented an example of the signature section. It contains the certificate of the person who signs the document and the fingerprint to certificate the validity.

To check the document integrity, the receiver of the document needs to compute the signature of the full document with the certificate defined on the section.

The last section is the Encryption section, this section stores the values used during the encryption process which are required to decrypt it.

2.2 Crypt4GH

The Crypt4GH protocol is defined as a secure way to store and transport genomic records. Due to the increase of genomic data, the Global aliance of genomic and health (Global

aliance of genomic and health, developed the standard to ensure security during the exchange and storage process.

The Crypt4GH standard defines that the data will be decrypted just during the use, meanwhile the data it is stored on the disk remains encrypted [11]. The standard enables the multi-sharing, this mean that the key used to encrypt the data it is encrypted as many times as people will receive the document, with each person's public key.

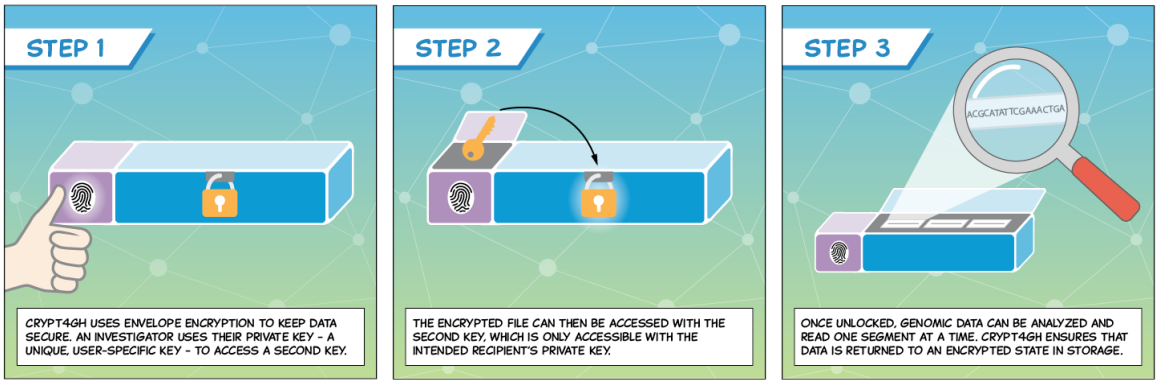

Figure 4: Crypt4GH encryption and decryption. [10]

On the figure 4 it can be seen schematically how a genomic record file is encrypted and decrypted. The file is encrypted using asymmetric encryption. Imagine two researchers, Bob and Alice, both of them own a private-public key pair, and the other one knows the public key of the college. Bob wants to send Alice a particular record of the current study, Bob, will encrypt the genomic record using a data key (K_{data}). The data Key encrypted using the public key of Alice and stored in the headers of the file. Alice will receive the document, decrypt the data key using her private key, and finally decrypt the data.

The data key (K_{data}) is encrypted with a Diffie-Hellman key derived from a shared key (K_{shared}). The definition of the keys will be explained in the following subchapter.

2.2.1 Cipher suite and used keys

To achieve the security mentioned before, the protocol is composed of two groups of keys: an asymmetric key and a symmetric key.

The asymmetric ones are used to enable the exchange between researchers, which on the protocol definition are defined like that [11]:

- Reader secret key (K_{sr})
- Reader public key (K_{pr})
- Writer secret key (K_{sw})
- Writer public key (K_{pw})

On the symmetric side, three types of keys are defined, the first one is **Diffie-Hellman key** (K_{dh}), which will be generated from the derivation of a shared key (K_{shared}), which is the second key. The *shared key* is the key that will encrypt the headers, where the symmetric key is stored to decrypt the data. That K_{shared} can be derived from: K_{sw} and K_{pr} or from K_{sr} and K_{pw}. The last key is the **data key** (K_{data}). The data key is a unique key for each file and is generated by a cryptographically secure random number. This key will be the one used to encrypt the file inside the file, and stored on the headers of the document. The header will be securely encrypted with the shared key.

As the shared key is derived from the key pair of the reader and the writer, this system enables the multiple sharing to be secure, if the writer wants to send the file to X number of people, will generate X number of sections of the header, where inside all of them will be the encrypted data key, using the shared key generated by the pairing of the keys with all the colleges' public data.

The symmetric cypher used is the ChaCha20_ietf_poly1305, this cypher suite is defined by the RFC8439. The Cha-cha cypher suite is a stream cypher, used to encrypt the data, and the Poly1305 is used as the authenticator, both are used as "combined mode" to Authenticated Encryption with Associated Data (AEAD) algorithm. [22].

The ChaCha20 uses a 12B nonce[4] to encrypt the data, and the Message authentication code (MAC) is generated by the Poly1305 Algorithm. As defined in the cryptographic description of the protocol, [11], The cypher ChaCha20_ietf_poly1305 was chosen because enables to encrypt larger messages and with the nonce, it can be encrypted a lot of different messages with the same key, the maximum message able to be encrypted with this cypher is $64 * 2^{32} - 64bytes$, but it is not a problem for this kind of files because it is calculated that the maximum file size is 64Kbytes.

On the asymmetric side was chosen the Elliptic Curve Diffie-Hellman cypher, using the X25519 curve. This cypher was chosen due to the encryption speed, and because it allows the same security level as RSA, but with smaller keys. The generated key is stored along with the public keys and passed through the Blake2b hash function to generate the signature.

The hash function Blake2b [3] is used due to the speed and the higher security. it is faster than SHA-256 and has similar security to SHA-3.

2.2.2 File Structure

A Crypt4GH file is composed of two main classes, inside them, we find groups of other subclasses. These two main classes are the header and the data block. The header is the one that contains the data of the file, with the instructions for decryption and the version of the file. Data block contains: the groups of registries, the MAC of the encryption, the encrypted data and a nonce to be used to achieve semantically secure encryption

On the figure 5 it is shown a graphical representation of the Crypt4GH document. On the top of the image, it is represented the two main classes defined before, the Header and

[4]Number once: Random number that can be used just once

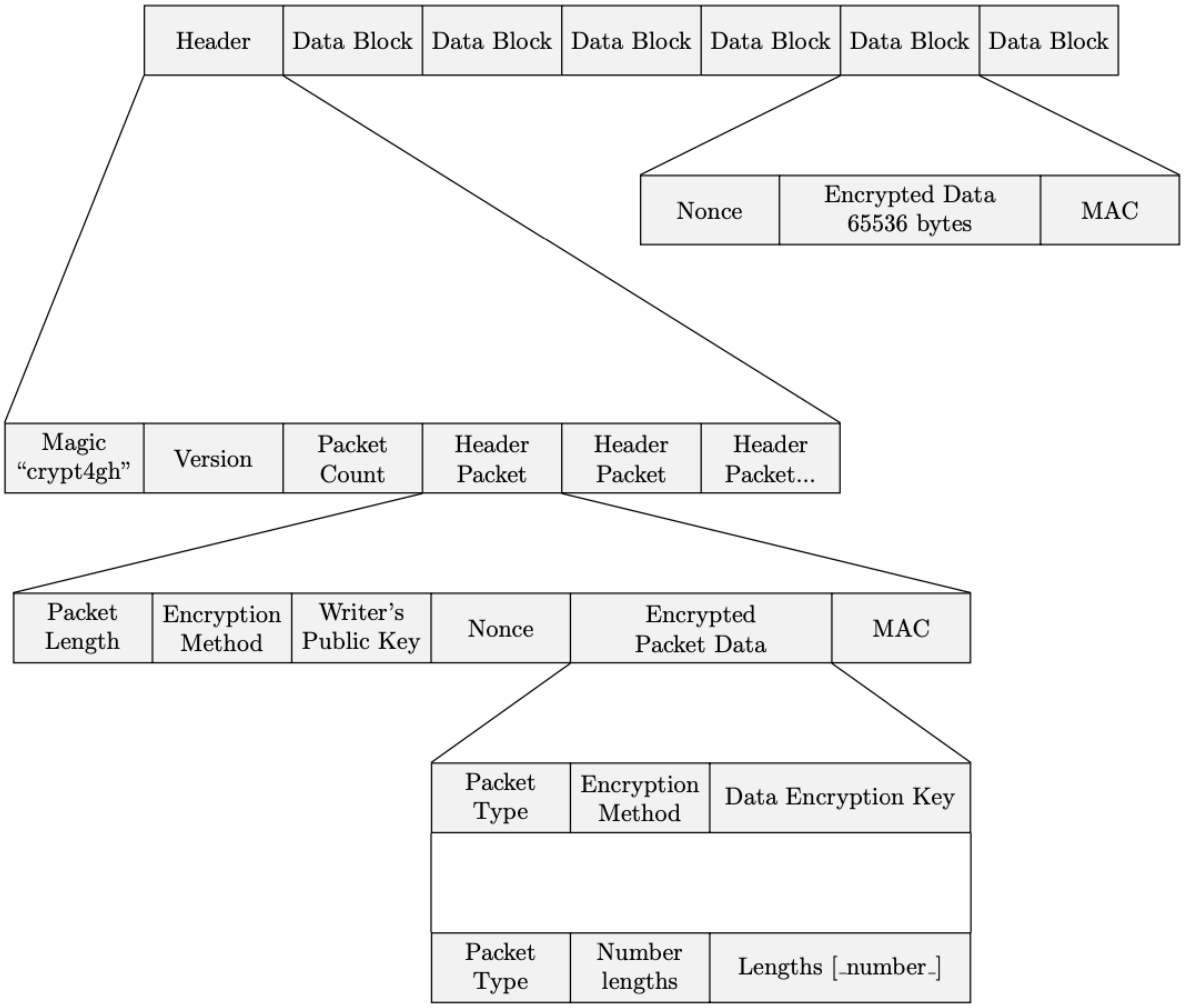

Figure 5: Crypt4GH file structure graphical representation [10]

the data block.

In the header class, we find the following elements and sub-elements in the order stipulated below:

- Magic "Crypt4GH", the string to identify the file type
- The version number
- The packet header counter within which are defined all the header packets which contains the multiple encryptions of the data key, each one for one of the readers with it is shared the document.

In the packer counter, it is found the N numbers of packer headers are in the file. Each packet header consists of the following values:

- Packet length, where it is set the length of the current packet header
- The encryption method, to tell the decrypt method which instructions must follow.
- The writer public key which in conjunction with the reader public key will allow obtaining the data key
- Random nonce to decrypt
- The encrypted data, which to be decrypted is used the K_{data}
- The MAC to verify the encrypted data

2.3 Access Control Policies

The access control policies are mechanisms to determine if a user is allowed to execute some action over another resource. There exist many mechanisms to achieve this kind of protection. All of them have a different approach to solving the problem or are used for different purposes.

The access control policies can be found in many areas for example in a computer system, a user can just access her files or determinate files which other users share with him, but the root user has access over all the system files. This is a hierarchy based access control system. The higher role the user has, the more are the actions that can perform.

There are many access control policies methodologies, the most common ones are [12]:

- **Access Control List (ACL)**, a system based on a list of rules associated to an object. The list contains the users and the actions permitted to a specific resource.
- **Role-Based Access Control (RBAC)**, this access policy approach is not based on rules, is based on roles. Are defined a list of roles, and the actions enabled to each, and the user just can perform determinate action based on the role.
- **Rule-Based Acces Control (RuBAC)**, an approach focused on restricting or able actions based on a list of rules, an example can be, that to access a resource

you must have an IP from Catalonia. All requests that do not validate this rule will be discarded.

- **Attribute-based access control (ABAC)**, this mechanism is based on policies and attributes. Also, defines the architecture and how to process the requests. The attributes to perform the evaluation can be obtained from many parts. For example can be from the user, the environment, or the resource.

The MPEG-G file defines the policies using an ABAC approach, particularly the XACML implementation. To better understand this standard, the following subchapter will get deeper into it. Will be explained the main components. Will be given a brief explanation of them and will be defined the request flow when someone demand permissions to perform an action.

2.3.1 eXtensible Access Control Markup Language

The eXtensible Access Control Markup Language (XACML) protocol is an Attribute-based access control implementation defined to obtain atomic rules to define very accurately who have privileges to execute an action to a resource.

The XACML standard defines how to build the full architecture and its components, on the figure 6, is presented a graphical overview of all the components and how they interact to each other.

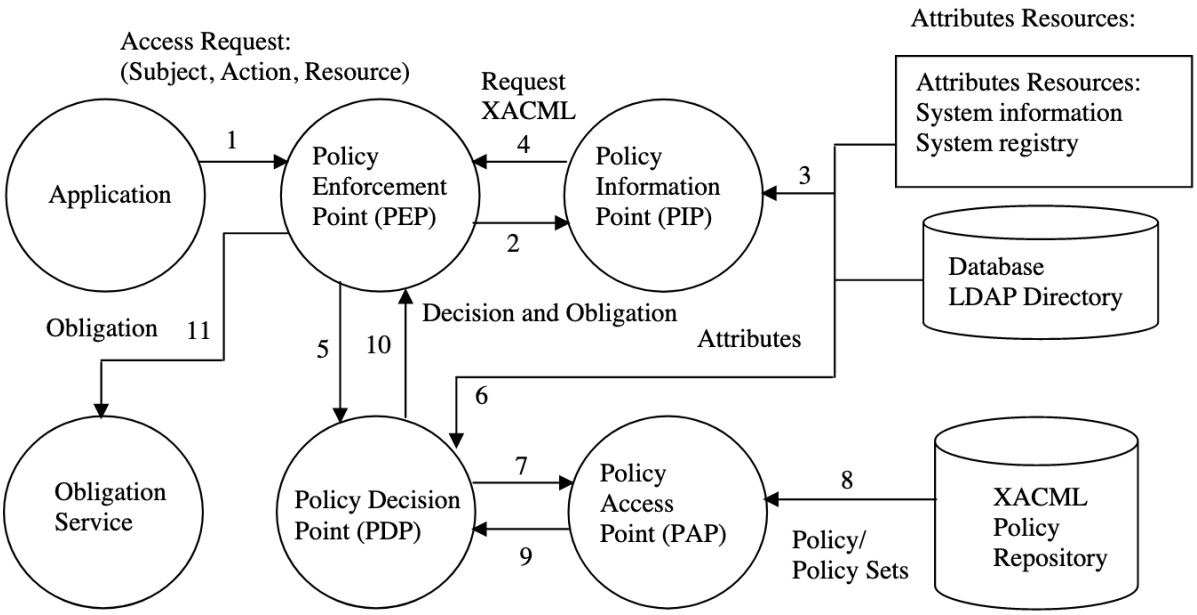

Figure 6: XACML system overview

Figure 6 presents the components involved in the permission request. Not all of them are part of the architecture, some of them are external components that will interact with the system. The ones which belong to the protocol are:

- **Policy enforcement point (PEP)**, it's the point which will receive the request for access to any resource and will derivate it to the PDP, to evaluate if the user has permission to perform the action
- **Policy decision point (PDP)**, this element is the one that will receive the evaluation request of the PEP, and will return if the final user has permissions or not to perform the action. To evaluate the request it will use two other components, the PAP, to evaluate the stored policies, and the PIP to get extra information about the attributes and evaluate more accurately.
- **Policy administration point (PAP)** this point will request to the PRP the access rules and will be the one in charge of managing them. During the verification process, when the PDP requires the access rules to be evaluated, then PAP will be the one that will send it to be verified.
- **Policy information point (PIP)**, will manage the achievement of additional attributes like, for example, some environment data or extra resources to return a more accurate response.
- **Policy retrival point (PRP)**, this point will be the one which will store the XACML policies. Those policies can be on a database or in a file. In this project, those rules will be on the MPEG-G, inside the protection box.

3 Methodology

3.1 Software architecture

The MPEG-G standard is composed of many parts, five of them are already stable and in continuous development, and there is a sixth part that is still in the development stage. Due to this six-part division, the architecture on which this standard is based is microservices oriented, as it allows to add and remove parts without affecting the other components of the system.

The project will be structured in four main components: The **authentication service**, the one which will manage the user login, which will return a JSON Web Token (JWT) to manage the session. Before the authentication barrier, the **file service** will be found. This service will manage the file requests when a user wants access to a file, the file server will ask to *Balana* [5] to check if the user has the right permissions to read or write the file. If the service returns a true, the file server will send a message to **encrypt and decrypt service**. This component is the one that will return the file decrypted. If the user doesn't have permission to access the file, the server will return a *402* to block the user and warn him that he doesn't have the right to access the file.

The last component on the mesh is the **Policy management service**, this component manages the policies, this service is the one that creates, deletes or edits the access rules from a file.

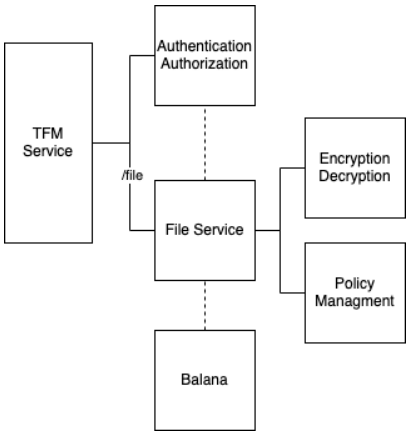

Figure 7: System overview

The microservices approach is used to enable the growth of the reference software, if some developer wants to increase the functionalities by adding new features developed by the working group, it will just need to add the service on the mesh. Also, must adapt the file service to use the new feature, update the service code, adding the requests to the new functionality.

This approach is also used to enable future developers the capability to change any component. In case someone wants to modify only the authentication service, with the mi-

[5] Open source XACML implementation, will be explained in section 3.3

croservices architecture, nothing more needs to be done than to directly replace the service in question. The Balana is also a separate service, for the same reason, if another team prefers another policy verifier, it just needs to change the Balana service.

3.2 Authentication and Authorization

As was defined in the previous subchapter, a component of the service mesh, is the authentication and authorization service. This component takes care of the user login process and the management of the session. To manage the sessions a JWT is used to achieve a RESTful API and help the PDP to obtain data related to the user who is sending a request to the server.

The authentication and authorization is an endpoint that will manage the login and sing up process, the service will manage three scenarios:

- Sign up process, the process to create new users. The service will receive a POST request, with the user email and password, and will generate a new entry on the user's database.
- Login process is the process to enable user access to the server.
- Validate sessions, in this scenario, the service will receive a request with a token on the header and will return if the token is still valid.

The login endpoint will check if the request payload which contains the user email and password which matches with any entry of the database, if a match is found a JWT will be returned. The JWT will encode the next data:

Listing 3: Header of the JWT

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

Listing 4: Payload of the JWT

```
{
  "sub": "mpegg_reference_software",
  "email": <user_email>,
  "iat": <timestamp_issue>,
  "exp": <timestamp_issue + 30days>
}
```

The token must be inserted on the header of any request to the file service. The file server will decode the token and take the user data. Then pass it to the Balana to check if the user can perform the wanted action to the desired resource.

3.3 Access policy management

On the previously defined service mesh, it can be seen a component called Balana, is an XACML open-source implementation of the protocol registered under an Apache2 license. This framework will be the one used to check the access policies.

The eXtensible Access Control Markup Language (XACML) protocol helps with the definition and verification of certain attributes related to the access policy. Helps on the relation between the subject, the resource, the action, and the environment. It enables the definition of the policies on a very low level, and it returns the resolution of an access request. [13]

The protocol doesn't just check if a list of attributes has the access to a determinate resource, but also enables the creation of them. The system management of the rules can be published into an external resource, this permits the modification of the rules from an independent point. These rules can be stored in a database or a file [7]. In the context of the project, it will be in a file, because it will come defined in the MPEG-G file, in the protection box.

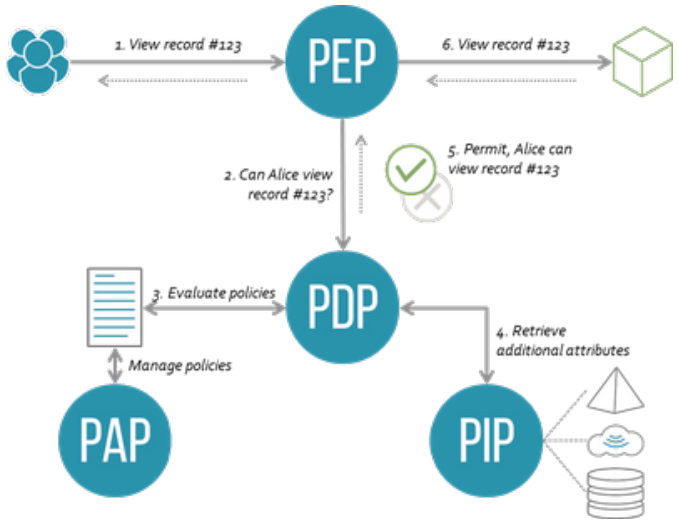

Figure 8: XACML Overview diagram

On the figure 8 it can be seen the flow of a request of a resource over a system that implements access control based on XACML, Balana, as it is an open-source implementation of the XACML follows the same structure (figure 9)

Balana works as the PDP (figure 9), which is the component placed after the PEP whose function is to check if the user can access the resource. On the figure 7, it can be seen that the one which will act as PEP will be a file management service, this one when it receives a read/write request it will ask Balana if the current user has rights to access to the server.

Once the file service receives the request, this one will generate a XACML request, which will be sent to the Balana. The Balana will execute the verification process. To do this

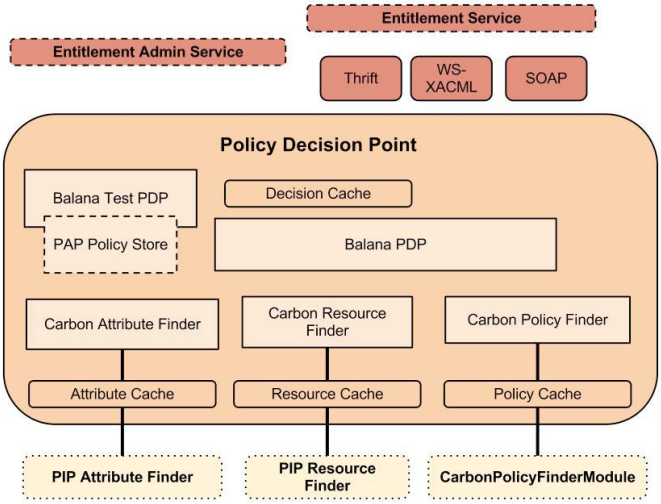

Figure 9: Balana as PDP: overview [35]

task it will read the policies from the protection box of the file and will get the user data from the token, with this data it will process the request, and return the result to the file service.

3.4 Rest API

A REST API, Is a collection of architectural parameters which must be followed during software development. The RESTful services enables the creation of web API's using methods defined on the Hypertext Transfer Protocol (HTTP): GET, POST, PUT, DELETE

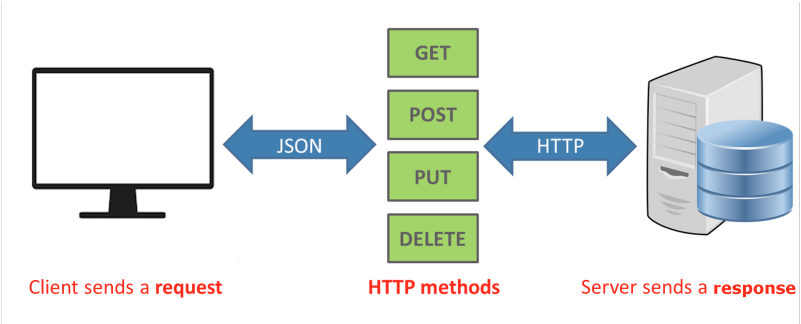

Figure 10: HTTP permitted methods

The web APIs are usually exposed through and URL, and enables an entry point opened to everybody.

To send data to the server, it's used the HTTP packet payload is followed by one of the methods defined to insert data: PUT or POST. The main purpose of the RESTful API is the increase of the performance, scalability, visibility, portability and resistance of a server. [27]

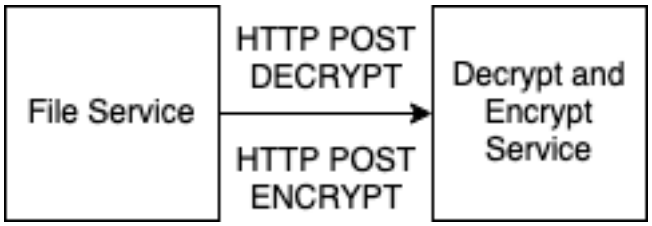

Figure 11: File service using the decryption and encryption service HTTP endpoints

Taking the encryption and decryption service as an example, in figure 11, can be a schematic representation of how the file server takes advantage of the HTTP endpoint exposed by the decrypt and encrypt service to work with the data stored on the file server.

If the decrypt and encrypt service is modified, but not changed the HTTP path, the file server will never know about the change.

This decoupling approach is used on all the composes, and all the services are exposed through an HTTP endpoint.

On the previous subchapter, was defined the PEP and PDP elements. The PEP, as was defined the element which will receive the request, on the project will be the file server, and the Balana, as was defined previously, the PDP. To enable the use of other PDP, the Balana service will be exposed through an endpoint too.

3.5 Use case and RefSW use Environment

In the previous subchapter of this chapter, it was defined the methodology used to develop the software, were defined the architecture, the approach, and the internal components. On this one it will be defined how the software is going to be tested and to do so, there will be defined a series of cases to analyze if the objectives were achieved or not.

The defined cases are the following:

- As a covid researcher, I want to enable my college in another university access to the medical study data securely .
- As a security researcher, I want to verify the signatures of encrypted documents to prevent impersonations.
- As a medical researcher, I want to decrypt and verify documents signatures with no knowledge of cryptography.
- As a researcher from an external institution collaborating with the local research team, I want to access the data from the team.

Those use cases would be used to define some scenarios which the reference software must pass to define the objectives as achieved.

4 Results

Results will be split into three subchapters, The first one will present the purpose to integrate the Crypt4GH protocol inside the MPEG-G standard.

Then will be presented the reference software developed, a section focused on giving a tour over the reference software, after that, it will be explained how to install it, how to deploy it and how to improve it. Finally, there will be found a section called *use case validation*, on this chapter, which will be executed the software over each use case defined in the methodology chapter.

4.1 Integration of Crypt4GH on MPEG-G

Crypt4GH is a cryptographic standard developed by the GA4GH. This standard enables the exchange of genomic data between researchers securely. [10]

The other standard is the MPEG-G. This standard is developed by the Moving Picture Experts Group (MPEG). MPEG is WG11 of ISO/IEC JTC1/SC29, developing standards for media coding, MPEG has been re-structured into several WGs, being WG8 the one in charge of further development of MPEG-G.

4.1.1 File comparison

If we compare the standard's purpose, we find the same objective, share genomic records with a big focus on security, prevent data leaks, and achieve the information exchange between researchers. The main difference is in the file structure and some additional features. MPEG-G, not just able of the encryption, also adds the option to explore the data and execute some search operations over the file. Due to the hierarchical structure, and the metadata.

4.1.2 Ciphers

In our context, a good point to compare is the able cypher suite on both files. We are going to talk about the cypher's that are considered on each standard. Will be compared the cypher suite used for encrypting the genomic records and encrypting the key. MPEG-G, not just able of the encryption, also adds the option to explore the data and execute some search operations over the file. Due to the hierarchical structure, and the metadata.

Genomic Data Encryption, On the MPEG-G side, we find other type of encryption standards, the working group decided to implement the AES standard, using CTR and GCM mode [15]. The key size can be, 128, 192 or 256 bits. For genomic data encryption, both use symmetric encryption algorithms, but they use completely different standards. On the side of Crypt4GH, it's used the ChaCha20-IETF-Poly1305 [11] They affirm that this algorithm was chosen due to the capability to encrypt larger files than other encryption standards.

The ChaCha20-IETF-Poly1305, variant was chosen because having the support of the IEFT. This support grants better library support. Key Encryption After encrypting the

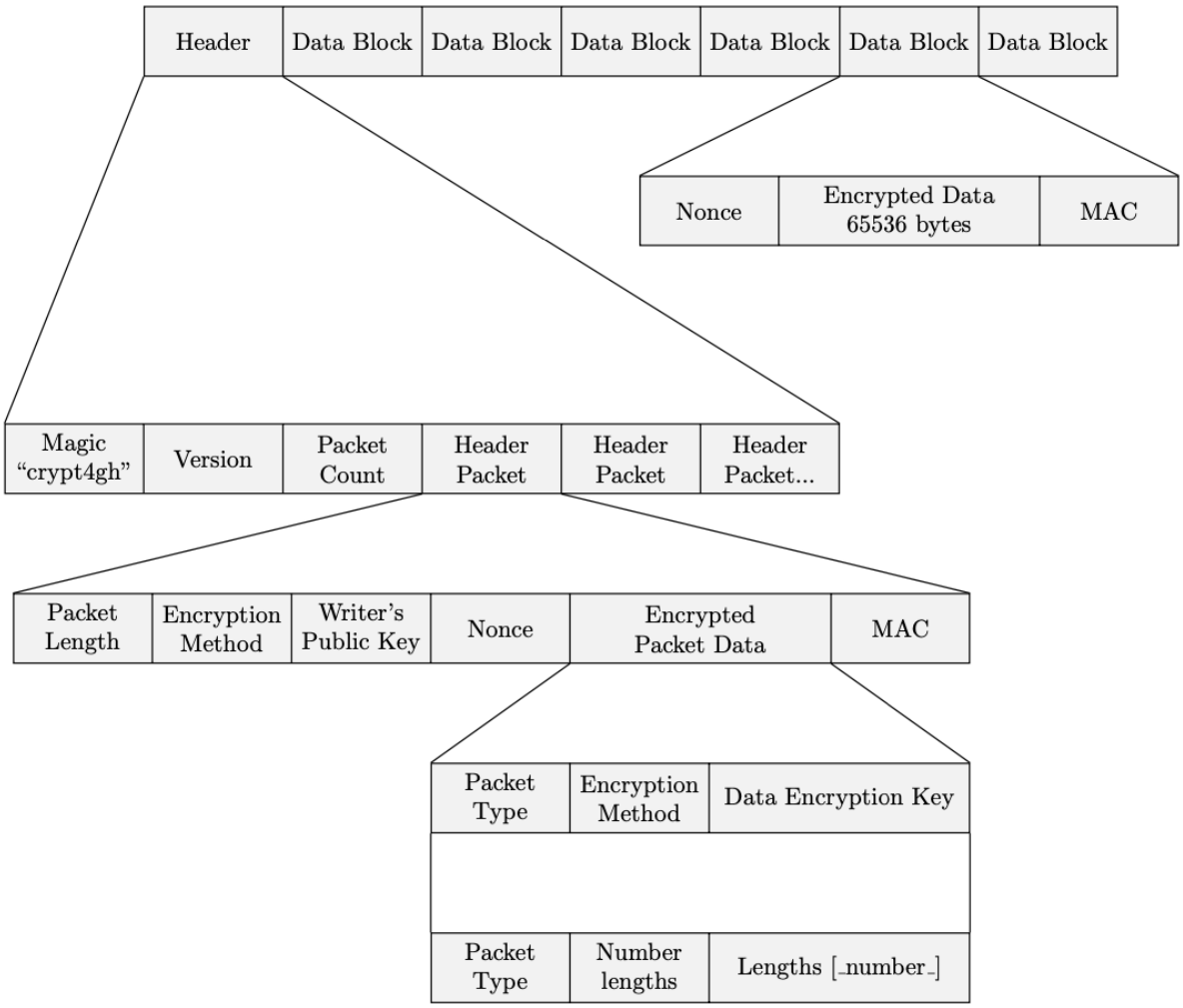

Figure 12: Crypt4gh file structure

genomic data, both of them encrypts the key to share it with the other researchers. MPEG-G uses asymmetric and symmetric encryption to store the key. Both standards able the multi sharing option, which consists in encrypting the key many times with different public keys, to send the file to many researchers or institutions. Crypt4GH uses asymmetric encryption to encrypt the key and send it to many. The ciphered key will store on the header, to encrypt the key it's used Elliptic Curve Diffie-Hellman uses X25519.

The encryption of the key to decrypt the data on the side of MPEG-G it's permitted with two different standards. One is the RFC 3394, these standards define a way to encrypt the key to decrypt the data with symmetric encryption over the key with an encryption key to decrypt it. The other permitted standard is PKCS1 OAEP [19]

4.1.3 Proposal

After the comparison of the type of standards, it is proposed to integrate Crypt4GH inside the MPEG-G standard. Both standards are structured differently, MPEG-G, offers an

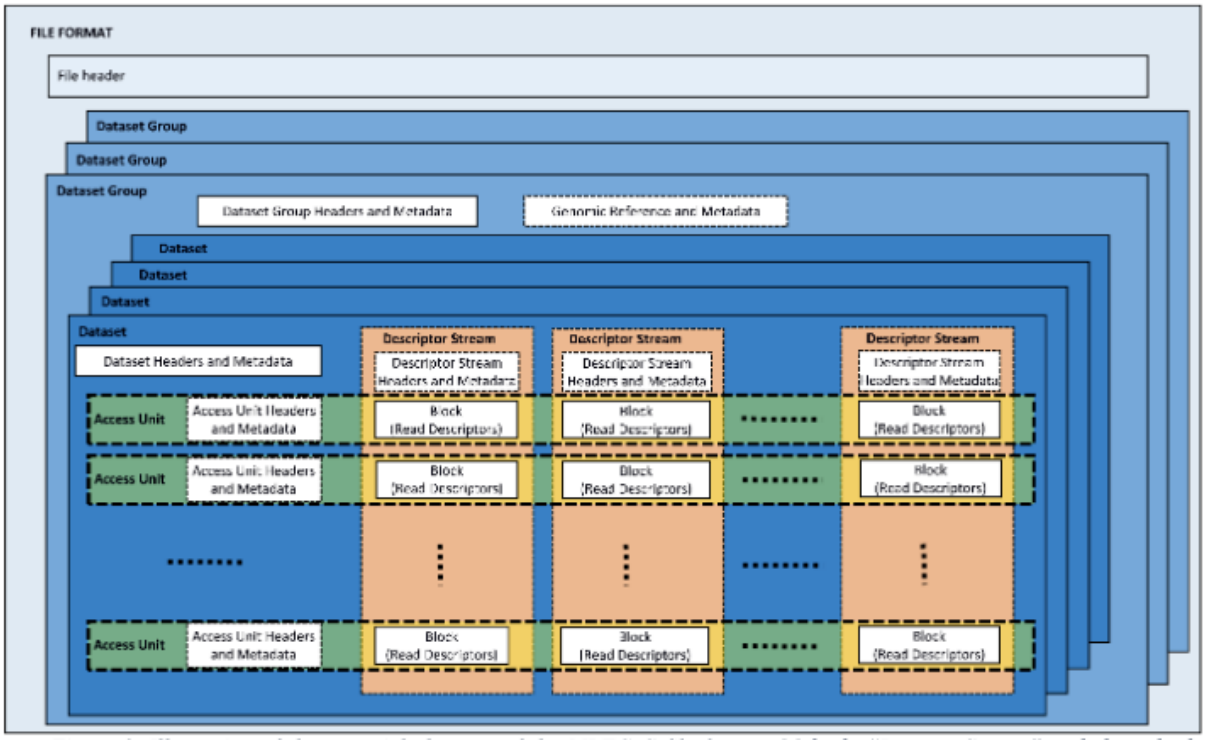

Figure 13: MPEG-G file transport format [14]

option to organize multiple studies on a single file, and Crypt4GH just able the encryption of genomic records. With this difference to the approach of this proposal, it inserts the Crypt4GH on the access unit level, MPEG-G. The access unit layer is the one with the genomic records, so there can be stored the genomic records. The idea builds an XSD adapting the crypt4gh serial structure to an XSD to insert it on the encryption box of MPEG-G. About the difference on the Cipher suits, it's solved just integrating the cyphers of crypt4gh on the MPEG-G standard just for this case.

The modification over the base protection box of an access unit it's the addition of an attribute on the protection type header, to set the file type, which can be Crypt4GH, to advise that the data inside it Crypt4gh standard, or empty to use the standard MPEG-G. The resulting schema can be checked on the annexes.

4.2 Reference Software

This section presents the software developed during the project. The software is built over a microservice architecture which exposes a API through an HTTP gateway. The results will be presented and discussed in the first subchapter. After the presentation and a little tour over it, there will come three subsections which are focused on showing how to install it, how to deploy it and how to improve it. These last three parts will be a little introduction on those topics but extended on the appended git repository.

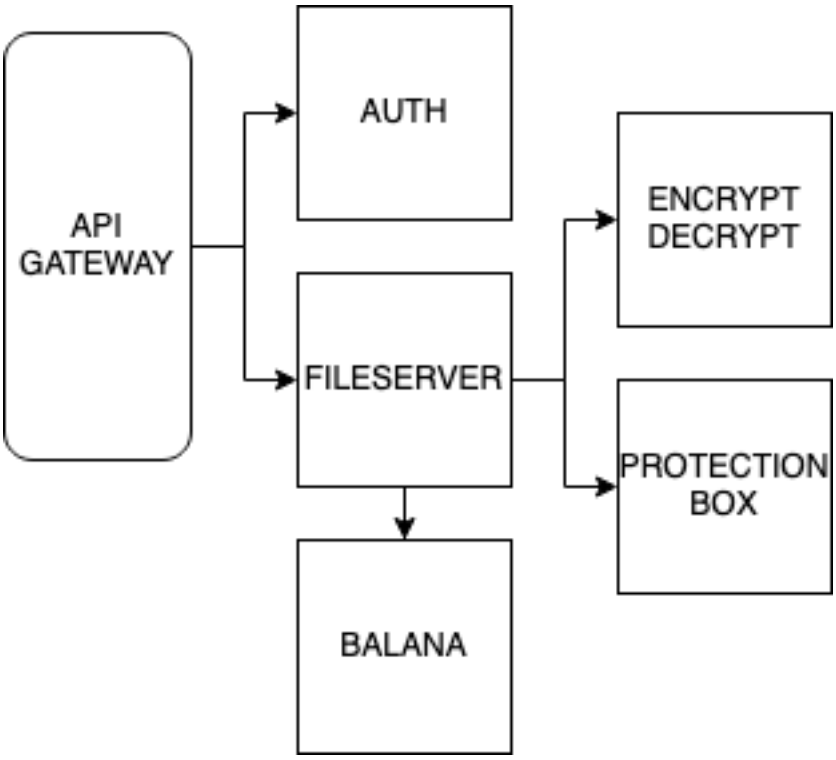

Figure 14: Service Mesh

4.2.1 Overview

The presented software as was commented before is built in over a microservice architecture. On the figure 14, it can be seen the service mesh diagram. The final result is composed of the API gateway, the auth service and the file service. The file service at the same time is composed of to encrypt and decrypt service, the protection box service and the Balana.

All the components and subcomponents operate as unique software, and the internal mesh it is hidden to the final user. The components which compose the architecture are:

API gateway, this element, is the door to the external world. it is exposed by and HTTP endpoint. This element will be the one that will redirect the traffic between the auth service, or the file server. Also, the API gateway will operate as an authentication guard, as all the services need that the user must be logged, every request which goes to the file server, first will be redirected to the auth service, which will return if the user is correctly logged or not.

Authorization and Authentication, on this service will be found the user's management. This service will manage the login and sign up process, also it will be controlling the token generation, and the user data checking. The login and sign up will be done just with the email and the password, if the user sends a valid user and password, this service will return the session token, this token must be inserted on all the requests over the file server due to not existing any public request. Another important task of this service is

the checking of the valid user data, on every request to the file server, the API gateway first will redirect the request to this service. The service will take the token and parse it to analyze and verify if it is correct. If all is correct will return a *status 200* which will advise the API gateway that the request can continue to the file service. But if the token is wrong, will return a *status 401* which will block the flow and return an error to the user.

File service, is the last main component, and it is the main focus of this project. The API gateway is aimed at offering the door to the world and managing the requests, the Authorization and Authentication service will manage the users, and the file server is the one that will manage the files. This server is where it is found the implementation of the MPEG-G protocol, using the subcomponents: Encryption, Decrypt, Balana and protection box, will perform all the operations of part three. The service will receive a valid operation, which will be processed using the submodules. The **Encryption and Decryption** service, will be the one that will manage the operation of encrypting and decrypting, this service will receive a request with the MPEG-G file and will perform the wanted action over it. If the action is decrypted, first will check the signature searching if it is valid, but if the desired action is the encryption of a file, the service will encrypt the file and sign it.

The other submodule is the **protection box**. That service will manage the generation and extraction of data of the protection box of a MPEG-G file. On this service, it can be created a protection box, and it can be deleted or added it new rules. The last one is the Balana, which is the implementation of the Balana framework over a rest API, to open an endpoint to verify if the user has access to the desired resource. The flow of a read operation over an encryption file, will follow the next flow:

1. The request will get into the server through the API gateway, which will send it to the auth server.
2. The auth service will check if the token set on the request header is correct and will return a response to the gateway.
3. If the auth service returns a *401* means that the user is unauthorized. That status will be propagated and returned to the user closing the request flow. But if a *200* is returned, the auth service the gateway will send the request to the file server to be processed.
4. The file server will receive the request and before decrypting the file and sending it to the user, will check if the user had the privileges to act. So will send to the **protection box service** a request to extract the policies from the wanted file.
5. The **protection box service** will receive a MPEG-G file, from which will extract the policies and send it in JSON format back to the file server service.
6. The file server will take the user data from the token, the MPEG-G file and the policies, and will send it to the Balana.
7. The Balana will perform a verification to check if the user has the right to perform the action and send the response to the file server.

8. Again on the file server if the Balana returns a false, the service will return a *401* to the user, but if a true is returned, will take the password and the file and send it to the decryption service.
9. The decryption service will receive a MPEG-G file and the password. Will take that data and perform a decryption operation and return the data decrypted.
10. The file server will receive the data decrypted and return it to the user.

It also exists another service, which is not presented on the mesh because doesn't perform any operation, but it is important to be explained because is the one that contains the API documentation. This service can be found under the path `/docs` and it is a static service that will return the list of endpoints, how to use them and interactive execution. This documentation is built using the Swagger software. This is a software developed by the open API organization focused on defining a standardized way to present Rest API's documentations(figure 15).

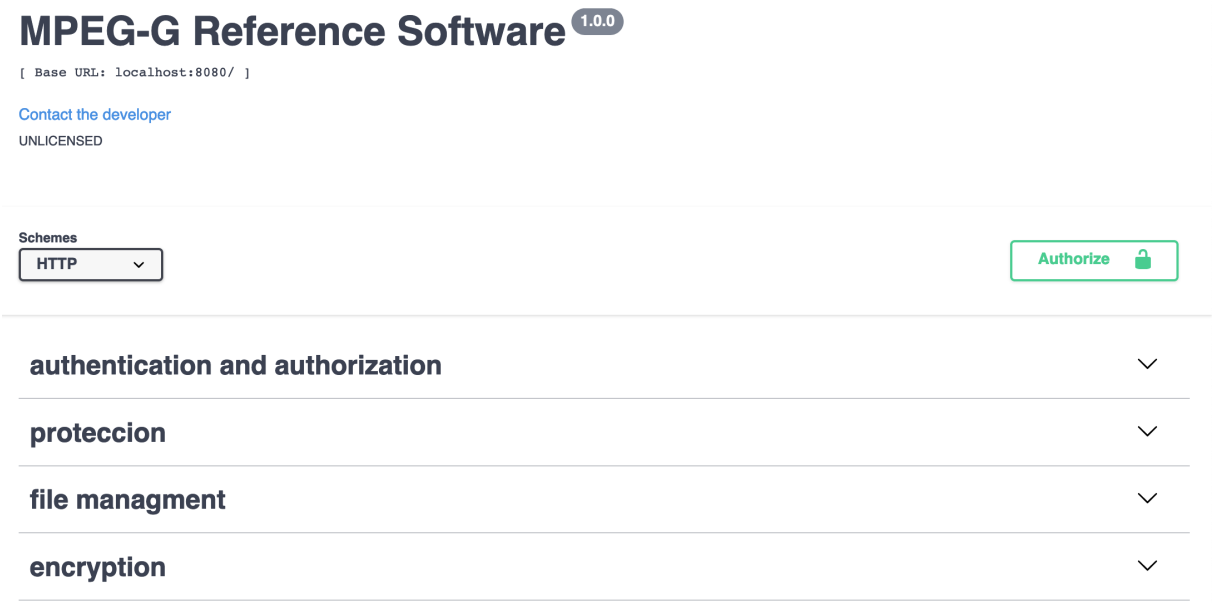

Figure 15: Swagger API documentation screenshot

The endpoints can be grouped into four blocks, two are the main ones, which will be the ones used on a relay deployment, those are the authentication and authorization endpoints, which belongs to the user operations. And the file server block, which is focused on the management of the MPEG-G files, the operations are read and written. Also enables the option to get the file list and the navigation into the files, getting into the subgroups of the MPEG-G files.

Finally, the other two blocks are defined to test the application, those are the encryption and decryption endpoints and the ones from the protection box. These two blocks enable the communication directly to those endpoints bypassing the system guards, which are defined because during the testing process was required direct access to those services and

can be useful to understand the under the layer of the app. It will also be useful to test the use cases defined in the previous chapters.

Now will be presented in a summed up way how to install, deploy and improve the software. The next chapters will be just a presentation and a little introduction on how to do it, the full documentation is on the Annexes, also with the user manual to enable anyone the use that software.

4.2.2 How to install it

As was presented in the methodology chapter, the software is built on top of a microservice architecture. To wake all the service mesh, it was defined and a docker-compose file, which will operate as orchestrator and service name resolver. (figure16)

To install the software is required to have installed the docker-compose and git, the appended document has a special chapter to solve the installation of those components.

To obtain the source code must be cloned from the next GitLab repository: MPEG-G Source code

Figure 16: QR to GitLab repository with the MPEG-G Referent Software

Once the code is cloned to the machine, it is time to deploy it, no extra resources are needed for it, due to all the libraries and programming languages being encapsulated inside the docker containers.

4.2.3 How to deploy it

After the code clone, the next step is to deploy the software. For that, the docker-compose engine is used. The docker-compose is the one that will create a private virtual network inside the machine, to hide the container IP's inside, and enable the network connection between them.

Also, docker-compose will set the service name as the domain name, this helps the developer enable the use of the service name to call another API service, and the engine will manage the redirection to the correct service. To wake up the container mesh with the

docker container, a terminal on the cloned project is need it. Once the terminal is ready, docker-compose can call it to start the process:

Listing 5: Command to execute MPEG-G software reference

```
1 $ docker-compose up
```

That instruction will start the stack. the container will be pulled from their registries, and after that, it will be run. After the waking up process is done, a browser can be opened and search: *localhost:8080* a white page or non-found page will be shown, but that's normal due to the root of the domain being empty, but that means that the reverse proxy is working.

4.2.4 How to improve it

The presented software contains the implementation of parts 1 and 3. But not it is the completed standard, this is because the protocol also contains other parts, and it is still under development. For this reason, the software is licensed under an open-source project to enable anyone to improve, modify and use it.

But as an open-source project, collaboration rules are defined to improve the quality of code, create a wealthy development environment, and keep clean the working tree. Also, the definition of collaborating code of conduct is proposed to encourage participation in the project defining a transparent decision-making process creating a clarifying way on how to participate in the decision making process.

- To add any improvement a pull request must be done
- Any branch can be merged without at least 1 approval
- To add any new service must be created a new repository and just add the container on the main deployment YAML file

4.3 Use case validation

This subchapter will iterate over the uses cases to test the presented reference software. The

- As a covid researcher, I want to enable my college in another university access to the medical study data securely
- As a security researcher, I want to verify the signatures of encrypted documents to prevent impersonations
- As a medical researcher, I want to decrypt and verify documents signatures with no knowledge of cryptography.
- As a researcher from an external institution collaborating with the local research team, I want to access the data from the team.

To execute those scenarios will be used, Postman, software built to test Web API's, all the use cases will be explained and contextualized, after that will be presented the initial data, the expected return and the result.

4.3.1 Use Case 1

The first use case is *As a covid researcher, I want to enable my college in another university access to the medical study data securely* the objective of that use case is to use the software from the point of view of a researcher. The described scenario is composed of a researcher who works in a university, for example, at the UPC. The researcher from UPC collaborates with other researchers in the Universitat de Lleida (UDL). The UPC wants to enable to their colleges from UDL the access to the medical study data.

To build this scenario we need some extra information:

- Researcher 1, from UPC, email: upc.cat
- Researcher 2, from UDL. email: udl.cat

The data generated is stored in an encrypted file, and the protection box associated with this element is the next one:

Listing 6: XML input to the first use case

```
<?xml version="1.0" encoding="UTF-8"?>
<DatasetProtection
    xmlns="mpg-access-prot:DatasetProtectionType">
    <EncryptionParameters configurationID="0" encryptedLocations="0">
      ...
    </EncryptionParameters>
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      ...
    </Signature>

    <Policy>
    <Target>
        <AnyOf>
            <AllOf>
                <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
                    string-regex-match">
                    <AttributeValue DataType="https://www.w3.org/2001/
                        XMLSchema#string">da</AttributeValue>
                    <AttributeDesignator AttributeId="urn:oasis:names:tc:
                        xacml:1.0:resource:resource-id" Category="urn:oasis:
                        names:tc:xacml:3.0:attribute-category:resource"
                        DataType="https://www.w3.org/2001/XMLSchema#string"/
                        >
                </Match>
            </AllOf>
```

```
20          </AnyOf>
21      </Target>
22      <Rule Effect="Permit" RuleId="Rule-1">
23          <Condition>
24              <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
25                  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
                        string-bag">
26                      <AttributeValue DataType="https://www.w3.org/2001/
                            XMLSchema#string">modify</AttributeValue>
27                      <AttributeDesignator AttributeId="urn:oasis:names:tc:
                            xacml:1.0:action:action-id" Category="urn:oasis:
                            names:tc:xacml:1.0:action" DataType="https://www.w3.
                            org/2001/XMLSchema#string"/>
28                  </Apply>
29                  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
                        string-bag">
30                      <AttributeValue DataType="urn:oasis:names:tc:xacml:1.0:
                            data-type:rfc822Name">upc.cat</AttributeValue>
31                      <AttributeDesignator AttributeId="urn:oasis:names:tc:
                            xacml:1.0:subject:subject-id" Category="urn:oasis:
                            names:tc:xacml:1.0:subject-category:access-subject"
                            DataType="urn:oasis:names:tc:xacml:1.0:data-type:
                            rfc822Name"/>
32                  </Apply>
33              </Apply>
34          </Condition>
35      </Rule>
36      <Rule Effect="Deny" RuleId="Deny-Rule"/>
37  </Policy>
38  </DatasetProtection>
```

The previous XML block, shows the input protection box, on the section of the rule can be seen the attribute *Effect="Permit"* this means that if the boolean expression defined inside it is true, the action defined on it will be executed. The action is on the first *Apply* section, the value is *Modify*, and the next section on the field *AttributeValue* appears an email value: upc.cat.

That means any user with UPC email can modify the file. So the researcher with this email also can add a new rule to enable the college to access the document. To do enable this access will be executed a request to generate a new access rule. This request would be sent to the endpoint */policy/add* with the data of the listing 7.

Listing 7: Add policy request

```
1  {
2      "matchValue": "dthd",
3      "ruleId": "Rule-2",
```

```
    "action": "ACCESS",
    "email": "udl.cat",
    "xmlFile": "<?xml version="1.0" encoding="UTF-8" standalone="no"?><
       DatasetProtection xmlns="mpg-access-prot:DatasetProtectionType">
       ... </DatasetProtection>"
}
```

Listing 8: XML output to the first use case

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<DatasetProtection
    xmlns="mpg-access-prot:DatasetProtectionType">
    <EncryptionParameters configurationID="0" encryptedLocations="0">
        ...
    </EncryptionParameters>
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      ...
    </Signature>
    <Policy>
        <Target>
            <AnyOf>
                <AllOf>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
                       string-regex-match">
                        <AttributeValue DataType="https://www.w3.org/2001/
                           XMLSchema#string">dthd</AttributeValue>
                        <AttributeDesignator AttributeId="urn:oasis:names:tc
                           :xacml:1.0:resource:resource-id" Category="urn:
                           oasis:names:tc:xacml:3.0:attribute-category:
                           resource" DataType="https://www.w3.org/2001/
                           XMLSchema#string"/>
                    </Match>
                </AllOf>
            </AnyOf>
        </Target>
        <Rule Effect="Permit" RuleId="Rule-1">
            <Condition>
                <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
                   and">
                    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:
                       function:string-bag">
                        <AttributeValue DataType="https://www.w3.org/2001/
                           XMLSchema#string">modify</AttributeValue>
                        <AttributeDesignator AttributeId="urn:oasis:names:tc
                           :xacml:1.0:action:action-id" Category="urn:oasis
                           :names:tc:xacml:1.0:action" DataType="https://
```

```
                        www.w3.org/2001/XMLSchema#string"/>
27                  </Apply>
28                  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:
                        function:string-bag">
29                      <AttributeValue DataType="urn:oasis:names:tc:xacml
                            :1.0:data-type:rfc822Name">upc.cat</
                            AttributeValue>
30                      <AttributeDesignator AttributeId="urn:oasis:names:tc
                            :xacml:1.0:subject:subject-id" Category="urn:
                            oasis:names:tc:xacml:1.0:subject-category:access
                            -subject" DataType="urn:oasis:names:tc:xacml
                            :1.0:data-type:rfc822Name"/>
31                  </Apply>
32              </Apply>
33          </Condition>
34      </Rule>
35      <Rule Effect="Permit" RuleId="Rule-2">
36          <Condition>
37              <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
                    and">
38                  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:
                        function:string-bag">
39                      <AttributeValue DataType="https://www.w3.org/2001/
                            XMLSchema#string">access</AttributeValue>
40                      <AttributeDesignator AttributeId="urn:oasis:names:tc
                            :xacml:1.0:action:action-id" Category="urn:oasis
                            :names:tc:xacml:1.0:action" DataType="https://
                            www.w3.org/2001/XMLSchema#string"/>
41                  </Apply>
42                  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:
                        function:string-bag">
43                      <AttributeValue DataType="urn:oasis:names:tc:xacml
                            :1.0:data-type:rfc822Name">udl.cat</
                            AttributeValue>
44                      <AttributeDesignator AttributeId="urn:oasis:names:tc
                            :xacml:1.0:subject:subject-id" Category="urn:
                            oasis:names:tc:xacml:1.0:subject-category:access
                            -subject" DataType="urn:oasis:names:tc:xacml
                            :1.0:data-type:rfc822Name"/>
45                  </Apply>
46              </Apply>
47          </Condition>
48      </Rule>
49      <Rule Effect="Deny" RuleId="Deny-Rule"/>
50  </Policy>
```

```
51 </DatasetProtection>
```

On the answer from the server can be seen a new rule with the ID *Rule-2*, which is the id defined on the request. This rule enables the read of any user with an email from udl.cat.

4.3.2 Use Case 2

The second use case to be achieved is *as a security researcher, I want to verify the signatures of encrypted documents to prevent impersonations*, to test this use case, first will be encrypted the string: Use case number two, the returned protection box will be modified the signature value and the server must return an error.

The first step is to encrypt the defined string, to do it will be generated a request to the encryption endpoint, the access unit encryption endpoint will be used */encrypt/accessunit*.

Listing 9: Second use case encryption request

```
1 {
2     "data": "Use case number two"
3 }
```

Listing 10: Second use case encryption response

```
1 {
2     "encryptedData": "sRShpn7Rbgi6D49AUEvxRQ4QWQ==",
3     "protectionBox": "<?xml version=\"1.0\" encoding=\"UTF-8\"?>
4 <AccessUnitProtection
5     xmlns=\"mpg-access-prot:AccessUnitProtectionType\">
6     <AccessUnitEncryptionParameters>
7         ...
8     </AccessUnitEncryptionParameters>
9     <Signature
10         xmlns=\"http://www.w3.org/2000/09/xmldsig#\">
11         <SignedInfo>
12             <CanonicalizationMethod Algorithm=\"http://www.w3.org/TR/2001/
                   REC-xml-c14n-20010315\"/>
13             <SignatureMethod Algorithm=\"http://www.w3.org/2009/xmldsig11#
                   dsa-sha256\"/>
14             <Reference URI=\"\">
15                 <Transforms>
16                     <Transform Algorithm=\"http://www.w3.org/2000/09/
                           xmldsig#enveloped-signature\"/>
17                 </Transforms>
18                 <DigestMethod Algorithm=\"http://www.w3.org/2001/04/xmlenc#
                       sha256\"/>
19                 <DigestValue>U+zvFXYVy0aqX5QJELtX65wopSP2/4m4m5TiBDXbvEw=</
                       DigestValue>
20             </Reference>
```

```
        </SignedInfo>
        <SignatureValue>
          UoLV/qKNG0u3FKdp5ZMoiRCLHZ/ReHK6lnJ72nrbPPUJAIE7/J9aK5k5Sm0
             eSNQxsSDHqTqAi5E=
        </SignatureValue>
        <KeyInfo>
            <X509Data>
                <X509SubjectName>CN=Eric Monne,OU=Cybersecurity,O=UPC,L=
                   Barcelona,ST=Barcelona,C=ES</X509SubjectName>
                <X509Certificate>
                  ...
                </X509Certificate>
            </X509Data>
        </KeyInfo>
    </Signature>
</AccessUnitProtection>",
    "key": "Vz1neOqBAsSRp18q9hNEcA=="
}
```

To verify the signature verification will be taken the encryption response and modify the value of the signature, the server must return an error.

Listing 11: Second use case signature verification request

```
{
    "encryptedData": "sRShpn7Rbgi6D49AUEvxRQ4QWQ==",
    "protectionBox": "<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<AccessUnitProtection
    xmlns=\"mpg-access-prot:AccessUnitProtectionType\">
    <AccessUnitEncryptionParameters>
        ...
    </AccessUnitEncryptionParameters>
    <Signature
        xmlns=\"http://www.w3.org/2000/09/xmldsig#\">
        <SignedInfo>
            <CanonicalizationMethod Algorithm=\"http://www.w3.org/TR/2001/
               REC-xml-c14n-20010315\"/>
            <SignatureMethod Algorithm=\"http://www.w3.org/2009/xmldsig11#
               dsa-sha256\"/>
            <Reference URI=\"\">
                <Transforms>
                    <Transform Algorithm=\"http://www.w3.org/2000/09/
                       xmldsig#enveloped-signature\"/>
                </Transforms>
                <DigestMethod Algorithm=\"http://www.w3.org/2001/04/xmlenc#
                   sha256\"/>
                <DigestValue>U+zvFXYVy0aqX5QJELtX65wopSP2/4m4m5TiBDXbvEw=</
```

```
                    DigestValue>
20              </Reference>
21          </SignedInfo>
22          <SignatureValue>
23            WrongSignatureValue
24          </SignatureValue>
25          <KeyInfo>
26              <X509Data>
27                  <X509SubjectName>CN=Eric Monne,OU=Cybersecurity,O=UPC,L=
                        Barcelona,ST=Barcelona,C=ES</X509SubjectName>
28                  <X509Certificate>
29                    ...
30                  </X509Certificate>
31              </X509Data>
32          </KeyInfo>
33      </Signature>
34  </AccessUnitProtection>",
35      "key": "Vz1neOqBAsSRp18q9hNEcA=="
36  }
```

The figure 17, shows the response of the file with the wrong signature, the response from the server is a status 400, because the data is wrong.

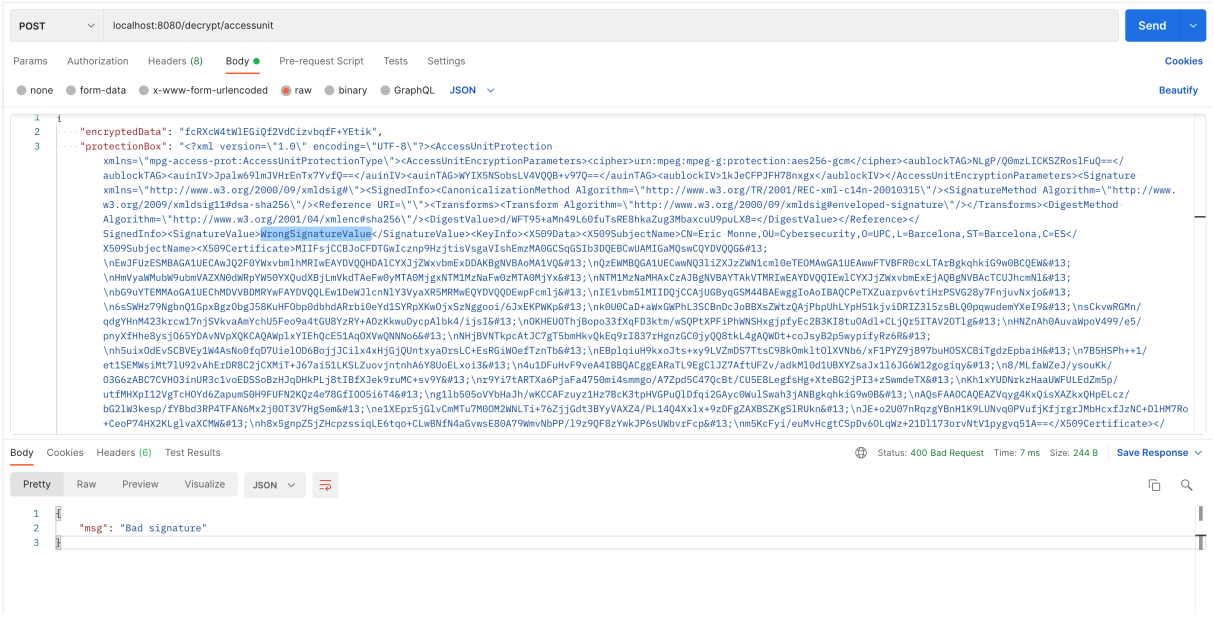

Figure 17: Second use case, signature error response, screenshot from Postman

The verification when a signature is valid will be done on the use case number 3. But with the returned error this use case can be defined as achieved.

4.3.3 Use Case 3

The third of the uses cases is the *As a medical researcher, I want to decrypt and verify documents signatures with no knowledge of cryptography*, the presented use case defines a context where a researcher connects to the system and wants to download a file. The file is stored encrypted on the server, but the users want it decrypted. The server must verify the user data and check if the user accomplishes the rules to read the document. If it is all correct, the server will decrypt the document, and send it back to the user.

To build this scenario will be generated an encrypted file. That file will be a raw string, in a real deployment, the string must be a valid MPEG-G document. The request to encrypt the file is defined on the listing 12.

Listing 12: Request to encrypt data

```
{
    "data": "Encryption Test"
}
```

The data field contains the elements to be encrypted, there goes a full MPEG-G document, but to test the software is sent a random string.

The endpoint called is */encrypt/accessunit*, this endpoint is which will call the file server to encrypt the uploads, but for testing objectives we are going to bypass the file server and send it directly to the encryption service, to test the output.

Listing 13: Response to encrypt data

```
{
    "encryptedData": "tdSvwrvpahj4oKCQZZHl",
    "protectionBox": "
<?xml version="1.0" encoding="UTF-8"?>
<AccessUnitProtection
    xmlns="mpg-access-prot:AccessUnitProtectionType">
    <AccessUnitEncryptionParameters>
        <cipher>urn:mpeg:mpeg-g:protection:aes128-ctr</cipher>
        <auinIV>/2xLPA7KUiREL+P4Kv7zlQ==</auinIV>
        <auinTAG>ayFXtBL30SEfbbibNHoTCw==</auinTAG>
        <aublockIV>TlIi7VuAGWiNoOr9l69diA==</aublockIV>
    </AccessUnitEncryptionParameters>
    <Signature
        xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
            <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/
                REC-xml-c14n-20010315"/>
            <SignatureMethod Algorithm="http://www.w3.org/2009/xmldsig11#
                dsa-sha256"/>
            <Reference URI="">
                <Transforms>
```

```
                    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig
                        #enveloped-signature"/>
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#
                    sha256"/>
                <DigestValue>bjolcVqOKaLWbqfLdV9vfV979ntUQUur4dKLInPuD/Y=</
                    DigestValue>
            </Reference>
        </SignedInfo>
        <SignatureValue>hlhZhm/gEYAz0ZUXDp5HRoERhKlgjCZJ7QOuHxcSGaBJ0
            cqJzsn5YLExR58C0gQ9Gozy7+WfIHc=</SignatureValue>
        <KeyInfo>
            <X509Data>
                <X509SubjectName>CN=Eric Monne,OU=Cybersecurity,O=UPC,L=
                    Barcelona,ST=Barcelona,C=ES</X509SubjectName>
                </X509Certificate>
                  ...
                </X509Certificate>
            </X509Data>
        </KeyInfo>
    </Signature>
</AccessUnitProtection> ",
    "key": "L6P5wwoc7T5WdIc8a/SMzQ=="
}
```

As the key was not sent to the server, the service creates and returns a random encryption key. This random key will be stored by the file server. Also, the server returns a field called *encryptedData* where goes the data is encrypted. The data returned is the protection box, where comes the data signature.

The focus of this use case is the decryption testing, so the returned protection box and the key will be sent to the decryption service:

Listing 14: Decryption request

```
{
    "encryptedData": "fcRXcW4tWlEGiQf2VdCizvbqfF+YEtik",
    "protectionBox": "<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-8"?>
<AccessUnitProtection
    xmlns="mpg-access-prot:AccessUnitProtectionType">
    <AccessUnitEncryptionParameters>
        <cipher>urn:mpeg:mpeg-g:protection:aes128-ctr</cipher>
        <auinIV>/2xLPA7KUiREL+P4Kv7zlQ==</auinIV>
        <auinTAG>ayFXtBL30SEfbbibNHoTCw==</auinTAG>
        <aublockIV>TlIi7VuAGWiNoOr9l69diA==</aublockIV>
    </AccessUnitEncryptionParameters>
```

```
    <Signature
        xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
            <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/
                REC-xml-c14n-20010315"/>
            <SignatureMethod Algorithm="http://www.w3.org/2009/xmldsig11#
                dsa-sha256"/>
            <Reference URI="">
                <Transforms>
                    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig
                        #enveloped-signature"/>
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#
                    sha256"/>
                <DigestValue>bjolcVqOKaLWbqfLdV9vfV979ntUQUur4dKLInPuD/Y=</
                    DigestValue>
            </Reference>
        </SignedInfo>
        <SignatureValue>hlhZhm/gEYAz0ZUXDp5HRoERhKlgjCZJ7QOuHxcSGaBJ0
            cqJzsn5YLExR58C0gQ9Gozy7+WfIHc=</SignatureValue>
        <KeyInfo>
            <X509Data>
                <X509SubjectName>CN=Eric Monne,OU=Cybersecurity,O=UPC,L=
                    Barcelona,ST=Barcelona,C=ES</X509SubjectName>
                </X509Certificate>
                  ...
                </X509Certificate>
            </X509Data>
        </KeyInfo>
    </Signature>
</AccessUnitProtection>",
    "key": "tBTQQ463tq2/MUdXJ3pS48xpBXfc0eLk1zIY9K/n2wo="
}
```

The Server verifies the Signature of the element and if it is correct returns the decrypted data.

Listing 15: Result of the third use case

```
{
    "decryptedData": "Encryption Test"
}
```

The returned decrypted data is the one that was encrypted before, so this use case can be set as achieved.

4.3.4 Use Case 4

The last use case is the number four: *As a researcher from an external institution collaborating with the local research team, I want to access the data from the team.*, this use case defines a scenario where a researcher from an external organization wants to access the data from another university, can be defined as the continuation of the first use case.

So if in the first use case, the UPC researchers enable the access to the file to a UDL researcher. Now the UDL researcher will get into the data. Before the request goes directly to the policy management service, now the request will go through the file server. The first request will be to read the data, and the server must return the decrypted data. Then will be a request to add a policy and the server must return an error because the user doesn't have the privileges to write on the file.

Listing 16: Value of JWT to access the server

```
{
  "sub": "mpegg_reference_software",
  "email": "udl.cat"
}
```

The listing 16 shows the JWT send it to the server, on it can be seen the email of the user required to verify if the user has access to the wanted resource with the wanted action.

Listing 17: Fourth use case read response

```
{
  "data": "Use case number one"
}
```

The response contains the encrypted string on use case number 1, so the server grants access to the resource. The request tries to add a new policy, the server must return an error because a user with UDL email can not edit the file.

On the figure 18 can be seen that the server returns a status *402*, because the user can't modify the file. So with this screen and the previous response, this use case can be defined as passing and validating all the use cases.

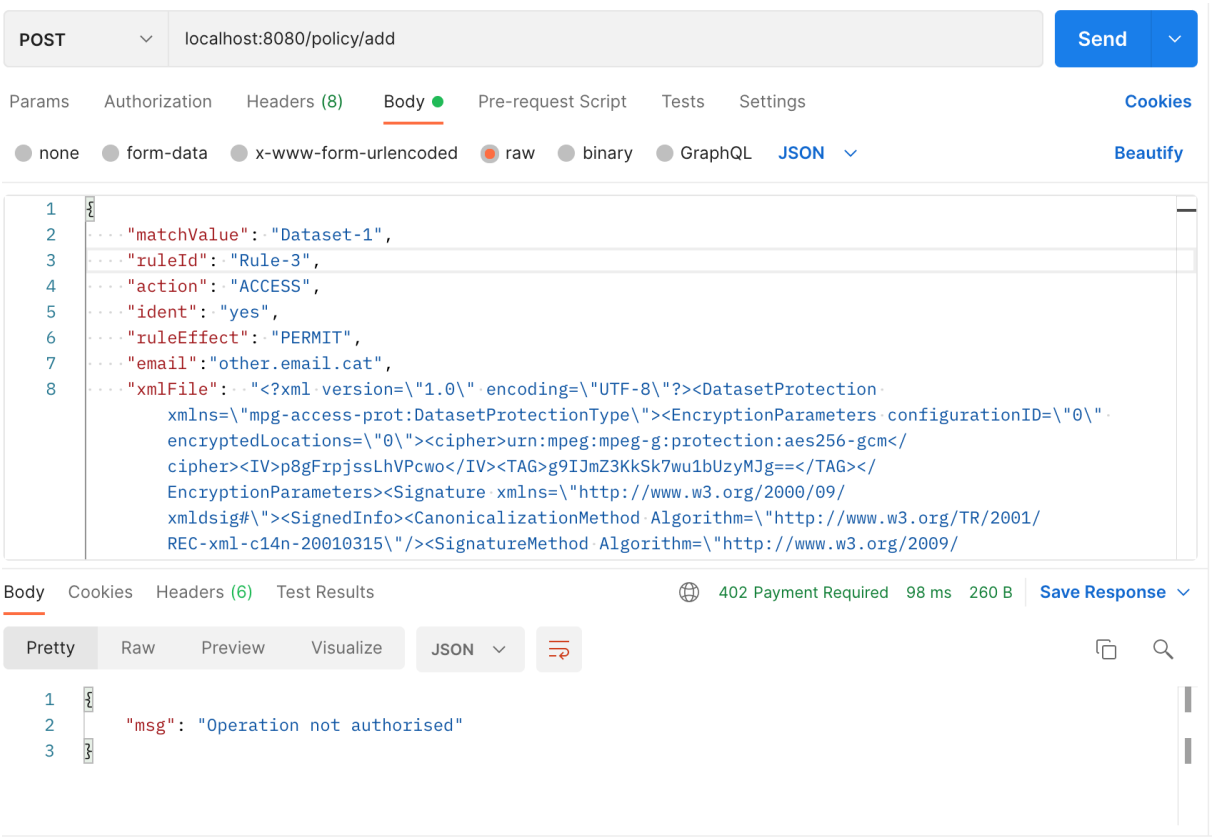

Figure 18: Fourth use case, error: not authorised action response, screenshot from Postman

5 Conclusions and Future Work

This chapter presents the conclusions of the project resulting from studying the MPEG-G and Crypt4GH standards, and the consequent implementation of the reference software. An overview presenting the results and the content generated is done. From the work presented so far and as a closing step, some possible future lines of work will be introduced.

The first objective of the project was to define a proposal for inserting Crypt4GH in MPEG-G. In the search to achieve this objective and based on the proposal made, we obtain the following conclusion: MPEG-G standard enables the classification in the hierarchical level of a medical record, permitting the navigation between levels. This enables the search of information in an ordered way. If we look at the encryption differences, the MPEG-G standard encrypts every level of the hierarchy, meanwhile, the Crypt4GH standard just encrypts the raw data without any hierarchy. Due to that difference in the storing format, the presented proposal defines an approach where the Crypt4GH can be inserted on the access unit level of the MPEG-G standard as an encryption method.

The second objective was to implement a software reference for the MPEG-G standard. The software has been validated against four use cases presented in the chapter 3.5. Every use case defines a scenario where the MPEG-G standard can be used. The use cases were reviewed one by one and all successfully pass the tests. So, we can consider that the second objective was also achieved.

In order for the reference software obtained from the second objective to be considered as such, a guide needs to be included in conjunction with the software implementation. The last objective of this thesis sought to meet this requirement and that is why we have defined a set of guidelines, making future collaboration on the project and adding new features more accessible. To achieve this objective a guide has been created and deployed on a GitLab page. With this user interface page being accessible we can consider the objective as achieved.

At the time of writing these conclusions, the presented software correctly implements parts one and three of the MPEG-G standard. As we have stated in the first chapters of this thesis, the standard is composed of a total of six parts which means that a future line of work could be to include the remaining parts into our software. Pointing in this direction and taking advantage of the microservices architecture used, adding new parts should be as direct as adding new microservices to the service mesh.

Another issue that could be really interesting to implement in the near future and that would be really helpful to increase the robustness of our system is to increment the number of rules that a policy service can create (being, at the present time, email access the only policy available).

Finally, like any software, it is important to keep in mind that a bug revision should be periodically or continuously applied while new features are being added, thus making the software less error-prone.

References

[1] Claudio Albert, Tom Paridaens, Jan Voges, Daniel Naro, Junaid J. Ahmad, Massimo Ravasi, Daniele Renzi, Giorgio Zoia, Paolo Ribeca, Idoia Ochoa, Marco Mattavelli, Jaime Delgado, and Mikel Hernaez. An introduction to MPEG-G, the new ISO standard for genomic information representation. Preprint, bioRxiv, September 2018.

[2] API Documentation & Design Tools for Teams — Swagger. https://swagger.io/.

[3] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: Simpler, Smaller, Fast as MD5. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Michael Jacobson, Michael Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security*, volume 7954, pages 119–135. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[4] Peter J. A. Cock, James K Bonfield, Bastien Chevreux, and Heng Li. SAM/BAM format v1.5 extensions for de novo assemblies. Preprint, Bioinformatics, May 2015.

[5] Information Technology Laboratory Computer Security Division. Access Control Policy and Implementation Guides — CSRC — CSRC. https://csrc.nist.gov/projects/access-control-policy-and-implementation-guides, September 2016.

[6] Empowering App Development for Developers — Docker.

[7] eXtensible Access Control Markup Language (XACML) Version 3.0 Plus Errata 01. page 154, 2017.

[8] Genome. *Wikipedia*, September 2021.

[9] GitLab. Iterate faster, innovate together — GitLab. https://about.gitlab.com/.

[10] Global Alliance for Genomics and Health. Crypt4GH: A secure method for sharing human genetic data. https://www.ga4gh.org/news/crypt4gh-a-secure-method-for-sharing-human-genetic-data/.

[11] Global Alliance for Genomics and Health. GA4GH File Encryption Standard.

[12] Vincent C Hu, David F Ferraiolo, and D Rick Kuhn. Assessment of access control systems. Technical Report NIST IR 7316, National Institute of Standards and Technology, Gaithersburg, MD, 2006.

[13] IAM Concept of the Week: XACML, March 2017.

[14] ISO/IEC JTC1/SC29/WG11. Information technology — Genomic information representation — Part 3: Metadata and application programming interfaces (APIs).

[15] ISO/IEC JTC1/SC29/WG11. Information technology — Genomic information representation — Part 1: Transport and storage of genomic information.

[16] ISO/IEC JTC1/SC29/WG11, N17468. White paper on the objectives and benefits of the MPEG-G standard. page 10, January 2018.

[17] D Naro J Delgado, S Llorente. Adding security and privacy to genomic information representation. *Studies in health technology and informatics*, (258):75–79.

[18] Julie Olenski. Elliptic Curve Cryptography. https://www.globalsign.com/en/blog/elliptic-curve-cryptography, May 2015.

[19] B. Kaliski and J. Staddon. PKCS #1: RSA Cryptography Specifications Version 2.0. Technical Report RFC2437, RFC Editor, October 1998.

[20] Moving Picture Experts Group. MPEG-G. https://mpeg.chiariglione.org/standards/mpeg-g.

[21] Moving Picture Experts Group. MPEG-G Genomic Information Representation and Transport. https://mpeg-g.org/.

[22] Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF Protocols. Request for Comments RFC 8439, Internet Engineering Task Force, June 2018.

[23] Node.js. Node.Js a JavaScript runtime. https://nodejs.org/en/download/.

[24] Oracle. Java Software. Oracle.

[25] PostgreSQL Global Development Group. PostgreSQL, 2021-10-03T14:11:46.372062.

[26] Postman. Postman Inc.

[27] Representational state transfer. *Wikipedia*, August 2021.

[28] J. Schaad and R. Housley. Advanced Encryption Standard (AES) Key Wrap Algorithm. Technical Report RFC3394, RFC Editor, September 2002.

[29] Spring Boot. https://spring.io/projects/spring-boot.

[30] Sun's XACML Implementation. http://sunxacml.sourceforge.net/.

[31] Fatih Turkmen and Bruno Crispo. Performance evaluation of XACML PDP implementations. In *Proceedings of the 2008 ACM Workshop on Secure Web Services - SWS '08*, page 37, Alexandria, Virginia, USA, 2008. ACM Press.

[32] Pamoda Wimalasiri. A beginner's guide to XACML, April 2021.

[33] Working with BAM Files. https://www.ncbi.nlm.nih.gov/tools/gbench/tutorial6/.

[34] WSO2 Balana Implementation. WSO2, July 2021.

[35] XACML Architecture - Identity Server 5.1.0 - WSO2 Documentation. https://docs.wso2.com/display/IS510/XACML+Architecture.

[36] XML Signature Syntax and Processing Version 1.1. https://www.w3.org/TR/xmldsig-core1/.

Appendices

- GitLab repository with MPEG-G implementation: https://gitlab.com/treball-final-master/mpeg-g-reference-software