A User-Centric Access Control Framework for Cloud Computing

UNIVERSITY OF TURKU
Department of Future Technologies
Master of Science in Technology Thesis
Networked Systems Security
December 2018
Imran Khan

Supervisors:
Farhoud Hosseinpour
Juha Plosila

UNIVERSITY OF TURKU
Department of Information Technology

Imran Khan: A User-Centric Access Control Framework for Cloud Computing

Master of Science in Technology Thesis, 64 p., 1 app. p.
Networked Systems Security
February 2019

A huge amount of data is generated due to the growth of advanced information technology, online availability and easy access to cloud computing. In cloud computing, user can easily store and share their information across the cloud. With the rapid growth of cloud computing, user's security and privacy has become a serious concern. Despite various existing security mechanisms, enterprises are still afraid of losing their outsourced data and unauthorized access. In most cases, access control mechanism and authorization rule follow a web application. This makes it limited, tightly bound to web application functionality and also doesn't complete the security requirements for the individual user that results in poor protection against unauthorized access. To overcome the issue of privacy and protection, a suggestion is given in this study to empower the owner of any piece of data and information to protect their resource according to their own semantics.

In this thesis, a new approach is presented that externalize access control policy and empower the user to control access on their data according to their semantics and wishes. The proposed framework provides PKI standard base secure access control mechanism and describes the protocol interface between the different components to enforce user-centric access control policy.

Keywords: Resource certificate, Access control, Certificate Authority, Public key infrastructure (PKI), cloud computing

Acknowledgement

I would like to thank and praise God Almighty, for blessing opportunity and strength to complete this Master thesis.

I would also like to thank my supervisor Farhoud Hossienpour and co-supervisor Juha Plosila for their guidance, interest and having the patience during the study period. Their support and supervision made this study smooth and swift.

This opportunity would not have been possible without the support of University of Turku, for which I am very grateful.

I would also like to thank my friends and family especially my wife. Without their support and help, it would not be possible to complete my Master thesis.

Table of contents

List of Figures

List of Abbreviations

IoT:	Internet of things
IoE:	Internet of Everything
CA:	Certificate Authority
RBAC:	Role base access control
RFID:	Radio frequency identification
TCP/IP:	Transmission control protocol/Internet protocol
PKI:	Public key infrastructure
QoS:	Quality of service
SSL:	Secure socket layer
JWT:	Json web token
CSP:	cloud service provider
LTS:	Labelled transmission system
OAuth:	open Authorisation
EK:	Encryption key
GCP:	Google service provider
Aka:	Also known as
CEF:	chromium embedded framework
SDK:	Software development key
AC:	Access
PCM:	Profile centric model

XACML:	eXtensible Access control markup language
FC:	Functional component
SAAS:	Software as a service
IAAF:	Infrastructure as a service
ITU:	International telecommunication union
IETF:	Internet engineering task force
IEEE:	Institute of electrical and electronic engineers

1 Chapter 01: Introduction

1.1 Introduction

Stanford encyclopedia of philosophy called 21st century as the century of big data and advanced information technology. As computing becomes ubiquitous, e.g. big data and processing become in gegobyte (10^31), security and privacy have been realized to more crucial than ever. Regulators use comprehensive and sectorial laws for the collection and dissemination of confidential information in many sectors, for example, education, banking, and health care industry etc. [1][2]. Cisco estimates that there will be 27.1 Billion of network devices and connections by 2021[3]. These networks are very crucial regarding security as they are collecting very critical confidential information in our day to day life in health, banking, security infrastructure etc.

Privacy and confidentiality of an entity is the degree of personal information that defines what information needs to be shared along with the purpose and with whom. The unauthorized entity is not allowed to access sensitive information. [4] Access control of data is the core part of security in Information and Communication Technology. Many IoT industries have proven to be insecure due to their weak Access control mechanism. Online viewing and unauthorized grant access of Interconnection of sensors and cameras are examples of weak access mechanism of smart cities and big data.

As the world is moving towards the Internet of Thing (IoT) and the Internet of everything (IoE) entity, the consequence of cyberattack has become crucial and compromises the privacy and safety of citizens. This also resulted in an economic and non-economic loss. Due to the interconnection of the devices (IoT and IoE), vulnerabilities from cyber attacks are a significant issue, and current technologies fail to address the security of big data very well. To solve this issue, we believe that users are the ones that can determine the semantic and importance of their data. Thus, in this thesis, we aim to facilitate and empower the user to define their own Access Rule.

Many alliances have investigated User-centric Access control policy and different frameworks and protocols have been introduced to increase the level of security in big data, yet it requires a significant research focus. Examples of few techniques that are proposed are smart data by Farhoud [5], active bundle by Othamnei [2], open's personal cloudlet framework by

McCarthy[6], information flow control from apps by Thomas [7] and a framework xAccess by Kapil Singh [8]etc. All schemes have their importance but, in this dissertation, our primary focus is to investigate an access rule for the data owner. For this purpose, different user-centric access control policy has been systematically reviewed, and a specific access control framework will be proposed for data owner in which user can secure their resources according to their access control policy, e.g. giving limited time access to their data.

1.2 Background

Advance information security implements different access control mechanism and policies to determine access of right subject to the right object. These mechanisms make an access decision to meet the security requirement of the system. Access control mechanism has two main components: 1) subject: who executes or requests an action. 2) object: a resource that needs to be controlled [9].

Several mechanisms of access controls policy have been introduced, and the most popular are given below

1. Discretionary access control: absolutely based on the subject identity and their access to the object specified by rule

2. Mandatory Access control: an access control in which an administrator defines and manages access control and policy whereas the user cannot change or modify it. It mostly used in the system where priority is based on confidentiality. [10]

3. Role-based access control: RBAC based on the role of the user or subject within the system.

In this master thesis, different user-centric access control mechanisms have been reviewed and technically analysed, and a framework for user-centric access control mechanism will be proposed.

1.3 Problem Statement

We want the user to have access control on their data and should have a choice to decide a specific period to provide access to the data. Although existing security mechanism research shows that people afraid of adopting cloud computing due to unauthorised access and losing

outsource data. To solve the issue of unauthorised access control, we will use PKI standard mechanism to empower the owner of the data who will decide on access decision.

1.4 Methodology

Interconnection of networks and keeping data with a third party (cloud computing) makes the data less secure or increases the risk of cyber-attack. To overcome the issue of protection and secure the data over the network, the following procedure will be conducted in this thesis:

1. Different user-centric Access controlled policies will be reviewed.
2. A new user-centric Access controlled framework that facilitates the user to protect their data according to their own criteria will be proposed.
3. A demo of the proposed Framework will be implemented.

The framework will be practically implemented by using a computer program written in Python language. First, a lightweight, secure cryptography technique will be selected, and second, a scenario will be developed in python language that explains the proposed framework and shows the movement of authorised and denial of unauthorised access to the data.

1.5 Aim and Objective

The study aims to protect the big data and facilitate the user to secure their data according to their specific access control requirements. Also to investigate the existing access control policy, to propose a new framework that facilitated the cloud environment efficiently to secure big data.

1.6 Motivation

Advancement of Information technology and the growth of IoT made everything interconnected and accessible from everywhere. It shrinks the world and facilitates the users; however, it easily opens the ways for cyber-attacks. data breaches have become a common occurrence and can happen in daily routine. Just today on 10 July 2018, an incident reported on www.trendmicro.com, www.databreachtoday.com, and www.infosecurity-magazine.com. This news confirmed data breaches on German-based host provider “Domain factory”, A smartphone app “Timehop” and Macy’s and sister retailer company “Bloomingdale’s”. About

21 million users are affected by this breach and resulted in confirming stealing of the important and sensitive information.[11]

To overcome the issue of data breaches and protection of own resource, a user-centric Access control policy framework will be investigated. This involves user to secure their data according to their security and access control requirements. In this framework, the owner of data will be alerted in case of any unauthorized access. So, whenever an unauthorized user accesses the confidential data, an alert will be sent to the owner and owner can protect their data.

1.7 Outline

The remaining part of the thesis is organized as follow. Chapter 2 explains the technical background of Information security and multiple other topics, which are essential for understanding the crux of this thesis. Chapter 3 proposes a new framework that facilitates a user to protect their resource according to their requirements. The implementation of the demo of the proposed framework is explained in chapter 4. The dissertation is concluded, and remarks for future work are presented in chapter 5.

2 Chapter 02: Literature Review

The Internet of thing (IoT) and the Internet of everything (IoE) have revolutionized the way we live and made dramatic changes in our life. It is called the efficiency enabler in different sectors including health, transport, logistic, banking and manufacturing etc.

However, IoT assists in the optimization of processes and works as a catalyst for innovative marketing, but their growth provokes challenges to our freedom and security. [12] Growth in cyber-attacks and data breaches make the information security and privacy as the top priority for IT leaders and businesses. These attacks can be controlled through a powerful enterprise information management strategy and tools. [13]

2.1 Internet of Things

Internet of thing (IoT) emerge itself as a more successful and prosperous area in the era of smart advance technology and ubiquitous computing. In 1999, Kevin Ashton gave the idea based on a challenge to the network published in the seventh report of ITU in 1997. The idea was to propagate the information of the physical world through the web. Later this name was called as “Internet of Things” in 2005. [14]

Currently, we have a different definition of IoT. There is not a single definition, but it is defined differently by organizations that work for the standardization process. According to IEEE IoT is a “network of items-embedded with sensors which are connected to the Internet”. ETSI does not use the word Internet of Things. They define it as a machine-to-machine communication. ITU called it a “Ubiquitous Network” which means everywhere. So, they endorse the definition of IoT as a “network that is available anywhere, anytime by anything and anyone”. According to IETF “the basic idea is that IoT will connect objects around us (electronics, electrical and non-electronics) to provide seamless communication and contextual service provided by them. Development of sensors, RFID tags, activators, and mobile phones make it possible to materialize IoT which interconnect and cooperate to make service better and accessible anytime from anywhere”. [15]

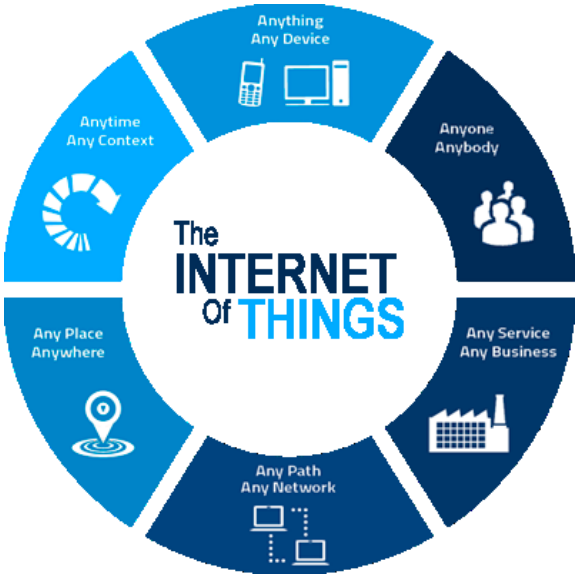

Figure 2.1 Internet of Things [16]

2.2 Information Security

Information security is a vast subject; therefore, its definition varies in literature, but almost all of them correctly define the basic principle adequately. A brief and simple definition of information security is the confidentiality, integrity, and availability of information or protecting the information in all forms from unauthorized access, disclosure, use, modification, examination, moving, copying, and destroying. The most crucial attributes of information security are given below [17] [18]

- **Confidentiality:** Confidentiality means that information is only accessed by authorized entity or process and prevents it from unauthorized access or disclosure.

- **Integrity:** integrity is the righteousness, trustworthiness, completeness, and accurateness of data and protection of information from improper and unauthorized modification, deletion or alteration.

- **Availability:** availability refers to the reliable accesses and use of information by an authorized entity. An attack that affects availability is called denial of service attack.

- **Non-Repudiation:** a property in information security that protects the denial of sender or receiver of certain information. In ICT, non-repudiation assures the delivery of data

to the sender and the sender's identity to the receiver. This is done by digital signature so that none can deny it later.

- **Accountability** is another important principle that assures the traceability or evaluation of an individual's or organization's performance. [19]

2.3 Access Control

Access control is the core part of information security. At this part, security engineering meets computer science. Access control prevents unauthorized access to the resource and limits the access of the user to a specific boundary to prevent the security breach. [20]

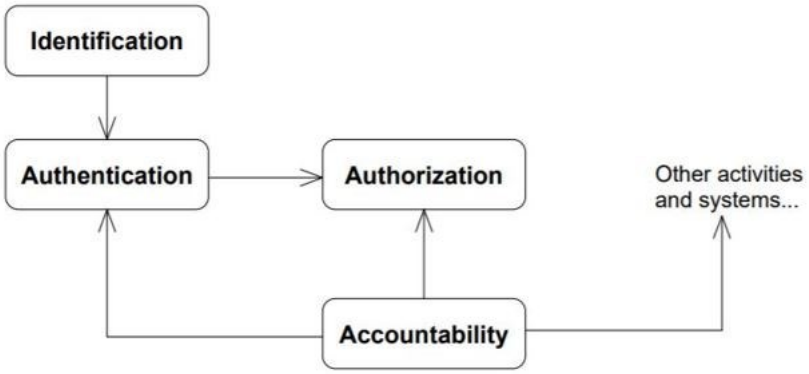

Figure 2.2 Access Control Steps [21]

- **Identification:** Identification is the first step that occurs numerous times by user or computer while accessing the information or information process.

- **Authentication:** Authentication followed by Identification is the stage or property that verifies the authenticity of the identity.

- **Authorization:** authorization refers to the right, permission, and privileges of an authenticated user. It ensures the right of the requested operation and prevents an unauthorized request. The authorization process is defined by security policy and set by the security administrator. [18][21]

2.4 Cryptography

Cryptography is a Greek word, which means ‘hidden words’, Cryptography is a science that protects the data by converting them into an unintelligible form. It is the most important and a basic building block of Information security. It is used to store or transmit the data over an unsecured network so that only an intended receipt accesses it. [22] [23]

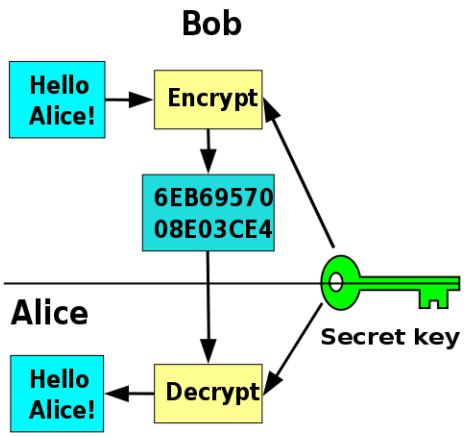

Figure 2.3 Cryptography

2.5 Privacy

Information privacy also known as data privacy is a concept used for both individuals and society. Individual privacy means, the individual has control over their data and defines what data can be collected, shared and used. Societal privacy is the autonomy of society that limits the power of the government interferes.

The concept of privacy changes over time, area and culture. For example, sharing of family picture and information 100 years ago was less common than now.

2.6 Public key infrastructure

Public key infrastructure (PKI) is a system or framework that creates, stores, distribute, manages and revokes both public keys and digital certificates. The main purpose of PKI is to facilitate secure transfer of electronic credential and information over the Internet, e.g. online banking, emails, shopping etc. PKI built trust that is confidence in, reliance on. PKI consists of many standards, policies, and procedures to build trust between two persons and entities. In public key infrastructure, trust comes from a Certificate authority called C.A. [24]

2.7 Certificate Authority (CA)

Certificate Authority is an important entity of PKI framework that verifies the identity of the users in online communication. CA has the responsibility to generate, validate, revoke, and manage the digital certificate. A digital certificate is the electronic credentials that are used in verification and establishment of identities in electronic communication and transaction. In other words certificate, authority is an entity in the PKI system that protects the authenticity and integrity of the certificate to maintain trustworthiness between strange users. [24]

2.8 SSL

Secure socket layer (SSL) is a cryptographic protocol that is widely used to secure communication over the Internet. SSL provides a secure encrypted link between the web server and a web browser. The encrypted link ensures the transaction between a web browser and server will remain integral and private.

SSL certificate is a small data file that needs to be installed on the web server. SSL certificates normally contain a domain name, company detail, details of certificate Authority responsible for issuance certificate, the expiry date of the certificate, and public keys etc. [25]

2.9 Smart Data

Smart data is a data structure that is design and developed by Farhoud et al. in his doctoral project to tackle the big data in the IoT system. The proposed approached is an encapsulated data structure of payload, metadata, and a virtual machine. Metadata has the rule and regulation for processing of payload while virtual machine executes the rules of metadata. In other words, they made the data in an intelligent and active form that facilitates in reducing/saving cost, network traffic, energy as well as QoS. Initially, the smart data is generated by IoT in a very basic form, but due to their intelligence and smart nature, it evolves on their life cycle into a piece of meaningful information. [5]

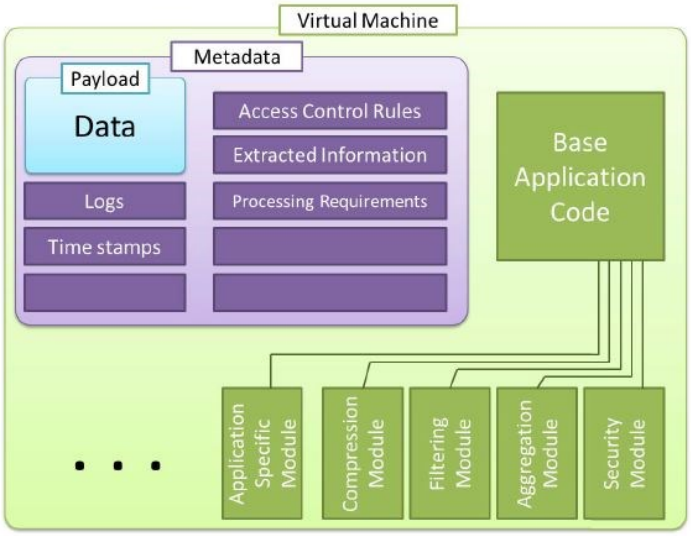

Figure 2.4 Structure of Smart Data [5]

2.10 Related work

A. Kapil et al. in their work on user-centric access control policy for a web application, proposed a framework called xAccess. The framework is based on Role-based access control scheme that has two main components. The first component is hosted on user's computer and works as an extension within the browser to implement the rules on the content, while the second runs on the server side of the web application to analyze the content according to the defined rule. The author also mentioned that the proposed framework is generically based and can be incorporated with other Access control models. [8]

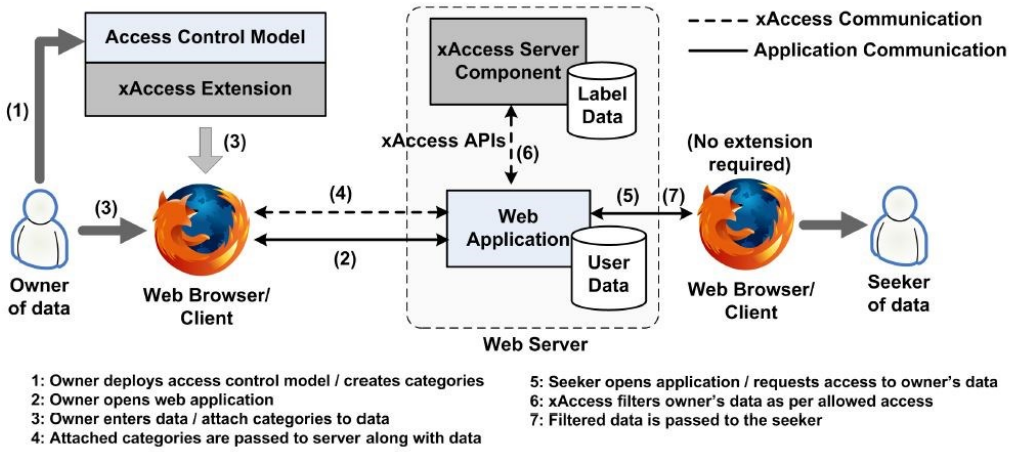

Figure 2.5 xAccess architecture and workflow scenario [8]

B. Beltran, Martinez, and Skarmeta in 2017 presented a decentralized security and access control policy for smart cities. In their research work, they proposed a Functional Component (FC) in the SMARTIE Platform that is compliant with IoT Architectural Reference model (ARM). XACML, DcapBAC (capability Based Access Control) and CP-ABE (ciphertext Based Attribute-based encryption) are the main components of security and privacy. DcapBAC defines authorization rules for the IoT Devices regarding client perspective. It issues a JSON/encoded base token that defined the client's authorization. CP-ABE functional component encodes the data according to user-defined preference. The user defines the rules at XACML for application-specific attribute data type and gives access to the related client. [26]

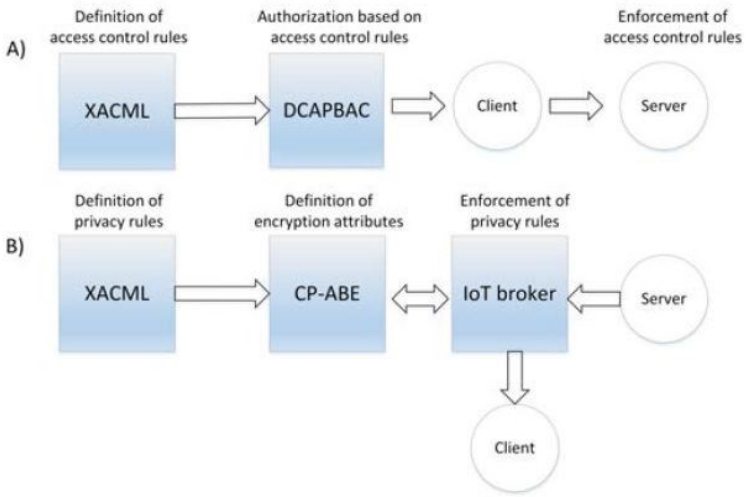

Figure 2.6 Communication flow for the Smartie Decentralize Access control mechanism DCapBAC and CP-ABE [26]

C. McCarthy et al investigated how the application can use their data over Openi's platform and empower the user for fine-grained access control. They investigated-to incorporate the mechanism for fine-grained access control, data reuse, user control, support for existing business model and web scalability. These objectives are achieved by object base AC, stateless JSON Web Token (JWT), REST-based endpoints and OPENi Types. The main feature in personal cloudlet Framework is OAuth & JWT, Openi Type Registry, Fine-Grained Access Control, User Dashboard, Privacy preserving aggregator and NoSQL datastore and Microservice Architecture. [6]

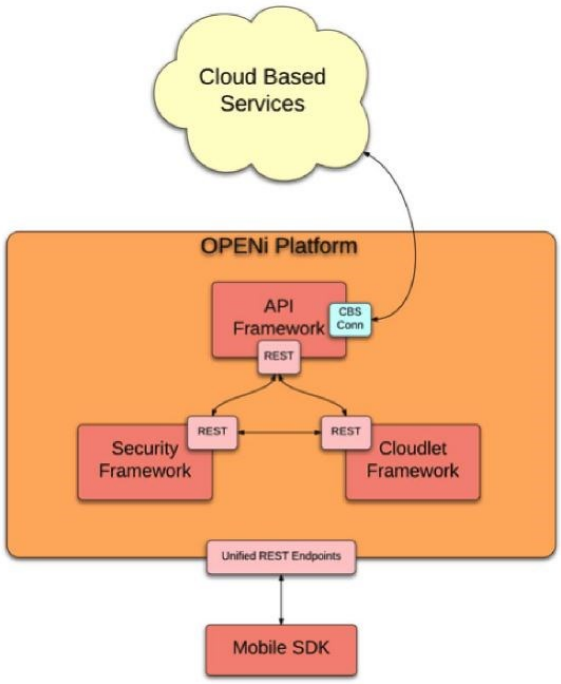

Figure 2.7 Main component of Open Architecture [6]

D. Roshandel and Tyler worked on "User-Centric Monitoring of sensitive information Access in Android Application" to help Android users with their privacy and accessibility to the sensitive data. For this purpose, they introduced a new class 'ContentMonitor' in Android OS that works on the same principle as 'ContentObserver'. This class not only gives a notification message but also informs the users about the operation type, i.e. insert, delete and update in the resource. They also developed a Monitoring app 'MobileDataMonitor' which collect information for data whenever they accessed. It normally stores type, frequency, time and corresponding app processes. They also used 'Hidden Markov Models' (HMM) to define transition and observation probabilities. [27]

E. Msahli, Chen, and Serhrouchni in 2014 proposed a profile base Access control model called Profile-Centric Model (PCM). In PCM, a user must validate his profile (A profile is the combination of Authorization, Condition, and Obligation) before access to store data. Chen et al. proposed PCM as a profile graph-centric implementation that gives support in user profile management, condition profile management, authorization profile management, profile-profile relationship, and obligation profile management. Profile graph implementation help in an exclusive membership among the profiles and access control administration. [28]

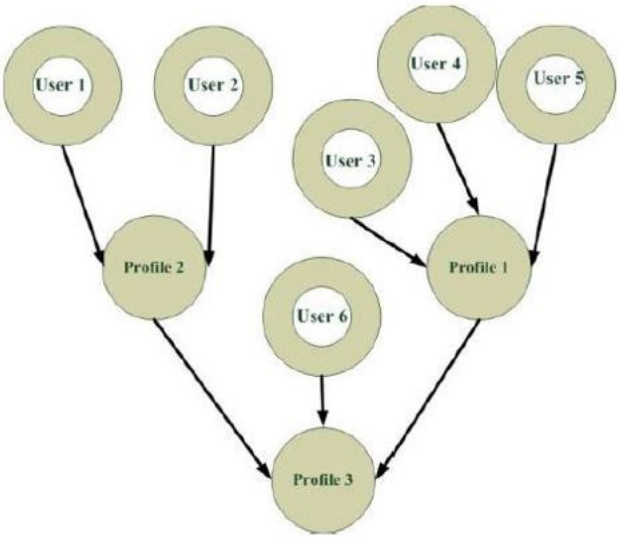

Figure 2.8 Profile Graph [28]

F. To remove the security fear for using different services of a cloud service provider, Diener et al. in [29] developed a user-centric tool which decides the criticalness of the data for processing over the cloud. The tool is implemented by a machine-learning algorithm, which assigns the sensitivity of the data. They believe that their machine-learning algorithm involves the user in data analysis and as a result of it reduces the fear of cloud adoption. A direct connection is established between the document and a CSP. User (employees) requires to know only the type and class of the documents. Consider the figure number the supplier contract belongs to contract class and employee's salary slip is related to invoice class etc. the security level is pre-established by the enterprise, i.e. public, secret and top secret etc.

document type	security level
Invoice	Extranet
Contract	Secret
Press release	Public
CAD	Secret
HR data	Top Secret
Price calculation	Top Secret
Organizational instruction	Extranet

Figure 2.9 Security Policy: Relationship between document types and security levels [29]

G. Grace and Surridge studied user-centred privacy model in 2017 [30] and presented a privacy model, Labelled Transition System (LTS). Labeled Transition System ~~a~~nalyses the action of online service carried out on user's data to determine their conflict on a

user's privacy preference. LTS is composed of two key elements. i.e. States and Transition. The state represents the user's privacy, e.g. agreeability or conflict with privacy preference. While the transition is the action between two states. LTS studies the potential behavior of the discrete complex system and allows the user to know how discrete complex system used their sensitive data. To demonstrate the effectiveness of the LTS Grace and Surridge applied the access control framework over medical services.

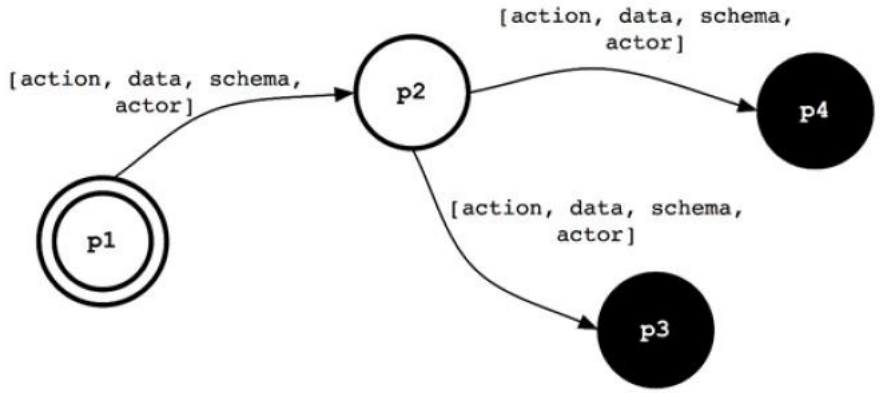

Figure 2.10 LTS Model of User Privacy with system behavior [30]

H. A framework for user-centric access control policy management for cloud application is proposed by Ghafoor et al. in [31]. The proposed framework facilitates the user to protect their resources according to their security requirement. In the proposed framework, a user first authenticates himself to the cloud then take a ticket from the authentication server and send it to the authorization server for the access control module. Authorization server evaluates the ticket of access request concerning previously stored policy. If the user is permitted then access is allowed to access the control module where they can write their own security rule to protect their resource.

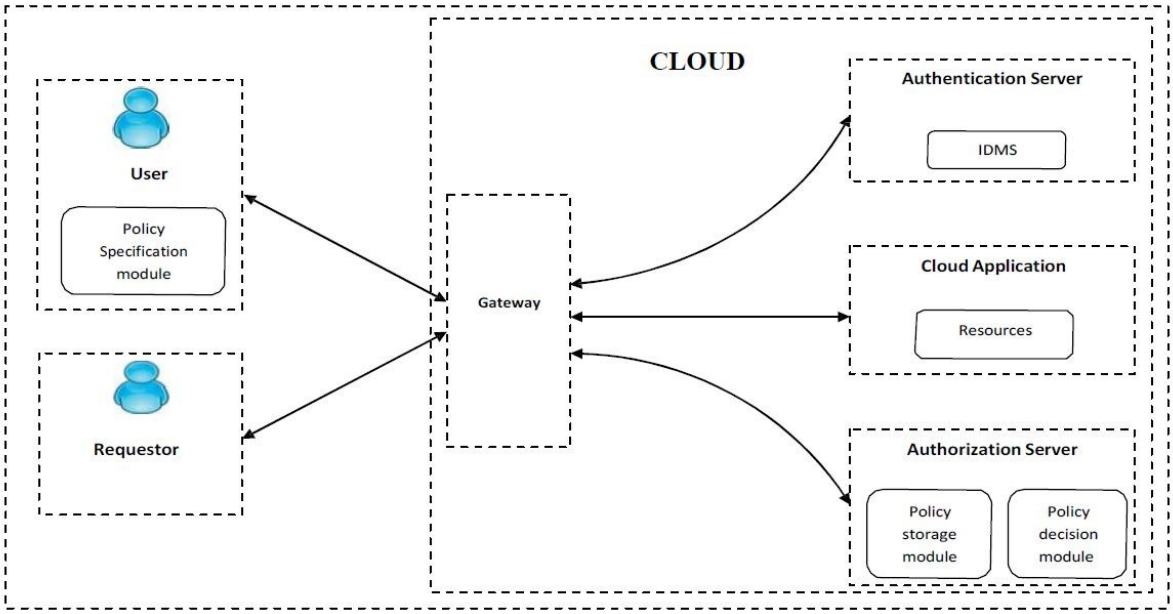

Figure 2.11 User Centric Access control Policy management Framework for cloud application [31]

2.11 Summary of Related work

All the researchers have done a great job to protect their resource from unauthorized access. Everyone proposed their ideas and somehow gave power to the data owner to protect their own resource. All of them achieved good results, but no one focused on the PKI standard base security. So, in this thesis, my focus is on the PKI standard base security in which a trusted party issues an authorization certificate to client valid for a specific period.

3 Chapter 03: Framework

In this work, a framework has been developed for a client to protect their data according to their security requirements and Access control. The design is based on analysis of different systematic reviews of user-centric Access control policy. The components and the protocol required for their communication are given below

Data owner (client owner): Data owner is the one, which provides new information that other consumers are interested in, 'or' a person or entity that owns the data and create, edit and manage access control on its resources.

Requestor (Client User): A user or application who tries to get access to another data user's protected resource in the cloud application.

Gateway: This component performs as a proxy module between users and the components specified for security services. It not only manages the users' communications and sessions but also handles their requests to access data and receives the decisions regarding access control from the authorization server. RSA cryptosystem is used to encrypt the communication between Gateway and Users to keep it secure.

Authentication Server: this server verifies the identities of users. The process of authentication includes the provision of digital certificates by the users to the authentication server, which after verification provides tickets to the authenticated users to get access.

Certificate Authority (CA): CA is a trusted third party between cloud and clients, and is responsible for issuing digital certificates for authentication, integrity, and non-repudiation

Authorization server: This server contains the policies related to access control, which is defined by Cloud users (for their resources) and Cloud applications (for their registered users). Based on these stored policies, it also generates decision regarding access control.

User AC Rule Component: it enables the user to create and implement their policies for Access Control on their data.

3.1 Protocol

The proposed protocol involves three steps, i.e. Authentication and Authorization, Access Control Policy Specification and Accessing Protected Resource. For the proposed protocol, the following assumptions are taken into consideration.

- The authentication server has registered all cloud users.
- The authorization server has predefined stored policies for access control of the registered cloud users, provided by the Cloud Service Providers.
- The cloud and authorization server have already established a trust relationship.

Proposed Access rule framework has the following steps:

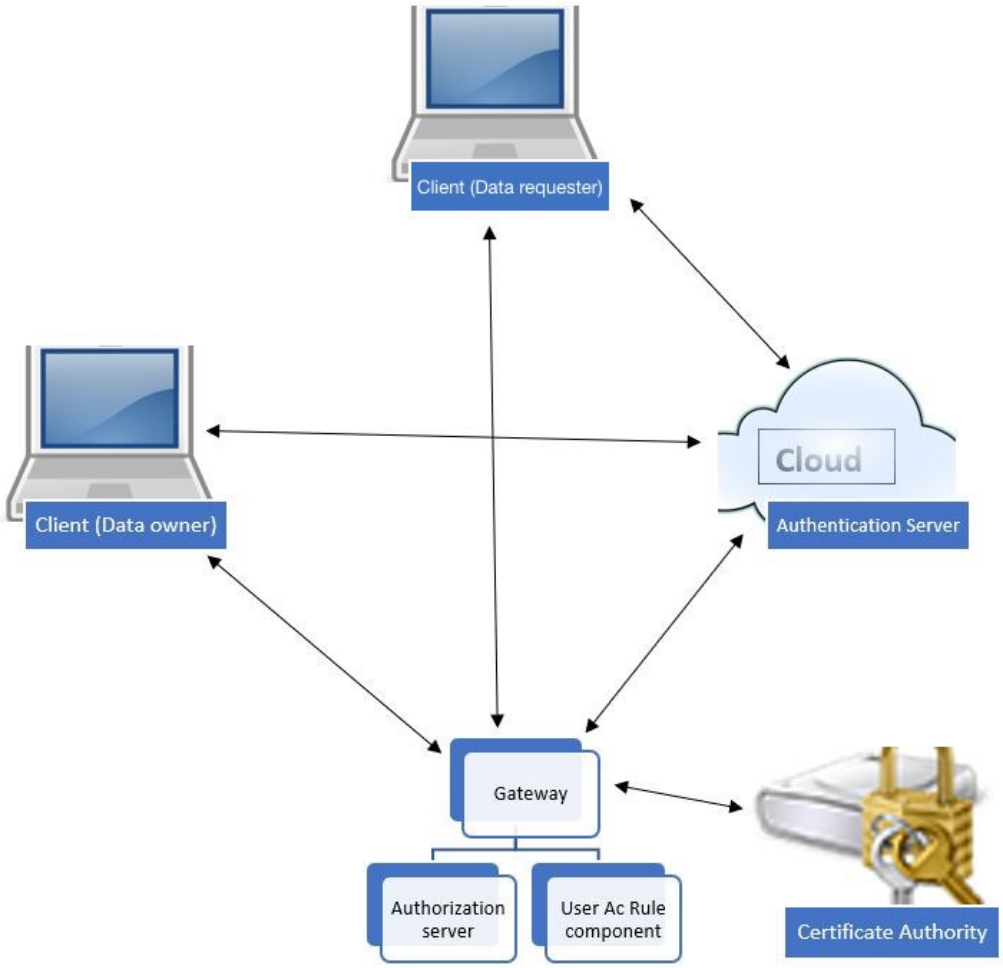

Figure 3.1 User Centric Access control policy Mechanism

3.2 Authentication and Authorization of clients

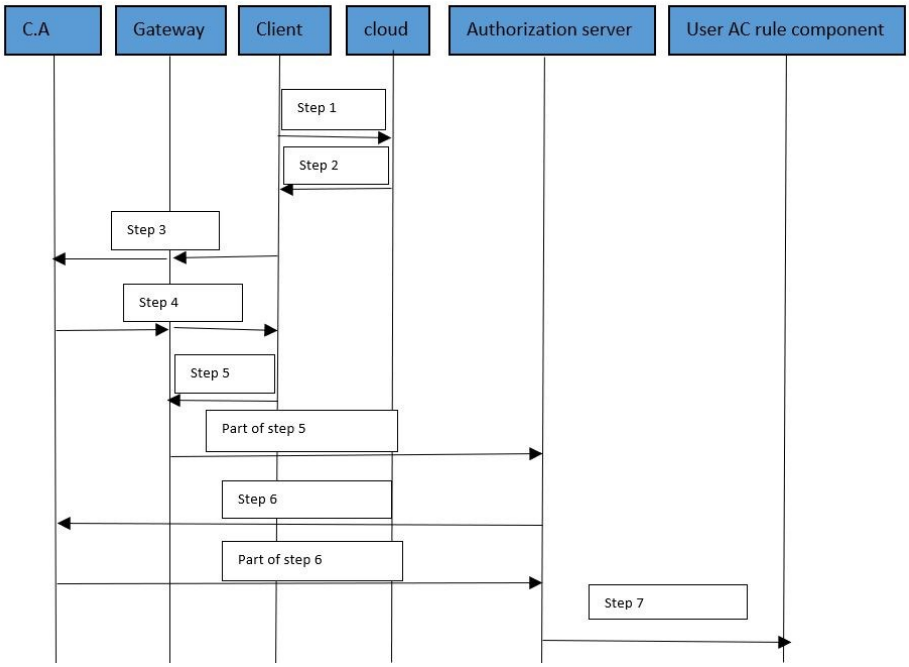

Figure 3.2 Authentication and Authorization Process

In the proposed protocol, Double Authorization of the client will be done to allow access to the Authorization server/Access rule component of the cloud. First, an Authentication and Authorization is done by using an OAuth protocol then required PKI certificate is used to access the data authorization component. Main Steps are summarized below are

1. First, client (data owner or data requester) sends an Authentication Request to the cloud by using an OAuth Protocol,

2. Cloud will authenticate the clients (cloud user) and issue an Authorization token for the clients (cloud user).

3. Now cloud user will request CA authority for a digital certificate through Gateway.

4. CA will issue a certificate for the cloud user using RSA encryption. The certificate includes an Encryption key, EK (Validation period, Username, and User unique ID).

5. The client will send an authorization request to Authorization Server through Gateway using their CA certificate plus Authorization Token.

6. Authorization Server will verify the certificate from the CA server and evaluate the request according to the previously stored policy by the cloud service provider (CSP).

7. On the success authorization, the client can upload their resource or request another client to access their resource.

3.3 Access protecting policy specification

1. In the user 'AC Rule Component' see Figure 3.1, the user can define access policy on their data

2. Defined policies along with User ID will be saved to the User AC Rule Component which will be used by authorization server as an access policy.

3. The authorization server also allows the user to edit previously created policies for their data and update them as well.

3.4 Access protected resources

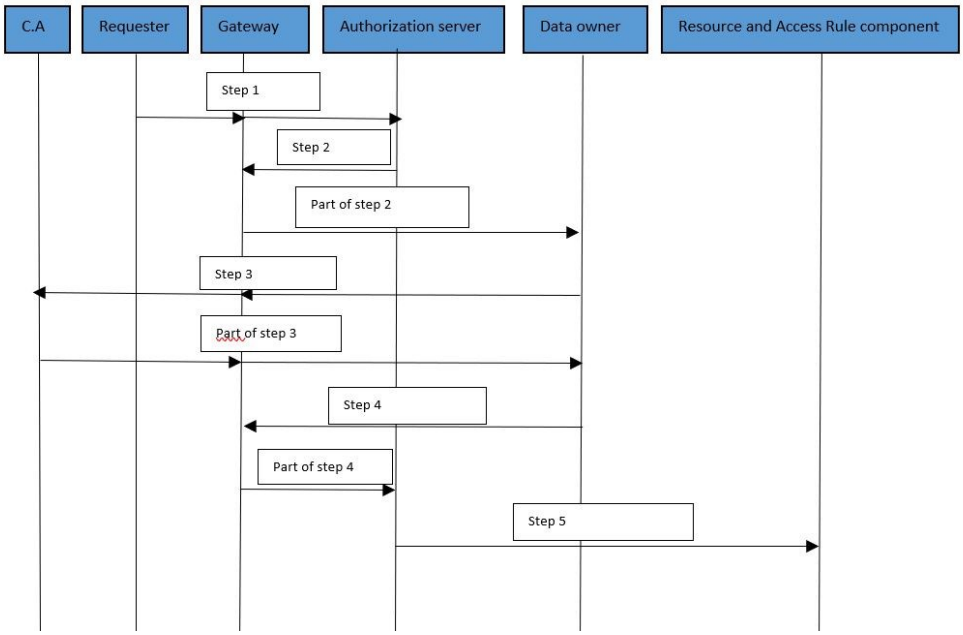

Figure 3.3 Client Request Access process

1. If requester (cloud user) wants to access another client's protected resource, then he or she needs to request Authorization server via the gateway for accessing the resource.

2. Authorization server will forward the request towards the data owner (resource owner) through Gateway.

3. On the request, the owner will decide to give access to the resource. Access will be allowed by a resource certificate with the help of CA for a specific period. In case if requester has already got resource certificate than C.A will check the validation of resource certificate. Resource certificate use an arguments EK (Valid from|| valid to|| User name|| User Unique ID|| and Resource ID). EK stands for Encryption key

4. On the request made by a new user, the owner will send a resource certificate to the Authorization server for giving access to the resource.

5. On the request made by an old user, after following verification step, permission granted to access the resource message will be sent to the Authorization server. All communication happens through the gateway.

6. User (who made request) can access the resource as long as the certificate is valid

4 Chapter 04: Framework Implementation

4.1 Summary

This document summarizes the particulars for data access protocol implementation. The implementation is a demonstration of a real-life working example of data access protocol. This demonstration is implemented in the form of a framework, covering only the major steps which are essential and are a minimal criterion to ensure the benefits of using this protocol.

Major actors of this framework are **data owner** and **requester**. The owner can upload his data into the cloud or any third-party service, which allows saving data remotely and provides ways for other users or third-party services to access it. Other users can not access any data uploaded by data owner unless data owner himself has allowed it to do so. Though the requester will be notified in real-time about the availability of information, they will be only able to access it after approval has been passed through the framework. When a new requester approaches the data, the owner will be notified. It is data owner's sole discretion to allow or reject any requester.

The safekeeping and ubiquitous availability of information are ensured via cloud services. Cloud services provide not only data storage facilities, but they also provide application deployment and infrastructure facilities, on which this implementation stands. Generally, **Firebase**, a part of the Google Cloud platform, is used for this purpose. This implementation utilizes Firebase's Real-Time database service, data storage services, and messaging and authentication abilities. The role of the requester as a third-party or a remote user is excellently mapped by using Firebase's authentication and authorization provider system. data owner's administrative privileges on data were available by Firebase's cloud rule scripting and Firebase Console.

A preliminary requirement of this protocol is the establishment of the end-2-end encryption communication model. Establishment of End-2-end encryption communication model is a preliminary requirement of this protocol. However, since it is already a well-known and available feature, and outside of the scope of this implementation, it is not covered in the study, rather a Firebase cloud service is relied upon for this functionality. However, this protocol

requires a second level of encryption, to restrict any party interested in information to directly access it. To address it, this implementation includes a custom **Gateway** built to oversee, relay and administer actors' communication with the cloud.

Gateway's responsibility is to provide secure client application to actors for information handling as well as to modifying the actors' authorizations based on available rules. These rules are saved in a subcomponent of Gateway called **access rules component**. In this work, Firebase Access Rules are used by Gateway, to determine appropriate authorization levels to any user.

Seamless but secure Authentication and Authorization are important features of the designed protocol. **Authorization Server** stores authorization levels for each user. Both Cloud and Gateway are allowed to access Authorization Server to validate authorization tokens on any request. Firebase's authorization capabilities are used for the authorization server as well.

4.2 Tools Selection for implementation

While implementing the framework, software components are selected based on their popularity, availability, and effectiveness in constructing the required building blocks for each module. The selection is made on the basis of three factors given below:

Popularity: means whether the software component has a widespread use by the masses or individuals. Here, the masses mean industry giants and tech corporations. "Individuals" refers to individual developers, individual industry leaders or open source organizations.

Availability: means whether the chosen component or tool is available under an open source license or if it is proprietary. Is it available free of charge for education or research purpose?

Effectiveness: means how a component is going to help, demonstrate the purpose of the module and the role it plays as part of the whole framework.

4.3 Overview

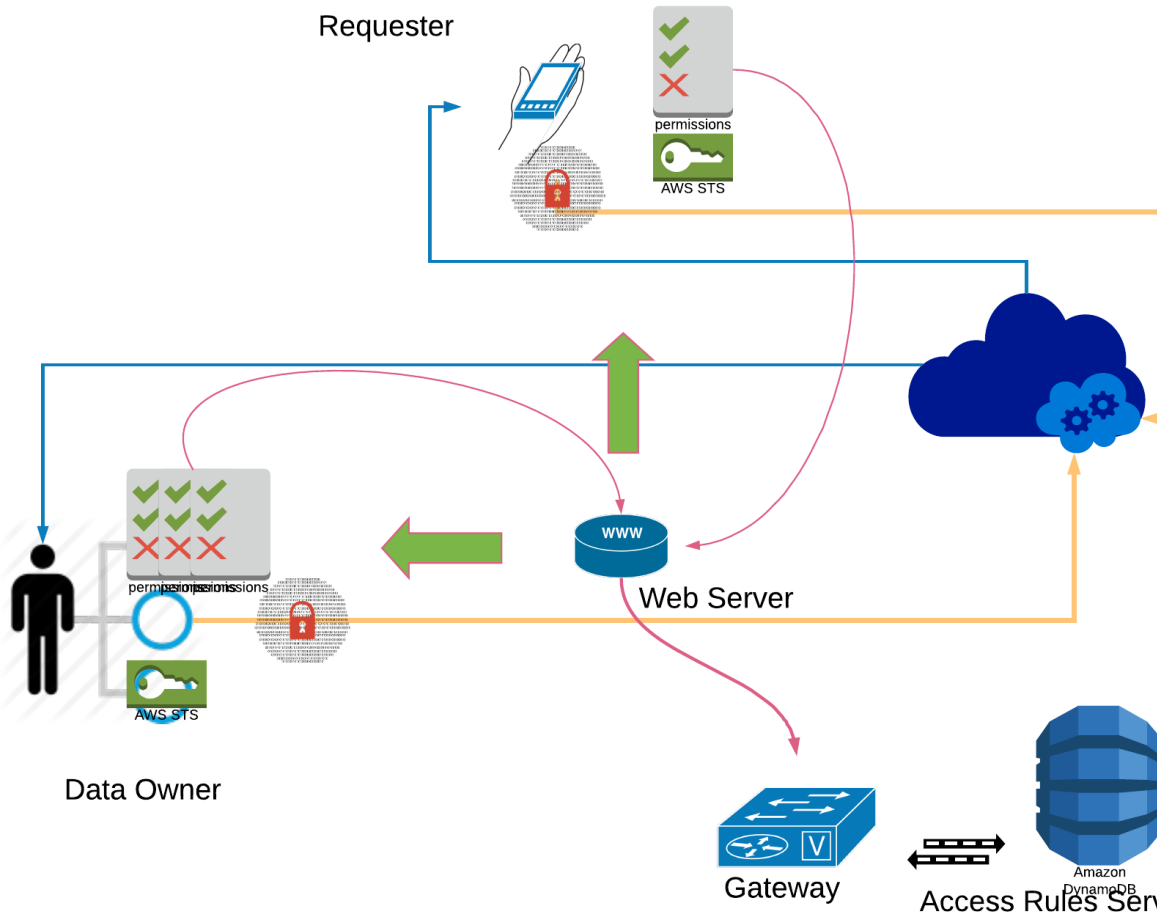

Figure 4.1 implementation specific broader view of the implemented framework.

Entities forming the framework in Figure 4.1 helps to run the system while following Resource Certificate base Access protocol. Each entity sits on a technology stack, or specifically a set of software components. These set of software components help the system simulate or emulate the behaviour in a similar way a modern real-life would do. Since each entity holds a set of software components as well as its implementation details, entities of Figure 4.1 will be referred to as “modules” in this work.

4.4 Actors

Implementation of the actors, **data owner,** and **requester**.

Actor modules include either source of data or consumers of data, effectively playing a role of information Sink. Actors need to either share or receive information over the network. The associated software components used in implementation enables the framework to register, send, request and receive information in a secure and informed way.

4.4.1 Data owner

Data owner is the one which provides new data that other consumers are interested in. Data owner must upload the data on a secure public repository like a cloud platform to make the data accessible for other users as well as third-party entities.

Cloud platforms use their security mechanisms while transmitting data from one node to another. These include Identity Verification, Role Verification, and Encryption.

Identity verification falls under **Authentication**, where a system user exchanges keys and passcodes to validate the authenticity. Successful authentication is usually accompanied with role verification, referred to as **Authorization**. After successful authentication and authorization, a session is generated on the cloud. A session usually consists of encrypted strings of text, containing the user's identity, user's device, allowed operations and timing information.

4.4.2 Requester

A requester is a role interested in remotely consuming data. This framework does not restrict remote, unattended data access, rather it ensures a secure and informed way requesters can query, request, change or read information.

Authentication and Authorization mechanisms are the same for all requesters, except the owner. For the requester, more restricted authorization rules are set to access cloud data.

Data owner is authorized to manipulate requester authentications. This will be further elaborated in the "Cloud" module.

4.5 How data owner or requester is authenticated

In the figure Figure 4.1, the orange line originating from data owner represents the authentication request that the data owner makes to verify his identity to perform further actions.

In this framework, Google Cloud Platform, aka GCP is used as a cloud entity, shown by the dark blue cloud in Figure 4.1. For authentication, authorization and information storage, GCP's cloud service Firebase has been used for its simplicity, availability and low cost. A smaller cloud represents firebase with a relatively lighter color and with two gear icons in Figure 4.1.

For a successful handshake and authentication, data owner are restricted to access through either a valid third-party authentication provider or via GCP itself. Firebase only allows a defined set of sign-in methods and each method should explicitly be enabled by data owner or administrator.

In this framework, it is ensured that Owner and requester will use different authentication providers.

Each authentication provider is required to talk over a protocol to successfully exchange authorization information securely and efficiently. In this framework, OAuth 2.0 has been used for authorization protocol. It will be further explained in the "Cloud" module.

4.5.1 Data owner client

Data owner will connect to GCP through a standard client. In this framework, the web platform is chosen as it is easily accessible on desktop and handheld devices as well. The web client can connect to Firebase through JavaScript SDK. This SDK has all the required pre-build APIs required to implement a successful client for the framework.

4.5.2 Browser Emulation

To emulate a standard browser so that clients for both owner and requester can be run, Chromium Embedded Framework (CEF) has been used. Since the implementation is made by using Python 3.6, a python wrapper for chromium embedded framework cefpython3 has been used to lunch two isolated clients. The isolation is made possible by launching new instances of browsers with newly created empty user profiles.

4.5.3 Client Implementation

To simulate a client app, a simple and renowned UI framework, Bootstrap has been used to create the user interface. Actors interact with the UI built on Bootstrap and HTML5, and their actions are then communicated with servers using JavaScript.

When launched, clients first initialize by providing a list of available authentication providers which data owner or requester can use to access for authentication and authorization.

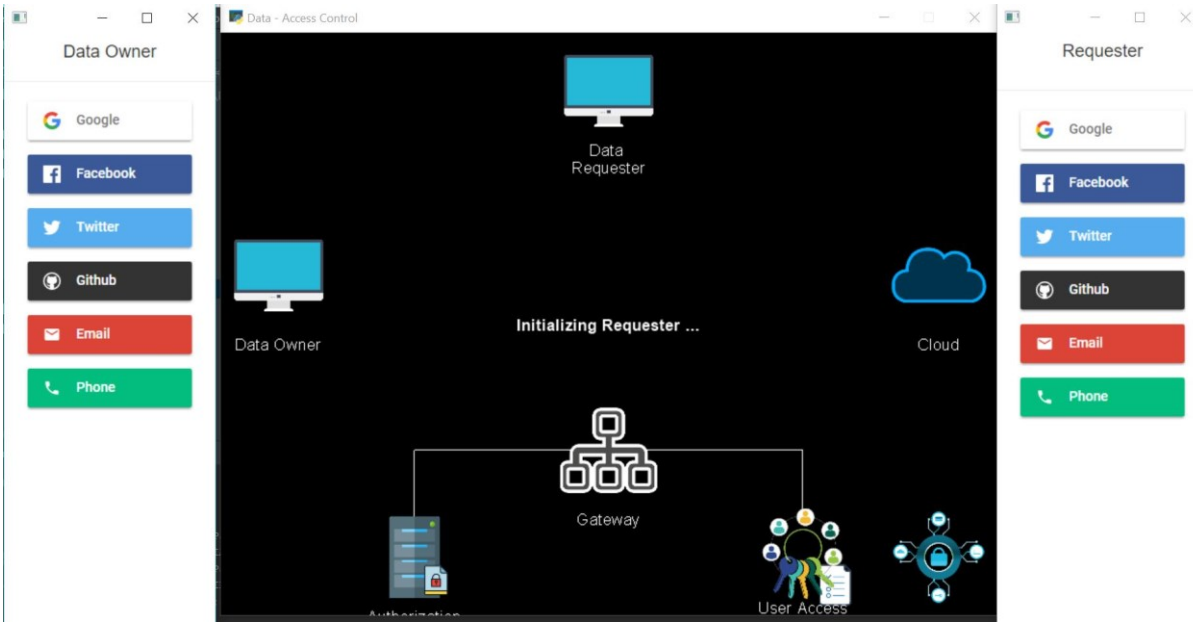

Figure 4.2 running implementation

Figure 4.2 demonstrates a running implementation. The left window is emulating a browser and running client for data owner whereas the rightmost window is running a client for requester.

After successful authentication, another interface is provided to either produce or consume data

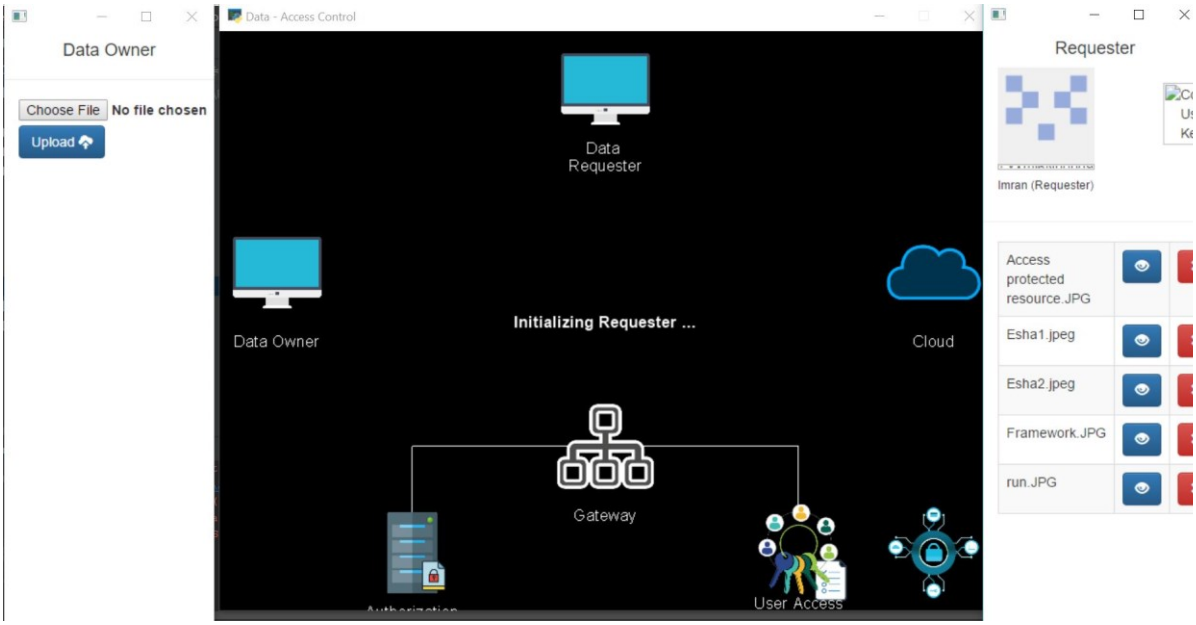

Figure 4.3 Interface to produce or consume Information

Figure 4.3 shows the interface that the data owner can use to produce information and requester can use to consume information.

4.6 Cloud

4.6.1 Google Cloud Platform

Google Cloud Platform is a diverse cloud services provider, which offers several facilities tailored to different consumers. Google Cloud Platform is chosen for its popularity and available services that could help implement the framework. The framework implementation is based on Firebase, a Google Cloud Platform's Application development service provided as SaaS (Software as a Service) and IaaS (Infrastructure as a Service) to consumers. This allowed us to efficiently implement protocol steps while paying less attention to configuration and hardware setup. Also, Firebase is a modern standard in cloud computing; thus its usage is perfectly aligned with the novel approach the data access protocol advocates.

4.6.2 Firebase

Firebase provides many facilities for the implementation of professional web and mobile application. Firebase provides Real-Time database which is a vital part for live updates and messaging. The following is the summary of Firebase services used in this implementation and the way this framework benefits from those services:

4.6.2.1 Real-time database

Resource certificate base data access protocol is based on two major functions, information availability and controlled access. Firebase's Real-Time database allows us to provide a reactive interface to both data owner and data requester. As soon as data owner provides new information, requester gets updates on his client.

4.6.2.2 Authentication and Authorization

Firebase provides a powerful admin dashboard to the data owner; the data owner can enable other third-party authentication providers or monitor their activity. Data owner can directly access Real-Time database by signing in as the admin user. Both data owner and requester can authenticate themselves as a user of the system. The owner can use his client to upload new information whereas requester by using his client, request and consume owner's information.

4.6.2.3 Storage

While Real-Time database provides facility to store lists of available records and logs of events, Firebase Storage provides storage where sensitive media like documents, images and videos can be stored after encryption. The framework allows the data owner to upload images, which with the help of gateway stored in Firebase's Storage Service.

4.6.2.4 Cloud Messaging

Each information related to activity is logged in the Firebase Real-Time database. To update the data owner about these activities in a non-interrupting but effective way, Firebase's cloud messaging has been used. This ensures data owner will get the notification in time irrespective to the device which has been used to access.

4.7 OAuth 2.0

OAuth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 focuses on of easification of client site development while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices.

4.7.1 Sign-In Methods

Firebase provides an easy interface to enable and configure Sign-In methods. data owner has access to allow the requesters to enter into the system through Sign in.

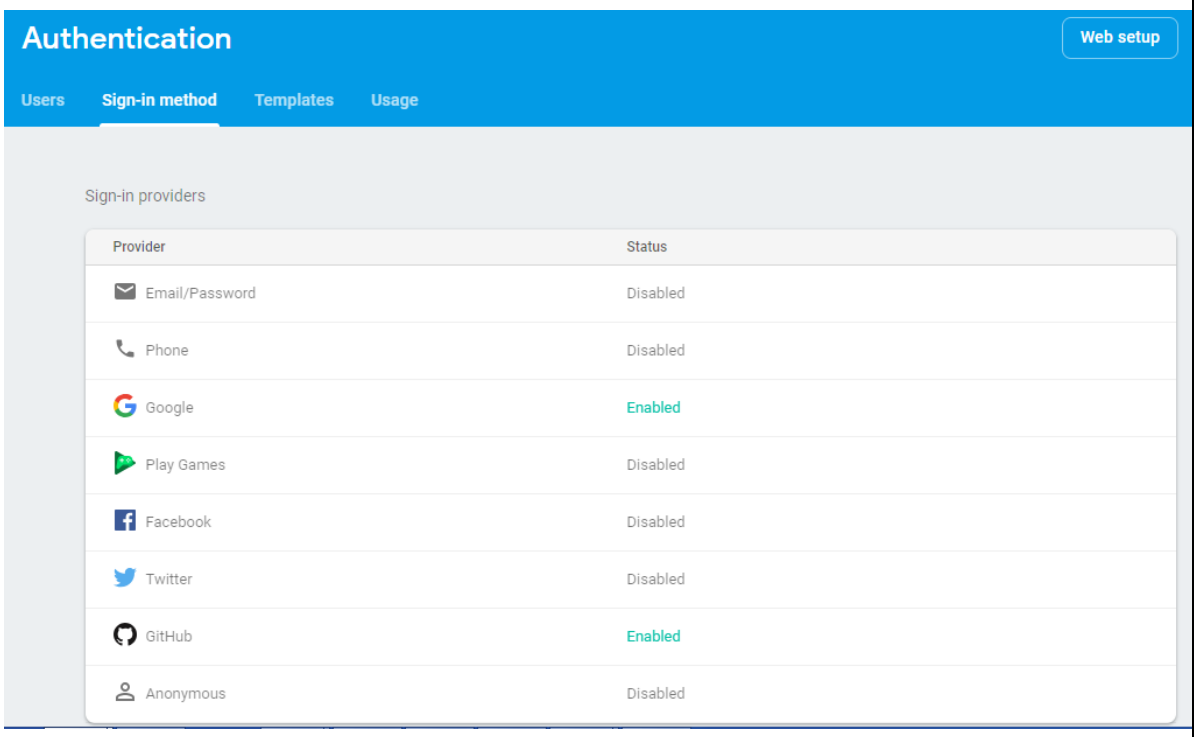

Figure 4.4 Most populate OAuth Authentication providers that firebase supports

Figure 4.4 lists various sign-in methods that enable and make it possible to log into the system. In this implementation, data owner will use Google Account for logging in as a user for the application. For requester both Facebook and GitHub authentication providers will be used. While Experimenting requester actions by logging him from such third-party providers is a perfect analogy with a real word running online system. Any third-party service can register himself as a valid authentication body provided that if it can transmit over OAuth 2.0 protocol.

4.7.2 Data owner's Admin Authority

As stated, data owners reserve a superior role of authentication in cloud system and can view and manipulate existing authentication. They can monitor each authenticated user's activities, their authentication provider's details. They have also right to partially disallow someone to use the system or block a user or the authentication provider itself.

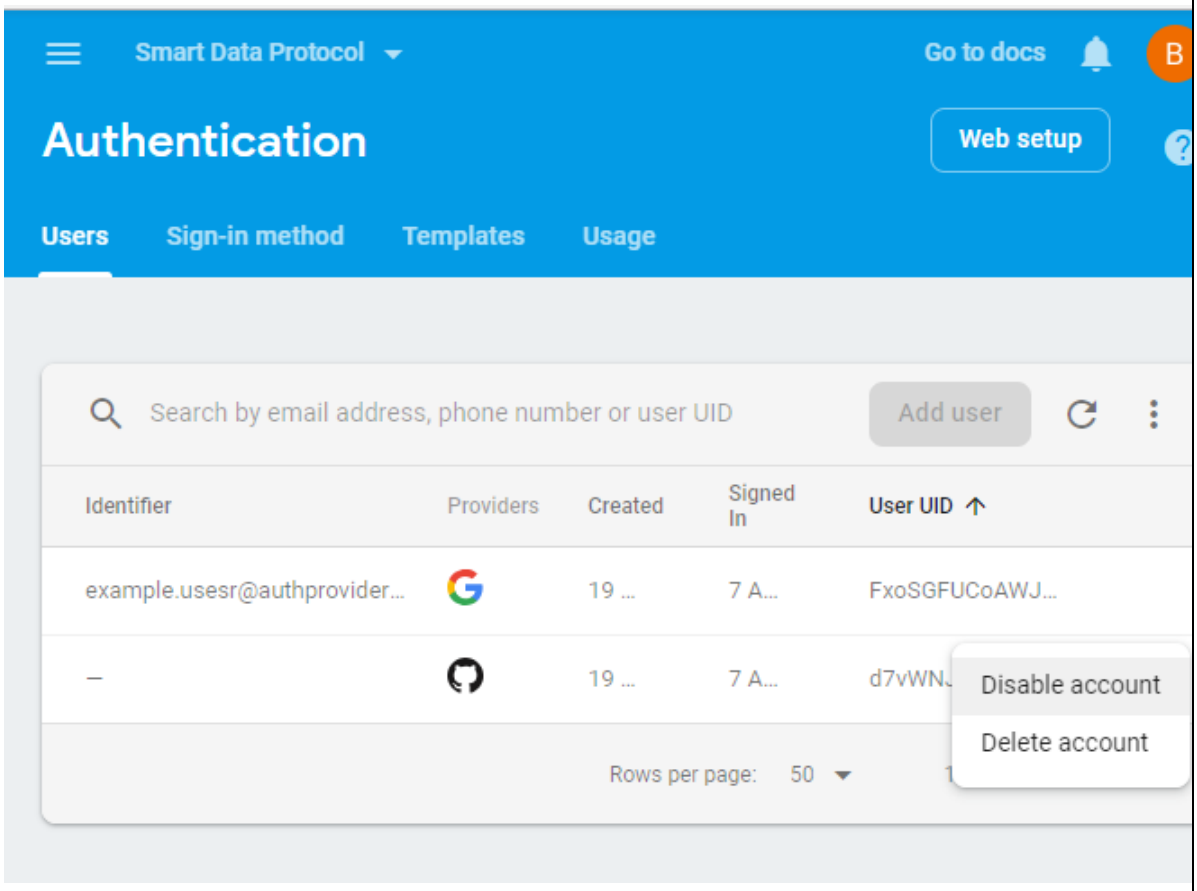

Figure 4.5 Interface available to data owner to control requester authentications.

4.7.3 User Role Authorization

After successful authorization, each user must have proper authorization rules to interact with cloud storage.

The following table describes a set of typical storage access rules that can be enabled or disabled user.

Rule Types	
.read	Describes if and when data is allowed to be read by users.
.write	Describes if and when data is allowed to be written.
.validate	Defines what a correctly formatted value will look like, whether it has child attributes, and data type.
.indexOn	Specifies a child to index to support ordering and querying.

Figure 4.6 Typical storage access rule

The following is a snippet that demonstrates how rules can be scripted using Firebase's rule definition language:

```
// These rules grant access to a node matching the authenticated
// user's ID from the Firebase auth token
{
  "rules": {
    "users": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

Figure 4.7 snippet that demonstrates how rules can be scripted using Firebase's rule

4.8 Gateway

Gateway module is the one which administers the whole communication and ensures that the proposed protocol is followed for each communication. Firebase, an already built cloud service is used in this implementation, but Gateway module drive the whole flow in a manner which facilitates the protocol. In principle, data owner and requester cannot communicate with

Firebase without first interacting with Gateway. Data owner can only bypass Gateway when accessing Firebase console via admin authentication.

Basic responsibilities of Gateway includes providing encryption/description, launching clients for actors, loading Firebase SDKs for each client, configuring URLs for clients, reading events from Firebase logs and registering clients for Firebase messaging to ensure timely notifications.

4.8.1 Access Rules Component

Access Rule Component or Access Rules Server is a sub-component of the Gateway module. It presents itself as a ledger where each actors' access rules, privileges, and permissions are stored. This component is not responsible for storing authentication information. In this implementation, the Firebase Real-Time database is used to simulate the behaviour of Access Rules Component. On each request, Access Rules Component first queries Real-Time database to confirm is the requester is allowed to access a piece of particular information.

On the basis of the defined access rule, component gateway identifies the authenticated user and then they are allowed to access the URLs

4.8.2 Authorization Server

Authorization Server is responsible for decoding security ticket to identify the sender's role. It also describes the accessibility of the users. Answers the question of when from where and how a user can access and which resource. This implementation uses Firebase's Rule Scripting facility to meet the needs of the Authorization server. It is worth to mention that it is not responsible for authentication mechanisms.

4.8.3 Operation

Gateway module is the heart of this implementation. Practically, the framework process starts from here. The process can be a breakdown in the following steps

Launching	1. Gateway provides a script which can be run from a software interface or command line interface. This is called the launch script. 2. The launch script initializes a web server component. 3. Web server component mainly deals with major endpoints. One is for the owner while another one is for the requester. Note that, instead of opening two different endpoints, Gateway initializes two different instances of the server on different ports. 4. Launching of two different endpoints or ports establishes a server infrastructure.
Clients	5. After establishing a server, Gateway launches clients. 6. To launch clients, gateway invokes Chromium Embedded Framework provider, **cefpython3** which then launches web browser's two new sessions. 7. Client opened for data owner is served with application i.e. for uploading media While client opened for requester is served with information consumption code.
Firebase	8. The server fetches Firebase JavaScript SDK from google cloud and loads it onto clients. 9. Clients are loaded with the SDK and interfaces initialized for authentication. 10. Gateway loads Firebase Python Admin SDK to read events and to notify clients.

The following Figure 4.8 summarizes the position of Gateway in a framework and the initialization of clients for each actor.

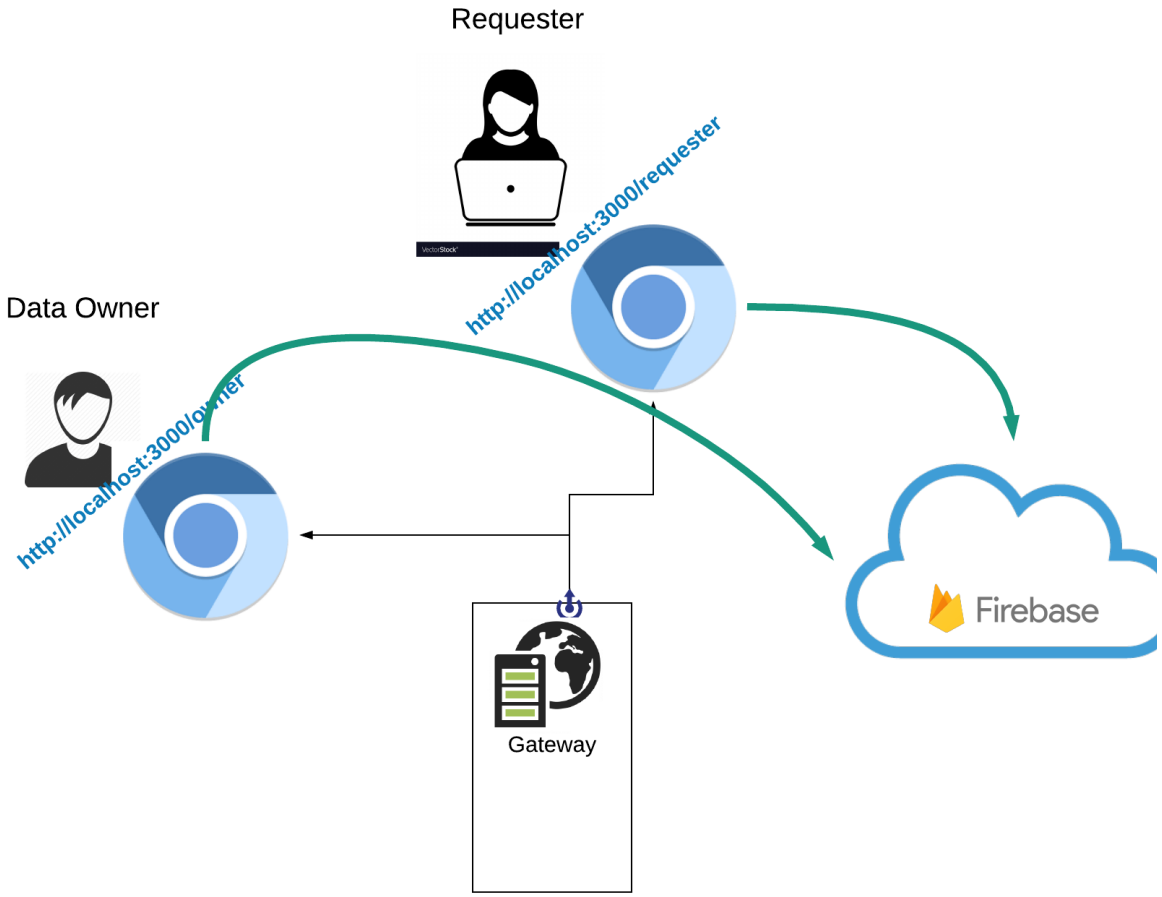

Figure 4.8 figure summarizes the position of Gateway in a framework and the initialization of clients for each actor.

4.9 Actor Operations

With complete actor authentication, gateway pushes interfaces onto their clients. Data owner gets an interface where documents or images can be uploaded. Whereas requester is presented with a list of available documents or images that owner has uploaded. Here's the summary of steps that took place in a typical workflow:

1. After authentication, Gateway connects both owner and requester to Firebase, according to their authorization roles.
2. Requester gets authenticated, and its clients get connected with Firebase to load information with the help of Firebase JavaScript SDK.
3. Initially, Real-Time database and Firebase Storage is empty.

4. With the help of JavaScript SDK, owner gets authenticated, and its client gets connected with Firebase to populate information
5. Owner finds UI controls to upload media.
6. Owner initiates the request to upload media.
7. Gateway, with the help of a web server, gets the information that the owner wants to upload.
8. Gateway encrypts the media which is being used by owner's authorization ticket.
9. Gateway logs the record for the media, in Firebase Real-Time database.
10. Access Rules Component registers the logged record against the owner by querying authorization ticket from authorization server (from Firebase).
11. Access Rules Component applies default access rules on the newly created record. By default, access rules component disables both read and write access permissions for requesters.
12. On successful creation of the record, Firebase fire event of information gets updated.
13. The gateway receives notification of a successful information update.
14. Gateway locally logs the event.
15. The gateway connects the Firebase Storage Service.
16. The gateway sends the encrypted media to Firebase storage while associating the logged record.
17. Firebase applies its cloud encryption and stores the media.
18. Firebase fires event for a successful media upload.
19. The gateway receives the notification and locally logs the event.
20. Gateway notifies owner's client and UI controls to upload media which followed to reset for another upload.
21. Gateway notifies requester client for information update.
22. Requester client with the help of Firebase JavaScript SDK listens for new information.
23. Information about new resources is fetched from Firebase and got listed on the requester interface.
24. Requester tries to access the new information by passing a user certificate.
25. The gateway receives a request from a requester to access information.
26. Gateway confirms requester certificate by CA to verify if requester's certificate is valid.
27. Gateway confirms by access rules component to identify whether this record is allowed for read access.
28. Access rules component grants or rejects read access by passing requester's certificate along with a resource id (which uniquely identifies each resource) to CA.
29. CA verifies the presence and validates public keys and expiry of both requester and resource certificates. Based on the validity of both certificate, CA responds access rules component with a success or failure flag.
30. Access rules component uses the flag from CA to grant or reject the read access.
31. If Access rules component denies read access, the Gateway notifies the owner about the failed read access incident and sends a negative notification response on requester's client prompting that read access is not allowed.

32. Owner issues resource certificate for a specific period either from admin console or from owner client application interface.
33. Owner enables read access for the newly created record.

34. Firebase notifies Gateway.
35. Access rules components keep track of the permission update.
36. Requester tries to access the record again.
37. Gateway notifies Access rule component for a new request.
38. Access rule component confirms that read access is allowed.
39. Access rule component verifies resource certificate via CA.
40. Gateway confirms from Access Rules Component that read access is allowed.
41. Gateway fetches the media from Firebase. Firebase sends the media after applying its cloud decryption.
42. Gateway decrypts the media and pushes it to requester client.
43. Gateway locally logs the event.
44. Requester gets access to the media.
45. Access rules component keeps track of this read access operation.
46. Gateway notifies the owner about a successful read access incident.

4.10 Project Details

4.10.1 Summary

To implement the idea of resource certificate base data access protocol, a project structure is implemented which hosts all the software components and scripts that aid in launching the project.

Python and **JavaScript** language are used. Both are high-level, user-friendly languages. JavaScript is an industry leading language, ubiquitous on a myriad of devices and platforms, whereas Python leads in data processing projects and is popular in the scientific community. Though JavaScript is available on both server side and client side, Python's main uses are on the server side.

Interestingly, this project utilizes Python for both as Server-Side language as well as a Client-Side Language. On the server side, Python is used to create a web server by using the Django framework. On the client side, python is used for visual simulation with the help of Pyglet python package.

Clients that actors use are controlled with the help of cefpython3 which is also a useful Python package.

Cloud usage is purely based on Firebase. Firebase Console is used for infrastructure configuration. Firebase Rule Scripts are used to implement framework specific workflow.

The goal of the project was to implement the minimum criterion details, only essential for minimum required protocol definition. However, scalability is highly regarded while setting the foundation of software components.

4.10.2 Directory Structure

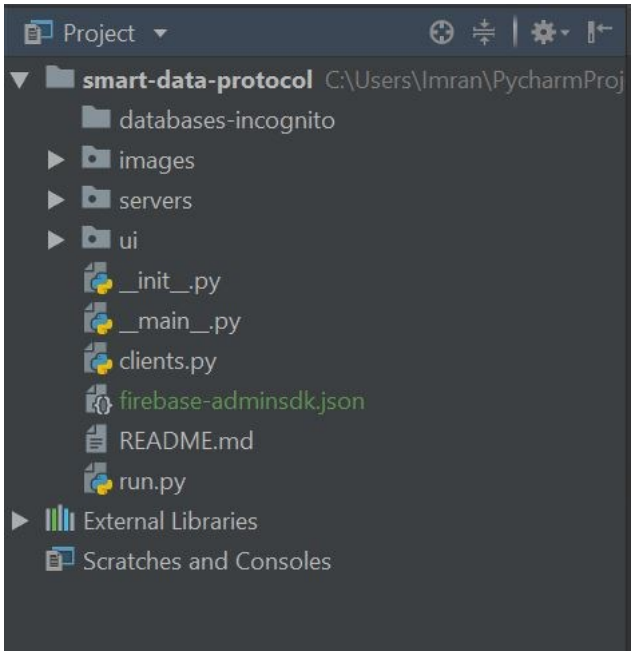

Figure 4.9 top-level directory organization of the project

<*figure*> shows the top-level directory organization of the project.

Here's are the details of the notable top-level directories and project files:

🗁 **Images** directory contains images and artwork, mainly used in the simulation.

🗁 **Servers** directory contains the implementation of the gateway module, web server, Certificate Authority (CA) server, and client applications.

🗁 **UI** directory contains graphics drawing routines and Pyglet simulation.

🗏 ***clients.py*** is the Python script that launches Chromium web clients for actors.

- ***__main__.py*** is the Python script that initiates Pyglet GUI window and objects, for the simulation.

- ***firebase-adminsdk.json*** is the JSON configuration containing API Keys, URLs, and secrets to configure and connect web clients with Firebase cloud with the help of Firebase SDK.

- ***run.py*** is the Python script that consolidates all initialization into a single entry-point, by opening separate threads for the gateway, web server, web clients and simulation. The following diagram demonstrates the operation of the *run.py* script.

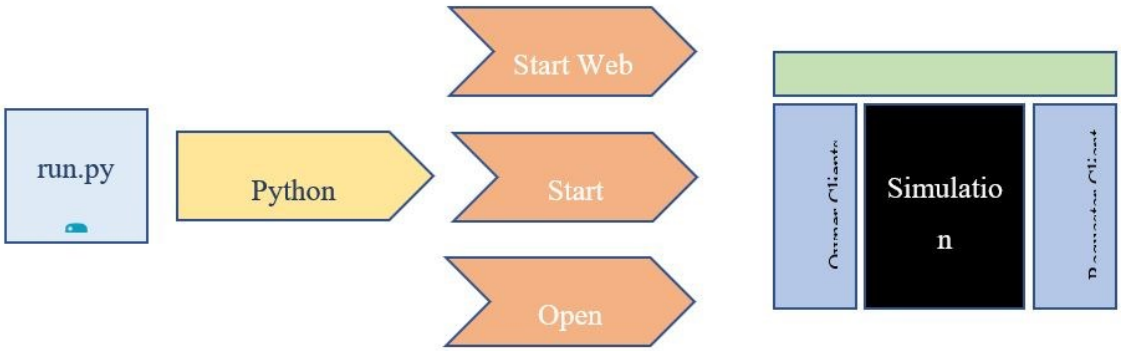

Figure 4.10 Operation of run.py

4.10.3 Python Language, Packages and Versions

Python is an object-oriented and high-level scripting programming which can be used for software application development. It emphasizes on easily readable code. Programs written in Python language require less maintenance time. There are several built-in standard libraries that can be distribute without any charges. [32]

This project uses **Python 3**, specifically **Python 3.6**. There is two active version of Python available, Python 2 and Python 3. Python 3 is chosen so as ensure a long life of the project and to use modern features to aid the architecture. For instance, in Python 3, parameters to a method can be typed, which Python 2 does not facilitate.

The following snippets demonstrate an example snippet for both versions.

Python 3	Python 2

`def __init__(self, icons: List[str] = []):`
`  self.batch = pyglet.graphics.Batch()`
`  self.sprites = []`
	`def __init__(self, icons):`
	`  self.batch = pyglet.graphics.Batch()`
	`  self.sprites = []`

Figure 4.11 Python versions

For Python package installation, **pip3** is used which comes bundled with Python interpreter installation. Pip3 in a python package manager, which is used to install Python package dependencies (and packages' own dependencies, if any) for the project.

Here's an example command to install **firebase-admin** python package:

```
pip3 install firebase-admin
```

There are four main Python packages used in the project:

1. **Django**: To set up a web server to load clients with necessary scripts and to send and receive messages and data.
2. **pyglet**: For drawing and animation to simulate each protocol related activity.
3. **cefpython3**: To launch Chromium Windows for data owner and requester.
4. **firebase-admin**: To connect gateway app to Firebase, for data communication and messaging.

Figure 4.12 lists Python interpreter, packages installed with their dependencies and versions of installed packages and their dependencies in the project.

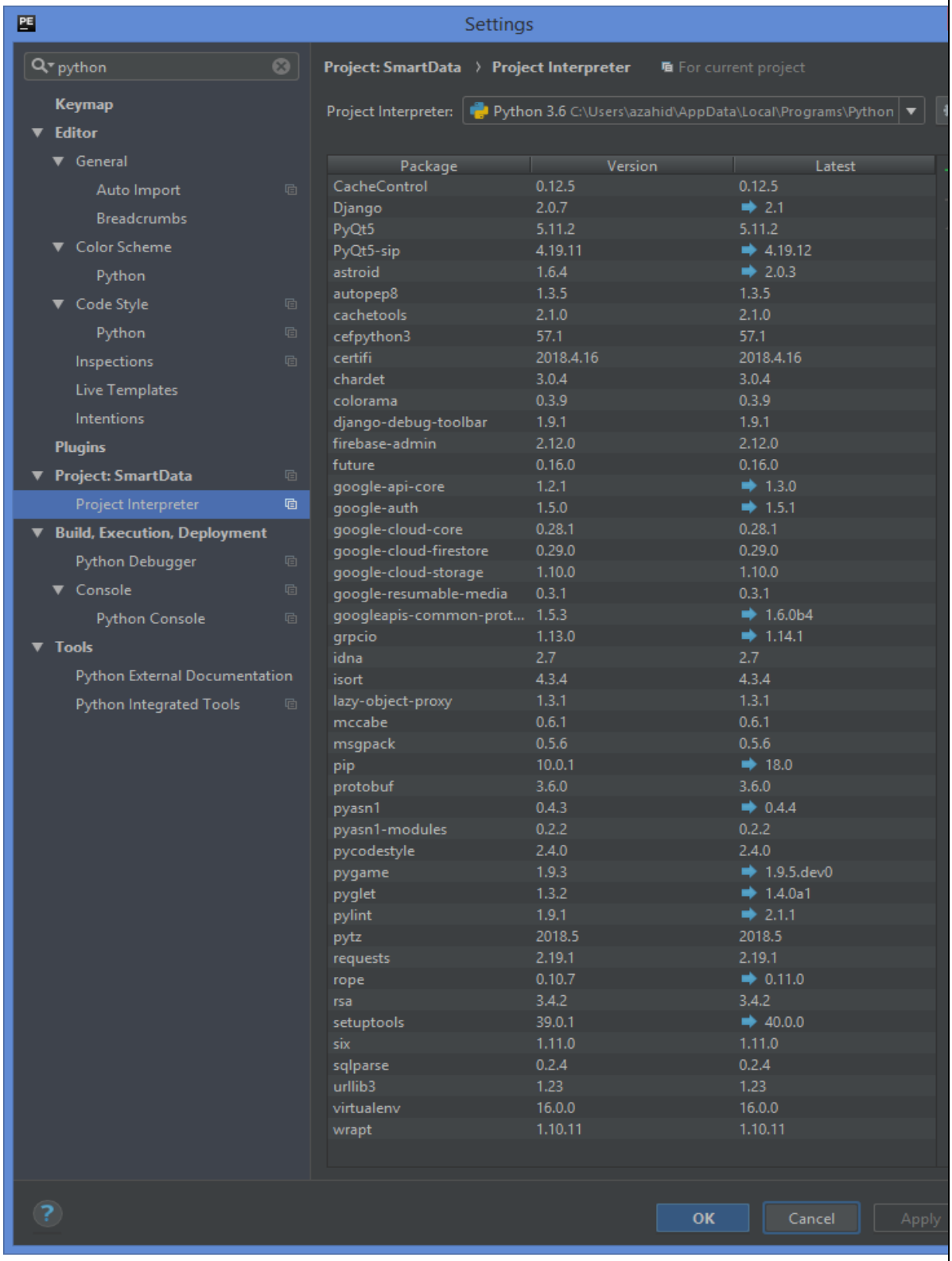

Figure 4.12 python packages

4.10.3.1 “servers” Directory

Figure 4.13 lists the top-level directories contained within “servers” directory:

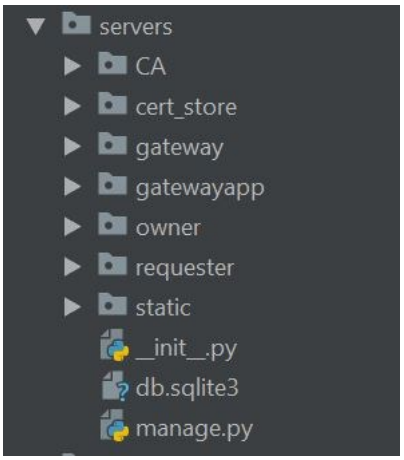

Figure 4.13 Server directory

This directory contains packages and files that form a web server environment, which effectively lays a foundation for our Gateway module. These are all the files that hold the implementation for the Gateway and Server responsibilities.

The “servers” in a Django environment, state web server, and gateway functionality is carried out with the help of the famous python web framework, Django.

Here’s the purpose of each directory and file under “server” directory.

🗁 **gateway** this is the “site” directory of a usual Django framework project. Django framework has an app-based architecture. For example, to implement a server part for the owner’s client, a separate app is needed to serve the needs of owner client which in our case sits in the **owner** directory. **gateway** is the folder to contain the default app that is created while scaffolding a Django project and it is also the entry point for a web application while the project runs. When a Django project is created and uses project’s name then by default, this directory gets generated.

🗁 **gateway app** directory is a Django app to act as a gateway and meet any gateway responsibility to the protocol needs.

🗁 **owner** holds implementation of a Django app, serving the needs that an owner client needs from a server to execute the proper operation.

🗁 **requester** directory contains the implementation of an app that wraps functions that a requester client would require in order to perform its user activities while by following the protocol.

🗁 **static** directory contains JavaScript code files needed to provide an interactive user interface and connective to Firebase.

🗏 ***__inti__.py*** is the Python script file that makes servers directory as a valid Python Package. It is generated by default when a Django project is created.

🗏 ***db.sqlilte3*** is the file generated by default with a Django project. It is not used in this implementation.

🗏 ***manage.py*** is the Python script generated by default with the Django project. This file contains scripts that help the user to run administrative and management commands from the command line, i.e. it is Django-CLI. One of the commands is `runserver` which starts main Django "site" app (**gateway** in our case) as well as other apps.

django Site Structure

Django project creates a directory with the same name as a project ("gateway" in our case) to keep project-wide web server configuration and settings. The following snippet lists children files of **gateway** directory:

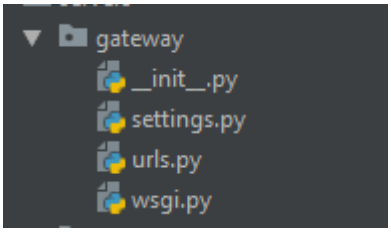

Figure 4.14 Files of Gateway

🗏 ***__inti__.py*** is the Python script file that makes "'gateway" directory a valid importable Python Package.

- ***settings.py*** is the Python script file containing all the settings and configuration of the project needed to properly set up the server, register apps and register app URLs to provide an opportunity for clients to connect with Django apps, thus establishing a client-server model.

- ***URLs.py*** is the Python script file registering all the URLs that this web server can be approached externally and registers each app against an inbound URL that the app is supposed to respond against.

- ***wsgi.py*** is the Python script that exports an instance of a runnable web application. It is this file that makes the project able to start a runnable server for a provided host and on a specified port.

4.10.3.2 django App Structure

The specific implementation flow needed in the project is implemented in individual sub-modules, called Django apps. For instance, while inside "**servers"** directory, the following command can be used to generate "owner" app:

```
python manage.py startapp owner
```

Figure 4.15 lists files and directories contained in **owner** Django app:

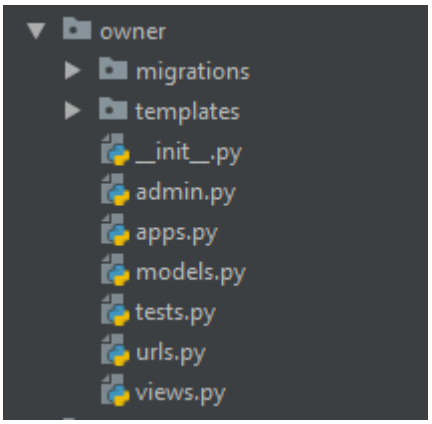

Figure 4.15 owner Django app

🗁 **migrations** is the directory which contains app specific database migrations. Since this implementation does not use a local database, this directory and its files are not used and are only generated by default.

🗁 **templates** directory has template files – HTML files which layout user interface. They define everything that the owner should see when his client is launched.

🗏 ***__inti__.py*** is the Python script file that makes servers directory a valid Python Package. It is generated by default when a Django project is created.

🗏 ***admin.py*** is the file generated by default with a Django app. It is not used in this implementation.

🗏 ***apps.py*** registers the app with the Django site.

🗏 ***models.py*** contains database models. Not used in this implementation.

🗏 ***tests.py*** Not used in this implementation.

🗏 ***URLs.py*** registers URLs that this app should respond to and also defines the corresponding views as according to the URL.

🗏 ***views.py*** has methods which define app's View Controllers. A View Controller is responsible for receiving request on an URL, read passed information in URL or request body, process information, prepare a response and send the response back to the client while rendering the response in a defined template**.**

4.10.3.3 "static" Directory

Static files are those which are not executed at the server side. i.e. they are not executed by python interpreter and considered as static resources on the server side. These files are aimed to be pushed to web clients on demand.

"static" directly contains all the JavaScript files that are part of any module.

Figure 4.16 lists JavaScript files under "static" directory:

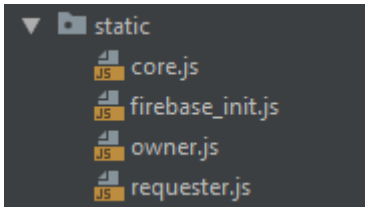

Figure 4.16 static directory

- ***core.js*** provides functionality to owner and requester web clients. It allows communication between the **gateway** and Firebase cloud. It also enables to send and receive messages, information, and media. This file also defines what type of message each client is allowed to send to cloud or gateway.

- ***firebase_init.js*** helps both clients to initialize a connection to Firebase for authentication and communication. This file contains secret keys, cloud URLs and other configuration essential. It enables successful connection with Firebase Real-Time database, Firebase Storage and Firebase Authentication services.

- ***owner.js*** implements the behaviour which defines how an owner interacts with the user interface, available on owner client and performs actions. For instance, it helps the user to authenticate or to choose media from the machine for upload.

- ***requester.js*** implements the behaviour which defines how a requester will interact with his client user interface and performs actions, on the request of the client user. For instance, it helps the user to authenticate or request for viewing or modifying an item uploaded by the owner.

4.10.3.4 "UI" Directory

"ui" directory holds files which implement a simulation for the project. Simulation has graphical objects, which is defined to represent actors and nodes for each module as well as animations. It represents communication between modules and activities. Simulation window also displays cloud events, that are recorded in the cloud to track protocol-specific actions.

Simulation is implemented with the help of **pyglet** a Python package. "ui" directory contains scripts which heavily uses **pyglet** package.

The following figure lists files contained within “ui’ directory:

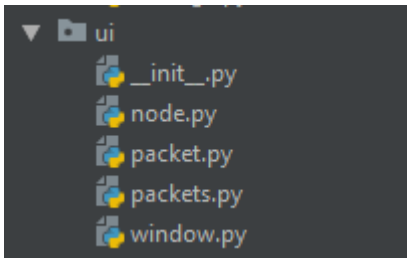

Figure 4.17ui directory

- ***__inti__.py*** is the Python script file that makes ui directory as a valid Python Package so that it can be imported in other project modules. It enables package to be imported by the project’s *__main___.py* script while during launch the simulation.

- ***node.py*** is the Python script file containing implementation and behavior of a node. In this project nodes are as under:

a) Owner
b) Requester
c) Cloud
d) Gateway
e) Authorization Server
f) Access rules component
g) Certificate Authority (CA)

- ***packet.py*** is the Python script file, contains definition and behaviour of a transmittable packet. The simulation shows packets with icons, depicting the type of information being transmitted.

- ***packets.py*** is the Python script which instantiates all types of packets which required by the simulation. Packets are transmitted from one Node to another. This transmission is simulated by animating a packet from source Node to the target Node.

- ***window.py*** is the Python script which initializes a window with required properties to run the simulation. This window is among the three windows launched when the project is run via the **run.py** script.

4.10.4 Certificate Authority (CA) Implementation

CA module holds implementation for a Certificate Authority component of the system which consists of the following directories and files:

🗁 **CA,** this directory contains the python implementation of a system component Certificate Authority (CA). It is responsible for generating, validate, create and remove RSA certificates for own memory, requester and Resources in RSA_x509 format.
First CA creates a foundational certificate for its memory and then uses the public key of this certificate to base requesters' and resources' certificates.
A requester's certificate is generated to validate whether a requester has valid access rights to all the resources whereas a resource's certificate is generated to validate whether a requester has a valid (available and non-expired) access to a specific resource.

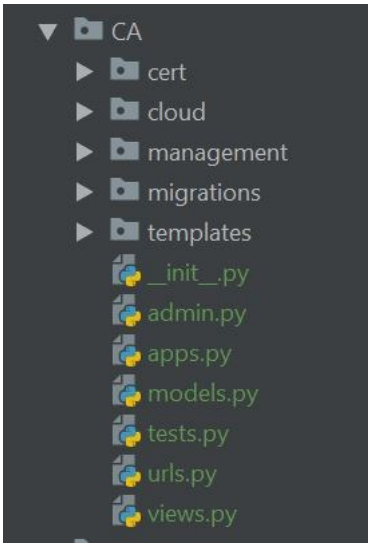

Figure 4.18 Certificate Authority (CA)

🗁 **cert_store,** ***this*** directory holds all the generated certificate files, and CA will always look in this central Certificate Store for presence and validity checks of requester or resource certificates.

🗏 ***cert/cert.py*** is the Python script file that serves as a helping interface to create and validate certificates.

- ***cert/certgen.py*** contain the library methods that directly access PyOpenSSL for certificate generation and validation.

- ***cert/validate_cert.py*** is the file that too directly accesses PyOpenSSL methods to verify a certificate chain. A certificate chain is a stack of the certificate with a CA certificate at the bottom, then requester certificate followed by a resource certificate at the top.

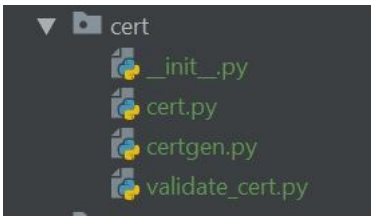

Figure 4.19 cert directory

- ***cloud/cloud.py*** this file provides interface methods to CA to access the Firebase logging mechanism. Whenever a certificate is generated, successfully validated or rejected, CA logs events in Firebase log with the help of these interface methods.

5 Chapter 05: Conclusion

In this thesis, a literature review of different access control framework has been studied to protect the cloud environment from cyber-attack and unauthorized access. These frameworks include open's personal cloudlet, information flow control from apps, the framework of xAccess, Active bundle, and user-centric access control policy. These access control policy frameworks have been investigated from different aspects including integrity, authentication, and non-repudiation.

As a solution of unauthorized access to a resource at cloud environment, the proposed, designed framework empowered the user and gave authority to issue a resource-based certificate to control unauthorized access to the resource. Owner of the data is allowed to have full control over their resource according to their security requirement and for a specific period. The designed framework provides trust between users and protects the resource from unauthorized access and other security threats. Moreover, a resource certificate is only assigned to a single user with a unique ID and is always verified by a third trusted party CA, to minimize the chance of leakage of data. Also, the data owner can directly access real-time database by signing in as admin user where he can define, edit and manage all the access rule on their resource.

5.1 Future work

This report opens many different scenarios for access control Rule. As the proposed framework is very general and is implemented in a real environment, so it can be fitted to any framework especially to the new idea by Farhoud [5] called smart data. The designed framework can easily be implemented as an access rule for smart data while assigning 'gateway' and 'User AC Rule component' as one module in smart data. On a request to access the smart data, the smart data will request the owner to issue or verify the certificate and then can access the smart data. Additionally, the proposed framework can be fixed for the Active bundle by othmanie, and different other protocols.

References

[1] Stanford University. and Center for the Study of Language and Information (U.S.), "Privacy and Information Technology." Stanford University, 2014.

[2] L. Ben Othmane and L. Lilien, "Protecting privacy in sensitive data dissemination with active bundles," *Congr. 2009 - 2009 World Congr. Privacy, Secur. Trust Manag. E-bus.*, pp. 202–213, 2009.

[3] "VNI Global Fixed and Mobile Internet Traffic Forecasts - Cisco." [Online]. Available: https://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html. [Accessed: 08-Jul-2018].

[4] "privacy Definition from PC Magazine Encyclopedia." [Online]. Available: https://www.pcmag.com/encyclopedia/term/49709/privacy. [Accessed: 08-Jul-2018].

[5] F. Hosseinpour, J. Plosila, and H. Tenhunen, "An approach for smart management of big data in the fog computing context," *Proc. Int. Conf. Cloud Comput. Technol. Sci. CloudCom*, pp. 468–471, 2017.

[6] D. McCarthy *et al.*, "Personal Cloudlets: Implementing a User-centric Datastore with Privacy Aware Access Control for Cloud-Based Data Platforms," *Proc. - 1st Int. Work. Tech. Leg. Asp. Data Priv. Secur. TELERISE 2015*, pp. 38–43, 2015.

[7] T. Pasquier, J. Bacon, J. Singh, and D. Eyers, "Data-Centric Access Control for Cloud Computing," *Proc. 21st ACM Symp. Access Control Model. Technol. - SACMAT '16*, pp. 81–88, 2016.

[8] K. Singh, I. Erete, and W. Lee, "I Own , I Provide , I Decide : Generalized User-Centric Access Control Framework for Web Applications," 2010.

[9] A. Crowell, "A Survey of Access Control Policies," 2011.

[10] V. Beal, "What is Mandatory Access Control? Webopedia Definition." [Online]. Available: https://www.webopedia.com/TERM/M/Mandatory_Access_Control.html. [Accessed: 09-Jul-2018].

[11] "Check Your Accounts: Timehop, Macy's, Bloomingdale's, Domain Factory Announce

Breach - Security News - Trend Micro USA." [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/check-your-accounts-timehop-macy-s-bloomingdale-s-domain-factory-announce-breach. [Accessed: 10-Jul-2018].

[12] V. Chellappan and K. M. Sivalingam, "Security and privacy in the Internet of Things," *Internet Things Princ. Paradig.*, vol. 8871, pp. 183–200, 2016.

[13] "Information Security and Privacy | OpenText." [Online]. Available: https://www.opentext.com/products-and-solutions/business-needs/information-governance/ensure-compliance/information-security-and-privacy. [Accessed: 13-Jul-2018].

[14] Z. H., H. A., and M. M., "Internet of Things (IoT): Definitions, Challenges and Recent Research Directions," *Int. J. Comput. Appl.*, vol. 128, no. 1, pp. 37–47, 2015.

[15] R. Minerva, A. Biru, and D. Rotondi, "Towards a definition of the Internet of Things (IoT)," *IEEE Internet Things*, no. 1, p. 86, 2015.

[16] M. Shubhi, "What exactly is Internet of Things (IoT)? - Quora," *2016*. [Online]. Available: https://www.quora.com/What-exactly-is-Internet-of-Things-IoT. [Accessed: 16-Jul-2018].

[17] "Definition of Information Security." [Online]. Available: https://www.cmu.edu/iso/aware/presentation/tepperphd.pdf. [Accessed: 14-Jul-2018].

[18] "ch01.indd." [Online]. Available: https://cryptome.org/2013/09/infosecurity-cert.pdf. [Accessed: 14-Jul-2018].

[19] M. Rouse, "What is accountability? - Definition from WhatIs.com." [Online]. Available: https://whatis.techtarget.com/definition/accountability. [Accessed: 18-Jul-2018].

[20] R. S. Sandhu and P. Samarati, "Access Control: Principles and Practice," *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 40–48, 1994.

[21] K. Brauer and V. Torvinen, "AUTHENTICATION AND SECURITY ASPECTS in an international multi-user network," 2011.

[22] “Cryptography - What is Cryptography ? Cryptography meaning, Cryptography definition - The Economic Times.” [Online]. Available: https://economictimes.indiatimes.com/definition/cryptography. [Accessed: 19-Jul-2018].

[23] S. T Sampathkumar, “Cryptography full report.” [Online]. Available: https://www.slideshare.net/harpoo123143/cryptography-full-report. [Accessed: 20-Aug-2018].

[24] “What is Public Key Infrastructure (PKI)? | Management and Authentication of Public Key Infrastructure | Thales eSecurity.” [Online]. Available: https://www.thalesesecurity.com/faq/public-key-infrastructure-pki/what-public-key-infrastructure-pki. [Accessed: 10-Nov-2018].

[25] “What is SSL?” [Online]. Available: http://info.ssl.com/article.aspx?id=10241. [Accessed: 10-Nov-2018].

[26] V. Beltran, J. A. Martinez, and A. F. Skarmeta, “User-centric access control for efficient security in smart cities,” *GIoTS 2017 - Glob. Internet Things Summit, Proc.*, pp. 0–5, 2017.

[27] R. Roshandel and R. Tyler, “User-centric Monitoring of Sensitive Information Access in Android Applications,” *Proc. - 2nd ACM Int. Conf. Mob. Softw. Eng. Syst. MOBILESoft 2015*, pp. 144–145, 2015.

[28] M. Msahli, X. Chen, and A. Serhrouchni, “Towards a fine-grained access control for cloud,” *Proc. - 11th IEEE Int. Conf. E-bus. Eng. ICEBE 2014 - Incl. 10th Work. Serv. Appl. Integr. Collab. SOAIC 2014 1st Work. E-Commerce Eng. ECE 2014*, pp. 286–291, 2014.

[29] M. Diener, L. Blessing, and N. Rappel, “Tackling the cloud adoption dilemma - A user centric concept to control cloud migration processes by using machine learning technologies,” *Proc. - 2016 11th Int. Conf. Availability, Reliab. Secur. ARES 2016*, pp. 776–785, 2016.

[30] P. Grace and M. Surridge, “Towards a Model of User-centered Privacy Preservation,” *Proc. 12th Int. Conf. Availability, Reliab. Secur. - ARES ’17*, pp. 1–8, 2017.

[31] I. Iakovidis, “The silver economy,” no. July, pp. 135–140, 2015.

[32] “What is Python? Executive Summary | Python.org.” [Online]. Available: https://www.python.org/doc/essays/blurb/. [Accessed: 17-Feb-2019].