
Identity Enabled Distribution Control System

UNIVERSIDADE DE AVEIRO

DIOGO SILVA 60337
TÂNIA ALVES 60340

Identity Enabled Distribution Control System

1st Project

Segurança

Universidade de Aveiro

Diogo Silva 60337

Tânia Alves 60340

November 15, 2015

Contents

Context	3
Introduction	4
1 System components	5
1.1 Database	5
1.1.1 Database tables	5
1.1.2 Technologies used	7
1.2 Server	7
1.2.1 Structure	7
1.2.2 Implementation	7
1.2.3 Technologies used	8
1.3 Web page	8
1.4 Player	9
1.4.1 Structure	9
1.4.2 Implementation	9
1.4.3 Technologies used	11
2 System keys	12
2.1 Player Key	12
2.2 Device Key	12
2.3 User Key	13
2.4 File Key	13
3 Server-Client interactions (REST API detailed)	15
3.1 Summary	15
3.1.1 Player - Server	15
3.1.2 Webpage - Server	16
3.2 Checking player integrity	16
3.3 Logging in	17
3.4 Purchasing the file on the web page	18
3.5 Downloading the file from the player	18
3.5.1 Download requisites (validations)	18
3.5.2 Full file download	19

3.5.3	Cryptographic information download	19
3.6	Logging out from the player	20
4	TLS Socket and Certificates	21
4.1	CherryPy Server SSL Certificate	21
4.2	Players SSL Certificates	21
4.3	Browser validation	22
5	Policies	23
6	Installation	24
7	Conclusions	25
7.1	Encountered problems	25
7.2	Future work	25

Context

This project was done for Security, for the 2015/2016 lective year. It aims to create an end to end secure digital rights management system to handle the distribution of video files, music files or books.

Introduction

Our project is made of two main components: the player and the server. The server is in charge of controlling the user access to the protected files. The player requests and plays the files from the server. In order for the user to have access to the titles he/she wants, we also have a web application where the user can buy the titles to play later. To reach the goal of this project, we also needed a database that helped manage the user and file related information.

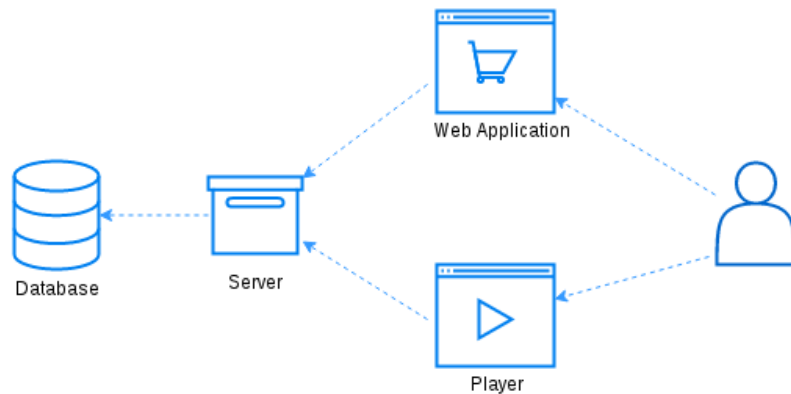


Figure 1: Component overview

Chapter 1

System components

This chapter includes information about the database we used to support the server. We stored information about the users, files, players and devices that were later used.

1.1 Database

For the database we thought of this layout:

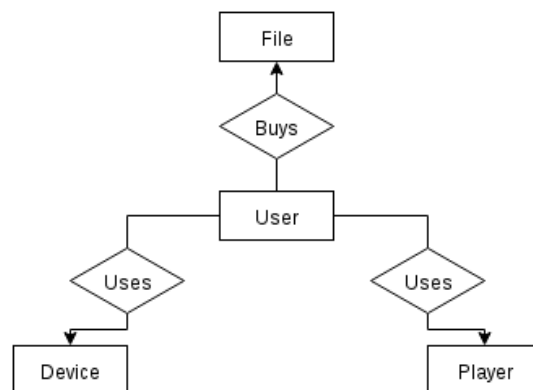


Figure 1.1: DB Schema

We need to save information about the users that belong to the system and buy the files. The files that are stored in the server and are then sent to the players. The players that will interact with the server and play the files. And the devices where the users play the files.

1.1.1 Database tables

Using the information presented, we derived the tables for the database, with all the attributes necessary for our implementation of the system.

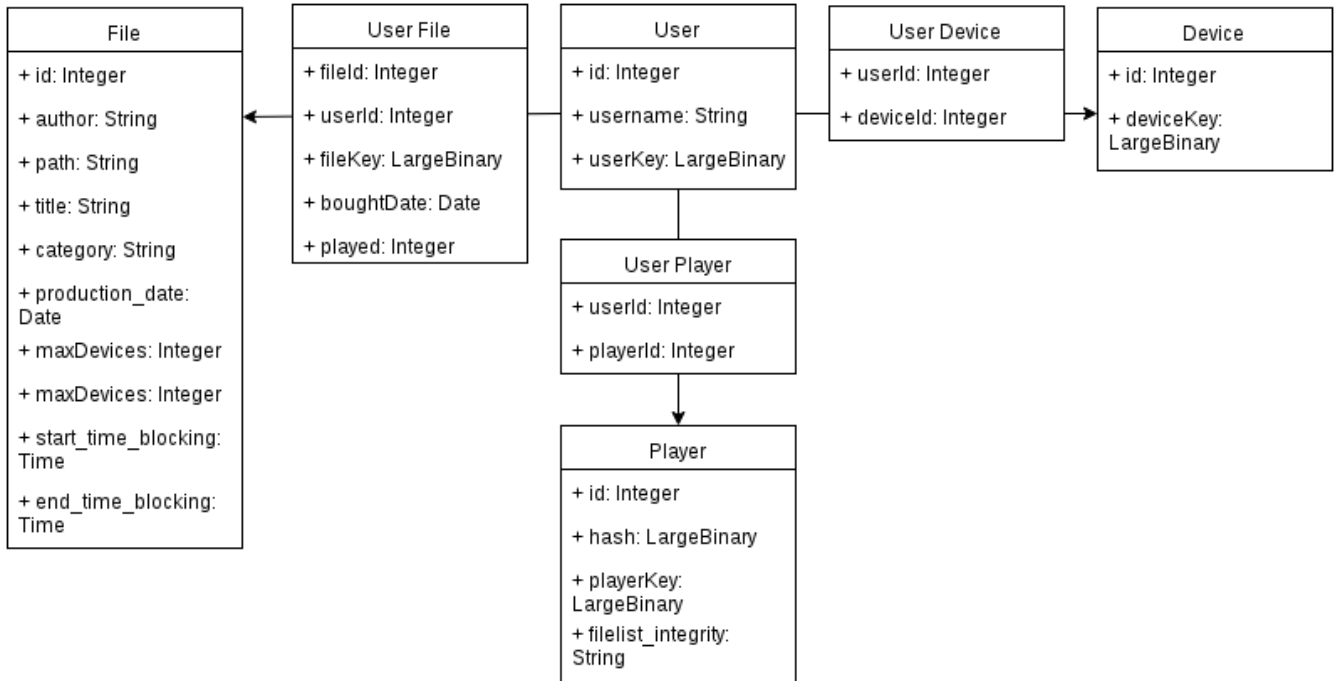


Figure 1.2: DB Tables

Each "main" entities (User, Device, File and Player) will have an id that identifies the entity in the database. We then have the required key for each one of them as well.

In the User table we have added the username field that identifies the each user.

The File table also has some extra fields that provide information to the user about the file he is playing or buying, such as, the title of the file, the author, the category and the date of the title's production.

The actions presented in Figure 1.1 had to be converted to extra tables since, in each case we had a many-to-many relationship. So, we have here the extra tables that store some information of the interactions, like specific keys and additional information like the date that when the user bought the file.

Besides the tables presented before, we needed a few more tables to help implement the authorization policies:

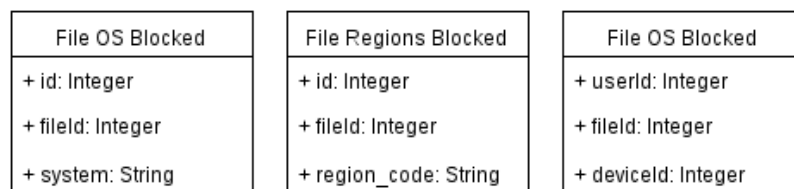


Figure 1.3: DB Auxiliary Tables

1.1.2 Technologies used

For the database we used:

PostgreSQL Open source, object relational database management system. We chose this over SQLite for example because it has a better support for storing secure data.

SQLAlchemy Open source SQL toolkit. Works as an object-relation mapper for Python. This allowed us to create database scripts that created the tables and populated the database.

1.2 Server

The server is meant to interact with the database and control the access of the users and players to the files. This interaction is done over a *HTTP Rest* interface.

1.2.1 Structure

The server is composed of several files:

Server file This is the main file that holds the code to execute the server. When we want to start the server, it must be with *sudo* because server is using port 443 to host (which requires previligies).

Custom adapter file This is the file that overrides the *BuiltInSSL* class developed by *CherryPy*, default adapter doesn't support certificate peer verification and it is useful to check if player is valid and associate it with a determined key.

Cipher file This file holds auxiliary cryptographic functions that are used by the server.

Checker file Holds auxiliary functions that make certain verifications before each request to the *HTTP Rest API*, such as making sure the user is logged in before the player tries to download a file, or check if there's a device key reported when player logged in, etc.

1.2.2 Implementation

The server communicates with the player through *SSL*. We did it like this so that we could focus on other parts of the system instead of implementing it from scratch. For this, we needed to create certificates for both server and players.

1.2.3 Technologies used

CherryPy Object oriented web application framework for Python. We chose this because it is built for rapid development of web applications and offers the basic configurations, that is what were looking for.

psycpg2 package This is a Python-PostgreSQL database adapter that manages context, diagnosis erros and more.

OpenSSL Toolkit that implements SSL and TLS protocols with cryptographic support. We used this so that we could create a certain level of abstraction on how the communications were made, and like that, we could focus in the rest of the system.

1.3 Web page

The web page is where the user can buy the titles. It requires the user to log in before proceeding with the titles purchase. The same API that the player uses is used here.

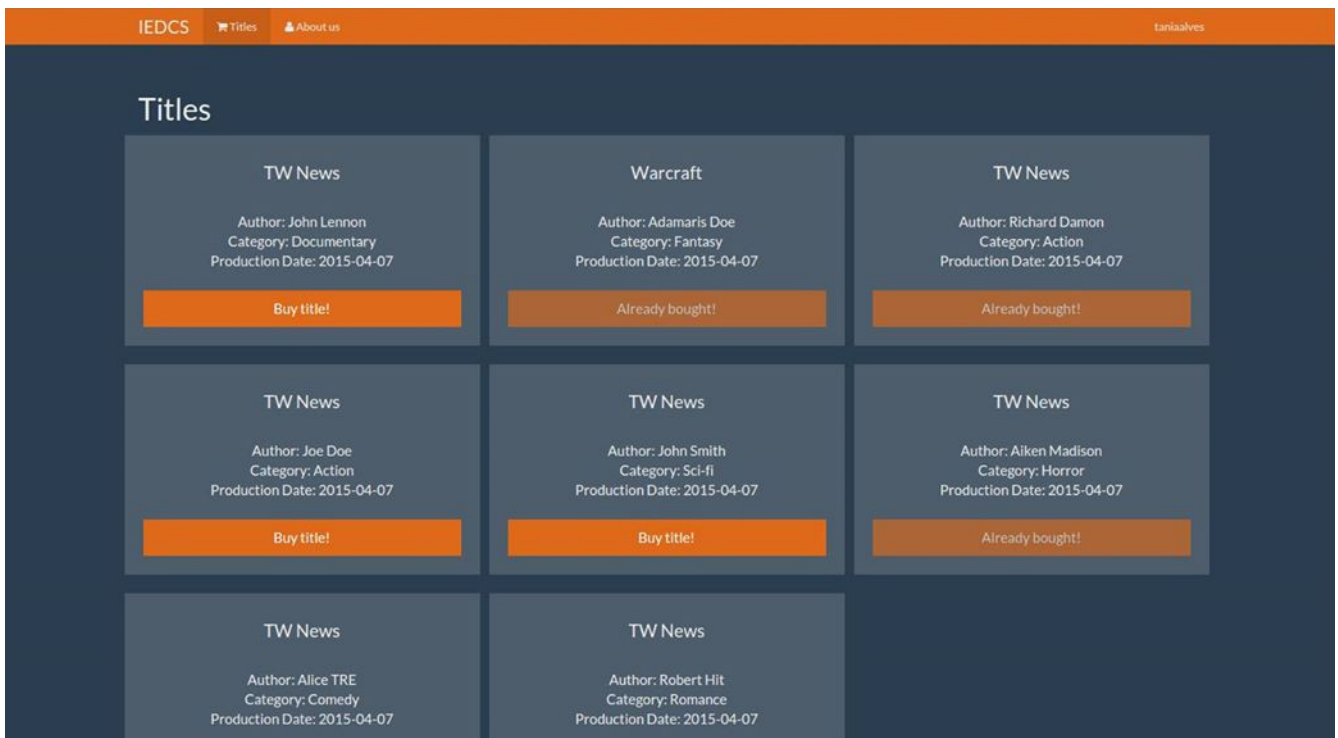


Figure 1.4: Web page interface

It is a simple interface that has a button to buy a specific title.

1.4 Player

The player allows the user to play the files he/she has bought previously. In this implementation we used video files in order to prove that our implementation can handle big files.

1.4.1 Structure

The player implementation we have is made of 3 main files.

Player file This file contains the mainloop for the player's graphical interface. It is here where most of the requests to the server and main operations are made.

Playback file The playback file contains the code that actually plays the file. The decryption is done here, block by block and fed to the thread that runs a VLC player instance..

My List file This file is an auxiliary file that defines the structure of the list that the player displays to the user containing the titles the users owns. This is only to improve the appearance of the list.

videos folder This folder is where the player stores the encrypted videos. Inside, there is another folder for each user that uses the player named after his/her username. This method was chosen so that it is easy for each player to go and grab the files that belong to him/her if he/she chooses to move the files and use another player or device.

Python requests package This package enabled us to build the requests and responses to perform the communication.

PyCrypto package This is the package that helps handle encryptions and decryptions.

1.4.2 Implementation

O player ao garante ao servidor que não foi modificado pelo utilizador e que o servidor conhece a sua implementação, para isso é feito uma validação da integridade do player que consiste numa troca de mensagens com o servidor, na section 3.2 é possível ver esta implementação com detalhe. After the user starts the player application, he/she must log in.



Figure 1.5: Login page for the player

At this stage, the only information required to perform the login is the username. Neste momento para além de ser fornecido o login também é fornecida a device key (item 1) e o certificado do player que garante a sua autenticidade (chapter 4). Further along the road, the login will also be possible using the Portuguese Citizen Card.

After the user logs in and it is confirmed by the server, a list of titles is displayed so that the user can choose which one he/she wants to reproduce.

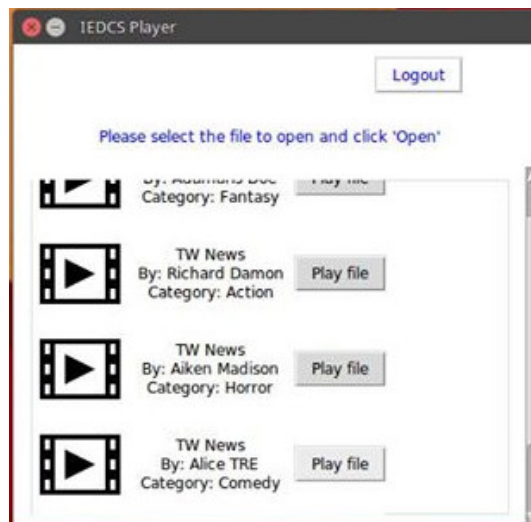


Figure 1.6: Page that lists the files for the user

When the user selects the file that he/she wants to play, a thread running the VLC player is started. The thread reads blocks of the encrypted file,

decrypts it and sends it to the VLC buffer. A obtenção do ficheiro é feita através de uma stream e não através de um simples GET, de forma a poder reproduzir de imediato ficheiros muito grandes, neste momento também é feito várias validações: políticas, existência de certificado do player, integridade do player validade anteriormente, device key enviada anteriormente (Este serviço é falado com detalhe na subsection 3.5.1).

The buffer is managed by the VLC player and is customizable (the user can give the buffer the size it wants by changing the value on the VLC settings).

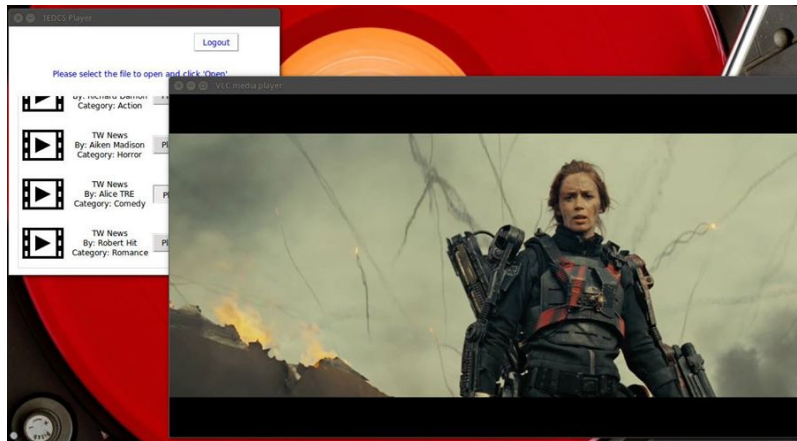


Figure 1.7: Video playing in the VLC thread

All the requests done to the server are performed over the server's *HTTP Rest* API that was presented earlier.

1.4.3 Technologies used

Tkinter We built the graphical interface with Tkinter that is the Python graphic user interface framework.

VLC Player To make things a bit easier when handling video files, we used the VLC player that takes care of the video format encoding.

Python requests package This package enabled us to communicate with the server producing requests and receiving responses.

PyCrypto package This is the package that helps handle encryptions and decryptions.

Chapter 2

System keys

2.1 Player Key

This is a key that must be generated after each code is evaluated by the company that wants to implement the system. This evaluation checks if the player, and the respective code, meet the security requirements. Taking into consideration when it is generated and the process behind it, this key ensures that the player works accordingly to the security policies.

At this stage, the player was generated from a random array of bytes and stored in the database, on the server side, since we are the ones that are developing the player. In the player, the Player Key is hardcoded.

This key is present in the server and in the player. Para além disto ainda é gerado um certificado para o player de forma a garantir a autenticidade do mesmo e poder associar o player em questão à player key, sendo que não é possível reproduzir ficheiros sem certificado do player válido.

2.2 Device Key

The Device Key is a key that identifies the device where the player is being executed. This key should be the same even if we have more than one player running in a device. In the Device Key generation, there were some difficulties in finding the best solution.

We wanted to use the processor serial number because we know that the number we get is already unique. However, this is an information that is hard to get. We came across the command *cpuid* that is used in linux systems, but the Python wrapper available, *Pycpuid*, requires an option to be enabled on BIOS to check the CPU UUID (not everyone has it enabled). We also thought about the MAC Address but it was automatically discarded since MAC Address can be modified easily.

We then turned to the command *dmidecode* that gives us a list of device informations that could be used. Yet again, we found an obstacle. To ac-

cess the processor serial number, we needed to run the player with *sudo* which goes against one of the principles of security ("No entity should have greater permissions in the system other than basic permissions it needs to perform its tasks"). So, with some research we found that the information the *dmidecode* command provides is stored in some system files, specifically, in the `/sys/devices/virtual/dmi/id/` directory. We then browsed all the files present in the directory and came across the one file that didn't require *sudo* to read. This file was the *modalias*, that contained all the information gathered by *dmidecode* that didn't require superuser permissions.

So, for the device key generation, we read the file and hashed the entire content with SHA256, using its digest.

In our solution, this key is generated when the user logs in the player and is sent to the server at this stage. So, the Device Key is present both in the player and in the server.

2.3 User Key

The User Key is a key that identifies the user in the server. In our implementation, this key is generated when the user registers in the web page and is then stored in the database along with the rest of the user information gathered. This key is only ever present in the server, and, as we will see, this will be part of the step that forces the player to communicate with the server each time a file is played.

2.4 File Key

This key identifies the file that the user bought and is independent of the device and player. When compared to the rest of the keys, this one requires a bit more work to derive.

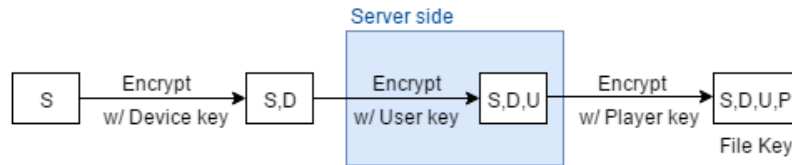


Figure 2.1: Derivation process for the File Key

As we can see from the previous diagram, the File Key is derived from the previous keys presented by encrypting them in a chain of operations. We used the AES algorithm for this.

In the server side, if it is the first file encryption that the server performs for that file and user, we have the following steps:

1. The server generates a random start seed and a random initialization vector since we are going to use CBC to decrypt the file.
2. It then performs the encryption chain presented above to get the resulting File Key.
3. The server then encrypts the file with the result.
4. The encrypted file and the start seed are sent to the player so that it can decrypt the file.
5. The server stores the File Key and the initialization vector (IV) in the database.

If the file has already been sent at least one time, the steps change a bit:

1. The server gets the File Key stored in the database.
2. It then performs the chain of encryption in reverse, decrypting the File Key with the rest of the keys that it has access at that time.
3. The result will be the seed that was used to derive that File Key, and so, it is sent to the player.

This chain allows the player to derive the File Key, even if the device or/and player changes, because the information that the server sends to the player is the starting seed.

In the player side, whatever the File Key is, the operation's chain is the same. However, the blue square present in the diagram above represents a step in which the player must include the server because it does not have access to the User Key.

1. The player gets the seed (and IV used to decrypt file later) from the server and encrypts it with the Device Key.
2. It then sends the result to the server and the server will encrypt it with the User Key, which the player does not have access to (to do this user must be correctly identified, in this case there must be a login associated to the session)
3. The server sends the result back and the player will then encrypt it with the Player Key, resulting in the File Key.

Chapter 3

Server-Client interactions (REST API detailed)

3.1 Summary

Todas as pre-verificações dos serviços encontram-se no ficheiro `checker.py` que implementa um decorador em que indica ao CherryPy todas as verificações necessárias a fazer antes de cada serviço (eg. `@require(logged(), device_key(), is_player(), player_integrity())` para o `download`, isto encontra-se no topo do serviço `GET` do `download`).

3.1.1 Player - Server

A interacção do player com o servidor faz uso de 7 pedidos diferentes ao servidor:

1. `POST /api/user/login` em que o parametro é o nome do username (não existe password tendo em conta que a segunda parte do trabalho é autenticação) em que é associado ao `session_id` actual do utilizador o nome do utilizador em questão, sabendo assim qual o login efectuado durante a sessão toda. Para além disso ainda reporta a device key como parâmetro no login.
2. `POST /api/user/logout` em que remove qualquer associação no servidor ao `session_id` (necessita de login, ou seja, de ter um username associado ao `session_id`)
3. `GET /api/title/user/` em que apresenta todos os titulos que o utilizador comprou (necessita apenas de login)
4. `GET /api/valplayer/` em que devolve um SALT que permite fazer a validação de integridade do player

5. POST /api/valplayer/<hash> utilizando o SALT do serviço anterior tem de chegar a HASH e o servidor vai confirmar se o resultado é correcto ou não, verificando a integridade do player ou não (espécie de desafio).
6. GET /api/title/<pk> recebe um stream do conteúdo do ficheiro (em que pk é o identificador do titulo que precisa de fazer download), esta é a rotina é critica na segurança do sistema porque não pode ser usada se uma das seguintes condições falhar: Existência do certificado do player, validação de integridade previamente efectuada, login efectuado, device key reportada no login.
7. POST /api/title/validate/<hash> este método serve para enviar ao servidor a hash que tem até ao momento e o servidor cifra com a User Key de forma e envia o resultado para o player (necessita de cumprir as mesmas condições que o serviço anterior)

3.1.2 Webpage - Server

A interecção da página Web faz uso de 5 pedidos diferentes ao servidor:

1. POST /api/user/login em que o parametro é o nome do username (não existe password tendo em conta que a segunda parte do trabalho é autenticação) em que é associado ao session_id actual do utilizador o nome do utilizador em questão, sabendo assim qual o login efectuado durante a sessão toda
2. POST /api/user/logout em que remove qualquer associação no servidor ao session_id (necessita de login, ou seja, de ter um username associado ao session_id)
3. GET /api/title/all em que apresenta todos os titulos disponiveis (não necessita de qualquer login)
4. GET /api/title/user/all em que apresenta todos os titulos mas com o detalhe para o utilizador para saber se foram comprados ou (necessita de login)
5. POST /api/title/<pk> em que pk é o id do titulo que vai ser comprado (precisa de ter login efectuado).

3.2 Checking player integrity

Every time a player executes, an operation to check its integrity is performed. This falls into the authorization headers, however, we deal with it right off

the start.

We get the files that belong to the player implementation and create a hash of each of the files. We then concatenate the hashes of each file and once more hash the result with a random generated salt and we get the final result.

The server generates a SALT and stores it to the session temporarily (SALT is discarded when routine to validate is called or when requested to generate a new SALT), and sending it to the player, player will execute the sequence of operations described before to generate the hash. The result will be compared with the information stored in the server, and if they are equal, the player's integrity will be confirmed, otherwise, we assume someone has altered the code and the player is rejected.

São invocados dois serviços:

1. GET /api/valplayer/ em que devolve um SALT que permite fazer a validação de integridade do player
2. POST /api/valplayer/<hash> em que submete o resultado do challenge (se falhar, o salt é descartado e tem de ser gerado um novo)

3.3 Logging in

To be able to perform any action, the user must first login. This is mandatory in both web page and player.

For this, the server offers a login operation through the *Rest API*

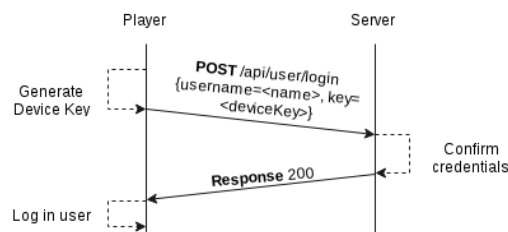


Figure 3.1: Login interaction (player - server)

As we can see from the diagram above:

1. The player starts by generating the Device Key for the device.
2. The player requests the login from the server, sending it the generated Device Key and the username that the user inserted.
3. The server then accesses the database to verify the username it received and associate the Device Key to the user (NOTE: If there's a

client certificate, it will compare with the hash of the certificate in the database, if it is the same, we have a valid player logged in, he still has to do a integrity check, but he proved authenticity)

4. The server sends the response that signals the operation was completed successfully.

The communication between player and server is done using sessions. Before the player starts communicating with the server, a session is created with the *requests* package for Python. From now on, all communications will be performed over this session.

3.4 Purchasing the file on the web page

This operation requires the user to be logged in already and is only available in the web page.

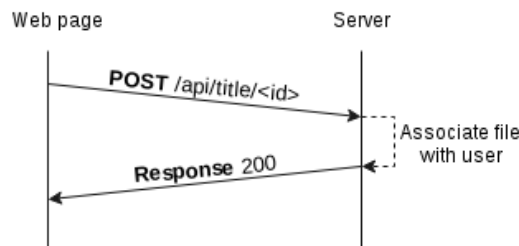


Figure 3.2: Purchase interaction (player - server)

As we can see, the operation isn't complex and the only thing done here is the association of the purchased file with the user. This association translates in a new entry in the UserFile table in the database.

3.5 Downloading the file from the player

In this interaction, we have two distinct situations:

1. Full file download
2. Cryptographic information download

3.5.1 Download requisites (validations)

Para poder ser efectuado o download de um ficheiro é preciso serem cumpridos os seguintes requisitos:

1. User tem de ter login efectuado

2. Device key tem de ter sido reportada no login
3. Player tem de ter validada a sua autenticidade (com o uso do certificado chapter 4TODO, vai permite associar a player key também)
4. Player tem de ter validada a sua integridade
5. User tem de ter comprado o ficheiro em questão
6. O ficheiro para o user em questão tem de cumprir todas as policies

A verificação dos requisitos é feita exactamente por esta ordem.

3.5.2 Full file download

The full file download situation means that this is the first time the player downs the file the user selected. And so, the server must send back the full encrypted file along with the cryptographic information that will allow the player to decrypt the file.

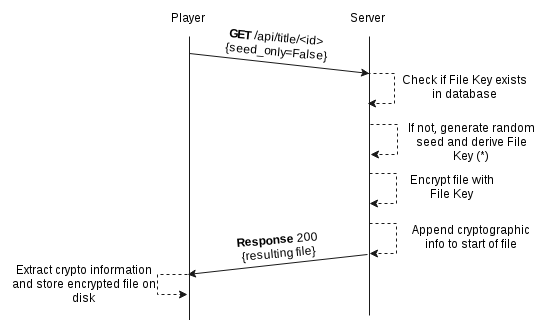


Figure 3.3: Full file download (player - server)

(*) In this operation, there can be yet another two situations:

1. The server has no File Key stored in the database, which forces the server to generate a random start seed to derive the File Key from. It then sends the seed with the encrypted file.
2. The server already has a File Key stored, and it must derive the starting cryptographic information to send to the player, along with the file.

3.5.3 Cryptographic information download

This situation only occurs if the player has already downloaded the file and has it stored on the disk.

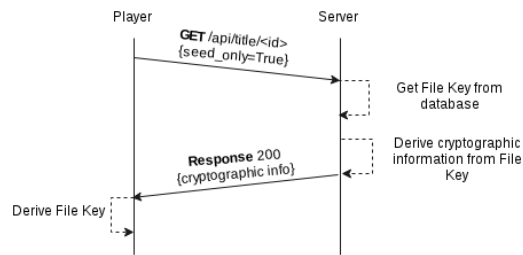


Figure 3.4: Seed only download (player - server)

3.6 Logging out from the player

Logging out is a simple operation that only implies ending the session in the server.

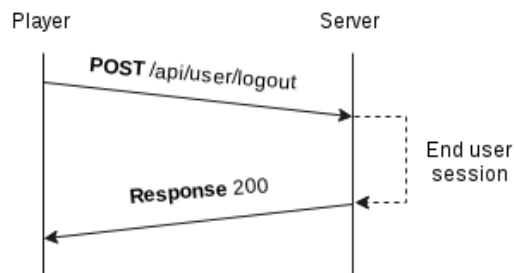


Figure 3.5: Logout interaction (Player/Webpage - Server)

Chapter 4

TLS Socket and Certificates

Para garantir que o canal de comunicação é seguro decidiu-se usar TLS Socket de forma a garantir que o canal é cifrado de forma segura, isto envolve alguns requisitos, tais como a geração de certificados de forma adequada.

Todos os certificados foram gerados usando XCA. Sendo que foi gerada uma Root Certificate Authority (CA) que é o Issuer de todos os certificados referidos neste relatório e que contém os seguintes campos:

1. CN - Security P3G1 Root
2. ST - Aveiro
3. L - Aveiro
4. O - Universidade de Aveiro
5. OU - Departamento de Electrónica, Telecomunicações e Informática

Também foi indicada o URI de uma CRL, no entanto não está a ser usada, podendo ser mais tarde implementada para revogar certificados.

4.1 CherryPy Server SSL Certificate

Para o CherryPy teve-se de gerar um certificado emitido pela Root CA de forma a garantir a cadeia de certificados a qualquer cliente que se ligue, player ou browser. Também se alterou o adaptador SSL do CherryPy de forma a suportar validação de certificado se existir no cliente (razões na secção seguinte).

4.2 Players SSL Certificates

Optou-se por gerar certificados para cada player de forma a efectuar a validação do player e podê-lo associa-lo a uma Player Key, sendo que foi preciso

alterar o adaptador original do CherryPy (BuiltIn) para suportar peer validation porque por defeito o adaptador SSL não verifica nada do cliente.

Sendo que se deixou a opção no CherryPy de o cliente poder ou não fornecer certificado (optional), caso não forneça tem a implicação de não poder fazer download de ficheiros, nem pedir ao servidor para fazer a validação da cadeia de chaves (aplicação da UserKey).

4.3 Browser validation

Para o browser indicou-se qual a Certificate Authority do nosso DRM de forma a que o Browser reconheça a CA como trusted e todas os certificados emitidos pela mesma.

Chapter 5

Policies

A verificação das políticas é feita quando o player tenta fazer download do ficheiro (apenas depois de garantir que está tudo bem com o player, isto pode ser verificado com detalhe na subsection 3.5.1)

1. Número limite de dispositivos por compra de um determinado ficheiro, sendo que existe uma tabela na base de dados que relaciona as compras com os dispositivos permitindo saber quais os dispositivos que reproduziram determinado ficheiro de uma compra.
2. Ficheiro válido para uma determinada região (sendo que a detecção da região é feita com base no Remote-Addr disponível nos headers do pacote HTTP).
3. Ficheiro válido para um determinado sistema operativo (sendo que a detecção do sistema operativo é feito com base no User-Agent disponível nos headers HTTP e fazendo o parse do mesmo à procura do sistema operativo em questão)-
4. Apenas possível reproduzir em determinada altura do dia (o calculo das horas é feita do lado do servidor).
5. Número máximo de reproduções de uma dada compra para um ficheiro (sendo que existe um contador no número de reproduções na compra).

Chapter 6

Installation

Está disponível um `install.sh` na pasta `/` do repositório que foi feito num Ubuntu 15.04 e também testado em 14.04 e 14.10 (a password para o PostgreSQL encontra-se no mesmo ficheiro de instalação em comentário).

Durante a instalação vai ser pedido várias interacções do utilizador, tal como seleccionar o certificado a instalar no sistema, ou a introdução da password para a Base de dados.

Foi gerada um clone de uma imagem VM, no entanto a imagem tem 3.7GB (não compactados), razão pela qual decidiu-se não fazer upload da mesma, de qualquer das formas podemos fornecer se houver problemas com a instalação.

Chapter 7

Conclusions

7.1 Encountered problems

7.2 Future work

Bibliography