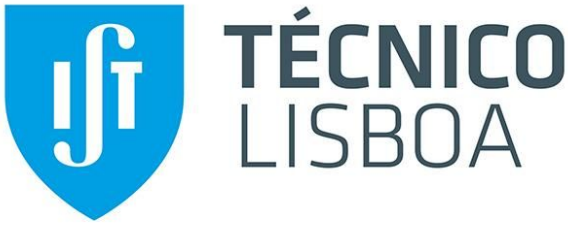

“Security System for Medical Records”

Network and Computer Security, Group A05

Ana Nogueira
82433

Miguel Grilo
86489

Guilherme Rosa
97032

Problem

Medical records constitute highly sensitive information, so it is of the utmost importance to have a secure solution that ensures confidentiality and integrity of the data. To support these properties, one can focus on the use of policies to define who and under which conditions has access to these particular records. One of the challenges is to keep this data private for the responsible staff only while being accessible for other different health care institutions, so that a patient can be assisted anywhere. In addition, the problem demands the usage of a cloud solution for storage of the records which requires for security of the communication channel between user and cloud. Another aspect to consider is to guarantee availability of this data, as these records are essential in helping clinical staff acting in the shortest time possible.

Requirements

The solution has several requirements for the client and for the cloud service.

User requirements:

- As a patient, I must be able to read my own medical records
- As a doctor, I must be able to read/write medical records of my assigned patients
- As a doctor, I must be able to read/write medical records of an unassigned patient if it's and urgent occasion
- As medical staff, I must be able to read patients medical records if I'm in an appointment with that patient
- As admin, I must be able to read/write all medical records
- As a user with a privileged role (e.g. doctor, admin), I must be able to manage who can read or write medical records

Cloud service requirements:

- It must store all medical records data
- It must implement ABAC access control paradigm architecture
- It must receive client requests and reply to them
- It must give access to read/write medical records only to people that have the right privileges
- It must keep a log with all events performed by a user, including date, time, action, and changes made.

Security requirements:

- The cloud service must guarantee the confidentiality, integrity and availability of all data
- As an authenticated user, I cannot repudiate my actions
- Communication between user and cloud service must be done through a secure channel
- User must authenticate themselves before using the cloud service

Trust Assumptions

- We trust there will be a master password which will not be tampered with
- We assume partial trust in registered users (clients, staff, doctors): they will not exploit vulnerabilities/perform complex attacks on the system, but they might try to access records that they should not.

Proposed Solution

Our solution will use, as suggested, the XACML policy language for implementing access control, relying on the ABAC paradigm - this will allow for finer granularity and control over each policy decision, and will allow for different use cases and contexts. At its most complex level, the architecture of the solution will have N clients and two cloud servers (one for backup). The application client will interact with an interface that allows for logging in as a particular user/role. The policy decisions will be enforced according to that particular user's permissions and current context.

Deployment

We will implement the solution in Java, using Apache Maven for the building and management of the project.
For the XACML model, we will rely on the open-source Authzforce framework for the Policy Decision Point (PDP). To simplify the implementation of policies, we will use the ALFA pseudocode language (using an Eclipse plugin).
We will also use Spring and our databases will be set up using MariaDB.
To communicate with the database JDBC will be used.

Secure channel

The communication between the client and the server will be done through a TLS channel to ensure its security.

Plan

Versions

Basic solution:
One client can connect to a cloud server that stores the medical records in a database.
The client uses one VM and the server another one. Both will communicate inside the same private network.
The client has to authenticate before doing any requests. There will be a second database with hashed and salted user passwords for the authentication process.
The server will, at this point, have a partial implementation of XACML, including the Policy Decision Point (PDP) and the Policy Enforcement Point (PEP). The following flow must be respected:

- A client makes a request for a medical record
- The request goes to Policy Enforcement Point (PEP) that generates an XACML authorization request

- The authorization request follows to PDP that evaluates it with the policies. If the decision is “Permit” the server must contact the DB and request the medical record, then a response to the client is created with it
- If the decision is not “Permit” the client must be notified
  (check figure 2)

The communication between the client and the cloud service will be protected using SSL/TLS protocol (check figure 3).

Intermediate solution:

The remaining parts of the infrastructure needed for XACML will be in place and some policies and use cases can be developed.

There will be a third database containing the user’s information/roles for the PIP module to give information regarding the context of the request, to the PDP. (check figure 2)

The network architecture will now change. In this stage the user is going to communicate with the cloud through the Internet (instead of being in the same private network). We must protect our database from attacks such as SQL injection since now that attackers can easily communicate with the cloud.

The server will keep a log with all the actions/requests made, in the PEP module, including the ones performed by doctors and the admin.

At this stage, we will be able to test the functionality of the system and the access control enforced by XACML.

We will also implement our Custom Secure Protocol. In this protocol we will encrypt the medical records with AES cipher. To encrypt with AES we need secret keys - one per medical record. These keys will be stored inside the server on a KeyStore. This KeyStore needs a password in order to get any given secret key. This password will be saved in an extern READ ONLY file (check figure 4).

Advanced solution:

The client has access to an interface for logging-in as an existing user with a specific role (patient, staff, doctor, admin), and where it is possible to view/change medical records, according to that role.

The system has a backup server to guarantee that the clients requests are served when the main server crashes (fault-tolerance). Since we’ll have no control over the database, each server will have one. When the client makes a request to the primary server, after the process is done, it must send an update to the backup server.

The primary server will send from time to time (time that is established) to the backup server an “I’m alive” message, if the primary server crashes this message will not be sent, so the backup server realizes the crash and assumes the role of primary server.

Effort Commitment

Weeks	Ana Nogueira	Guilherme Rosa	Miguel Grilo	Notes
28/10 - 03/11	Explore Alfa and Authzforce API Setup client project	Explore Alfa and Authzforce API Setup server project	Explore Alfa and Authzforce API Setup git repository with Github	Basic Implementation
04/11 - 10/11	Develop Client Create Databases with MariaDB	Develop PEP Develop / configure PDP	Network configuration Start TLS implementation	Basic Implementation
11/11 - 17/11	Set up PDP Develop PEP	Finish PEP XACML policies	Finish TLS	Basic Implementation Testing and fixes Deploy
18/11 - 24/11	Develop logging system Develop PIP	Start custom secure protocol Develop PIP	Develop PIP Develop XACML test cases	Intermediate Implementation Finish fixing defects
25/11 - 01/12	Develop/install KeyStore module	Finish custom secure protocol	Develop/install KeyStore module	Intermediate Implementation Begin testing and fixing defects Deploy
02/12 - 08/12	Develop interface	Develop backup server	Set up communication between server-backup	Advanced Implementation Finish fixing defects
09/12 - 11/12	Finish Interface Finish backup Testing	Finish Interface Finish backup Testing	Evaluate interface attacks Testing	Advanced Implementation Testing and fixing defects Final deploy

Appendix

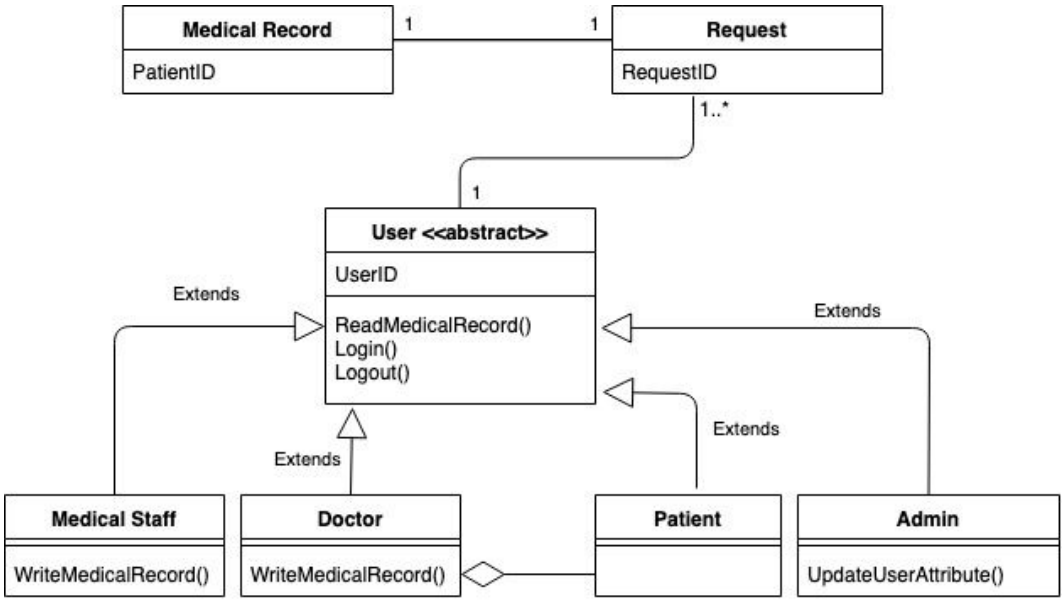

Figure 1 - Domain Model

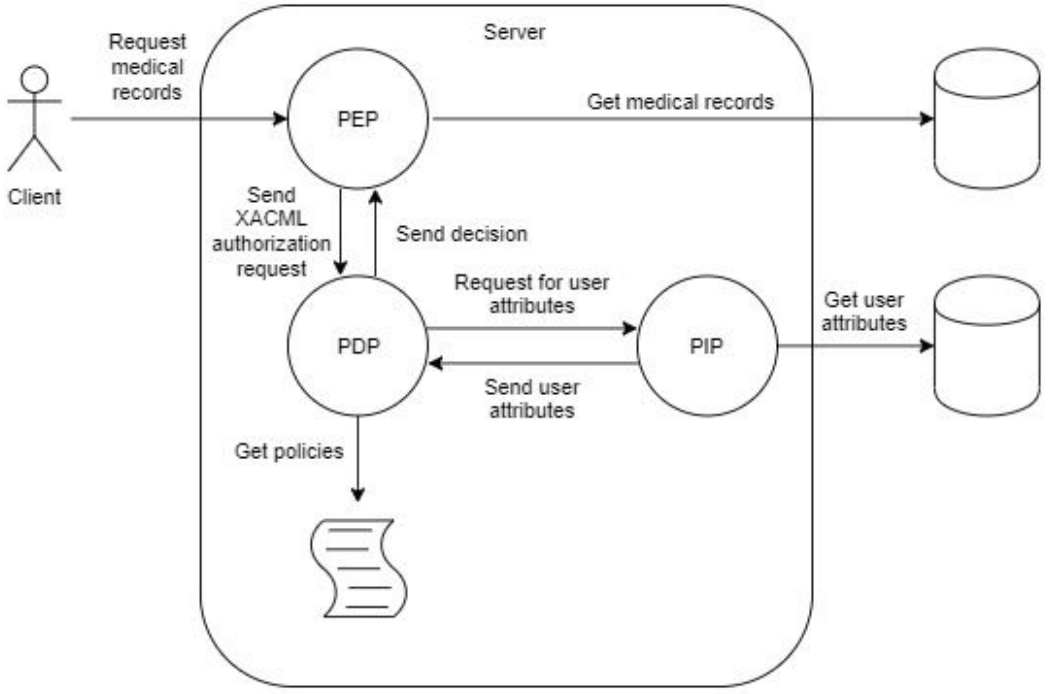

Figure 2 - ABAC Implementation Architecture

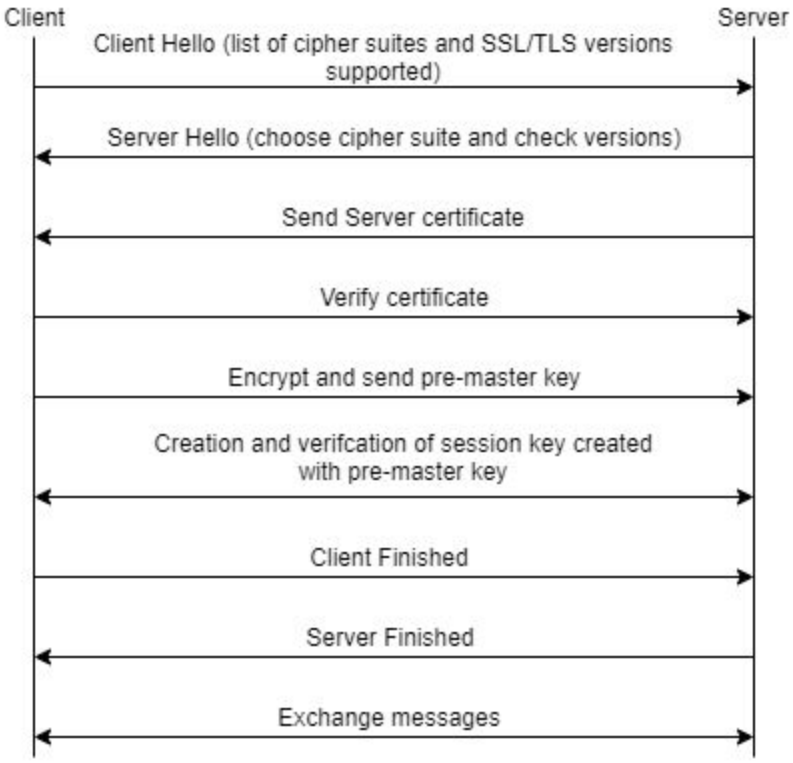

Figure 3 - SSL/TLS handshake protocol

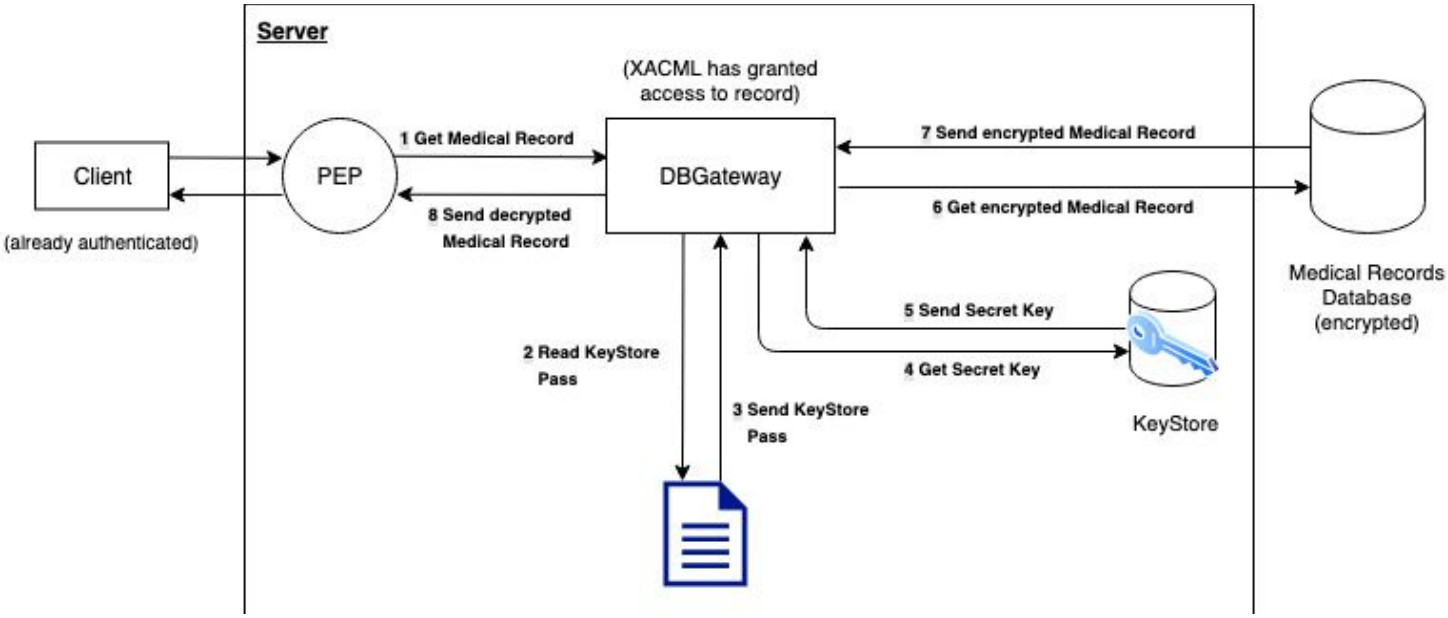

Figure 4 - Database encryption architecture (Custom Secure Protocol)