universidade de aveiro deti departamento de eletrónica, telecomunicações e informática

IEDCS: Identity Enabled Distribution Control System

Segurança 2015/2016

Relatório realizado por:

- Bruno Silva (68535)
- Bernardo Ferreira (67413)

29 de Dezembro de 2015

Descrição

O IEDCS é um sistema de DRM (Digital Rights Managment) para ebooks. Neste projeto de segurança o nosso objectivo é criar um sistema completo, desde o servidor ao cliente, para fazer a gestão e distribuição dos livros.

O nosso projeto consiste em duas partes, o servidor e o player, que vão ser descritos com mais pormenor mais a baixo.

Dadas as metas pretendidas para esta fase do projeto, podemos afirmar que foram conseguidas com sucesso uma vez que temos um sistema funcional, onde um utilizador se pode registar no sistema, comprar livros, ver os livros que comprou e ler os livros. Todo este processo se encontra dentro dos requisitos pedidos onde, sempre que um utilizador pretende ler um livro necessita sempre do servidor, nunca são enviadas as chaves necessárias a decifra do ficheiro e toda a comunicação é realizada usando https e um certificado x509.

Como aspetos a melhorar, consideramos que deveria haver uma melhor gestão dos erros mas, uma vez que o trabalho era mais focado nos aspectos de segurança, decidimos gastar grande parte do tempo a tentar implementar as soluções que queríamos no âmbito da segurança e não da usabilidade do utilizador.

Em relação aos aspectos da segurança o confinamento foi uma área onde achamos que a solução dada e explicada mais a baixo no relatório não foi a melhor. Outros aspectos a melhorar seria a validação do player pela CA, e a validação da validade do cartão de cidadão ao ser usado para login. Tirando estes pequenos defeitos achamos que o trabalho está bem conseguido, cumprindo no geral os requisitos pedidos.

Deparamo-nos também com outro pequeno problema que não conseguimos encontrar solução, nesta ultima versão a leitura de um livro da erro na altura de leitura. Isto estava a funcionar e até já foi avaliado na primeira entrega. Além disso nos temos uma imagem de uma maquina virtual onde isto está a funcionar com a versão mais recente do código do servidor e do player o que nos diz que o problema está em alguma configuração que falta na maquina virtual e que não descobrimos. Ainda assim não achamos que seja um dos pontos principais desta segunda entrega.

29 de Dezembro de 2015

Componente Server

Descrição:

O IEDCS Server é o componente responsável por garantir a boa utilização e distribuição do conteúdo que fornece. Isto significa que ele tem de garantir a gestão de utilizadores, das compras desses utilizadores. Além dessas funções ele tem de assegurar a distribuição dos conteúdos adquiridos pelos utilizadores de forma segura e controlada.

O servidor, no nosso sistema, está implementado como um servidor web usando a linguagem de programação Python e a framework de Django. A comunicação com o player e o web site é feita através de uma API com endpoints HTTP REST.

Descrição da API:

A API no servidor está implementada com a “Django REST framework”, implementa endpoints para permitir a comunicação entre o servidor e o cliente seja este o player ou uma pagina web. Os seguintes endpoints foram implementados a medida das necessidades do sistema e fornecem a funcionalidade pedida assim como validações simples a todos os parâmetros recebidos:

/books/get_books/

Endpoint que suporta o método GET para receber a lista de todos os livros existentes no sistema. Cliente recebe a lista completa em json contendo informação como o: título, autor, url para a capa, língua em qual está o livro escrito, e um identificador único do livro.

/books/get_book/?book_id=<0-9>

Endpoint que suporta o método GET para receber informação especifica sobre o livro indicado como argumento “book_id”. A informação recebida é a mesma do endpoint anterior mas apenas para o livro pedido.

/users/create_user/

Endpoint que suporta o método POST para criar um novo utilizador no sistema.
Para registar um utilizador novo no sistema é necessário fornecer os parâmetros: primeiro nome, ultimo nome, e-mail e password. No momento de criação de um novo utilizador é gerado um valor aleatório de 16 bytes que é a user key. Este valor é guardado na base de dados numa tabela diferente da de utilizadores, e é usado para cifrar as compras daquele utilizador.

Após o registo e ainda na mesma função o utilizador é autenticado e é realizado o login do mesmo e é definido um cookie com o primeiro nome e e-mail do utilizador para ser usado pela página web.

/users/user_login/

Endpoint que suporta o método POST para fazer login a utilizadores do sistema. O login requer os parâmetros: e-mail e password. De seguida utiliza os métodos do Django para realizar o login do utilizador no sistema.

/users/user_logout/

Endpoint que suporta o método POST para fazer logout a utilizadores do sistema. O logout nao requer quais quer parametros, usando apenas o request. Para efetuar o logout são usados os métodos do Django.

/users/buy_book/

Endpoint que suporta o método POST para um utilizador poder comprar um livro. Para a compra de um livro é necessário apenas enviar como argumento o identificador do livro os outros parametros necessários são conseguidos a partir do request.

Para efetuar uma compra o Django cria um random e de seguida cria uma entrada na tabela de compras com o utilizador atual, o identificador do livro e o random associado que vai ser usado para cifrar leituras daquela compra.

/users/get_purchases/

Endpoint que suporta o método POST para receber todas as compras de um utilizador. Recebe como argumento o e-mail do utilizador para o qual queremos as compras e retorna uma lista em json de todas as compras desse utilizador.

/users/register_device/

Endpoint que suporta o método POST para registar um device com um user. Recebe como argumento o e-mail do utiizador, a device key e um nome do device. Caso o nome do device seja vazio este é preenchido com “<first_name_user>’s Computer”. Numa situação em que esse dispositivo já esteja associado aquele user é dado uma resposta a informar dessa situação.

/requests/read_book/

Endpoint que suporta o método POST para inicializar o processo de leitura de um livro. Como argumento recebe o id do livro que o utilizador quer ler e valida se ele já comprou o livro. Como resposta é enviado um header com os parametros que o cliente tem

de enviar para garantir que a leitura do livro é possivel. Esses parametros são: book_id, device_key, player_version, location, so e time.

/requests/validate/

Endpoint que suporta o método POST para validar os dados do utilizador e cifrar o livro para ele ler. Como argumentos este método recebe todos os parâmetros que pediu ao utilizador na chamada ao endpoint anterior (/requests/read_book/). Após as validações dos parâmetros o método acede a base de dados ás tabelas das restrições e verifica se os dados do utilizador estão dentro dos permitidos para a leitura do livro. Depois de garantir as restrições é efetuado o inicio da cifra do livro. Para isso é gerada uma file key que vai depender do random da compra, da player_key, da user_key e da device_key da seguinte forma: file_key = E(device_key, E(user_key, E(player_key, random))), considerando a função de cifra E(chave, texto). Todas estas cifras são realizadas com AES no modo CBC.

Após estar gerada a file key, essa chave é usada para cifrar, do mesmo modo anterior, o livro por blocos e gerado um novo ficheiro para o livro onde este esta cifrado.

/requests/get_file/

Endpoint que suporta o método POST para receber o ficheiro cifrado previamente. Este método recebe o book_id do livro pretendido e envia o conteúdo do livro com a codificação base64. Após o utilizador chamar este método o livro cifrado é apagado e para uma nova leitura todo o processo tem de ser repetido.

/requests/decrypt/

Endpoint que suporta o método POST para permitir auxilio ao utilizador de fazer a decifra do ficheiro. Este método é necessário para o utilizador necessitar sempre da interação do servidor para conseguir ler o ficheiro, uma vez que a user key nunca sai do servidor e ela é usada no processo.

/users/addCC/

Endpoint que suporta o método POST para permitir a um utilizador adicionar a chave publica do seu cartão de cidadão aos seus dados no servidor para permitir assim que o login na aplicação seja efetuada a partir do mesmo. O endpoint recebe como argumento a chave publica do utilizador, valida a sua autenticidade, e valida que este esta autenticado e que ainda não adicionou nenhuma chave publica, para poder adicionar este método de autenticação.

/users/loginCC/

Endpoint que suporta o método POST para iniciar o processo de login através do cartão de cidadão. Este método recebe o e-mail do utilizador e gera um random de 1024

bits que vão ser enviados para o player no header da resposta como uma challenge ao utilizador. Vai ainda ser adicionada uma entrada na base de dados onde é guardado o random que foi enviado.

/users/validateLoginCC/

Endpoint que suporta o método POST para finalizar o processo de login com o cartão de cidadão. Neste método o processo de login é validado, para isso o método recebe como argumentos o id da transição, usado para aceder a base de dados ao random que foi utilizado, e a assinatura do random que foi efetuada pelo cartão. Usando a biblioteca pycrypto e sabendo que o cartão de cidadão funciona com RSA e SHA-1 o servidor cria uma chave publica RSA com o que já tinha armazenado anteriormente, de seguida faz o SHA-1 do random que foi enviado para o utilizador assinar e usando métodos da biblioteca valida a assinatura do utilizador, autenticando-o caso a assinatura seja válida. Para autenticar o utilizador sem palavra passe foi necessário criar um novo backend de autenticação que tem de ser usado com caução por poder autenticar utilizadores sem palavra passe.

Descrição da WebStore:

Neste projecto a webstore não era um dos objetivos principais do trabalho, dado está condição foi implementado um modelo simples que permite apenas registar utilizadores, fazer login, ver os livros disponíveis e efetuar compras.

Validação do servidor

Por forma a validar o servidor, foi criada uma CA, usando o software XCA usado nas aulas, e um certificado para o servidor assinado por essa CA. O apache foi também configurado com o certificado a usar assim como o caminho por onde ele pode validar a cadeia de confiança, neste caso o certificado da nossa CA.

Confinamento

Por forma a confinar o nosso sistema o nosso objectivo era, uma vez que usamos django que é escrito em python, criar um virtualenv e assim isolar a instalação de python. Esta parte do processo foi conseguida com sucesso e de facto o apache está a utilizar uma instalação de python do virtualenv criado com apenas as bibliotecas pretendidas. A segunda parte do processo passava por fazer chroot a pasta onde o apache estava a executar e está parte não foi conseguida. Tentamos com o mecanismo do apache que já vem incorporado e dava um erro que não encontrava o módulo time do python, erro que mesmo depois de todos os esforços não foi conseguido ser resolvido. Tentamos também outra alternativa em criar uma pasta nova que seria o nosso diretório e tentamos copiar tudo para lá e fazer o chroot, mas também não funcionou. Agora mais para o final foi tentado usar um container

de docker para correr o apache, mas ao iniciar a configuração do docker foi visto que o código do Dockerfile para criar o container era em tudo semelhante ao do Vagrantfile que está a ser usado para criar a maquina virtual. Com isto, e pela falta de tempo para configurar o Docker uma vez que ia ser semelhante a configurar o vagrant, deixamos ficar o confinamento por aqui, usando o confinamento como sendo a nossa própria VM em relação ao sistema instalado de raiz na maquina.

Sistema de ficheiros seguro

Como forma de guardar os livros de forma segura no servidor foi usado o encfs usado nas aulas práticas. Este sistema permite guardar os livro de forma cifrada numa maquina distinta, no nosso caso uma pasta diferente no mesmo sistema, e montar essa pasta no servidor web a usar o seu conteúdo de forma decifrada, isto tudo de forma completamente transparente para quem usa a pasta decifrada. Toda esta configuração é realizada pelo nosso script que configura a maquina.

Componente Player

Descrição:

O IEDCS Palyer é o componente responsável pela reprodução de um livro previamente comprado e distribuído pelo servidor.

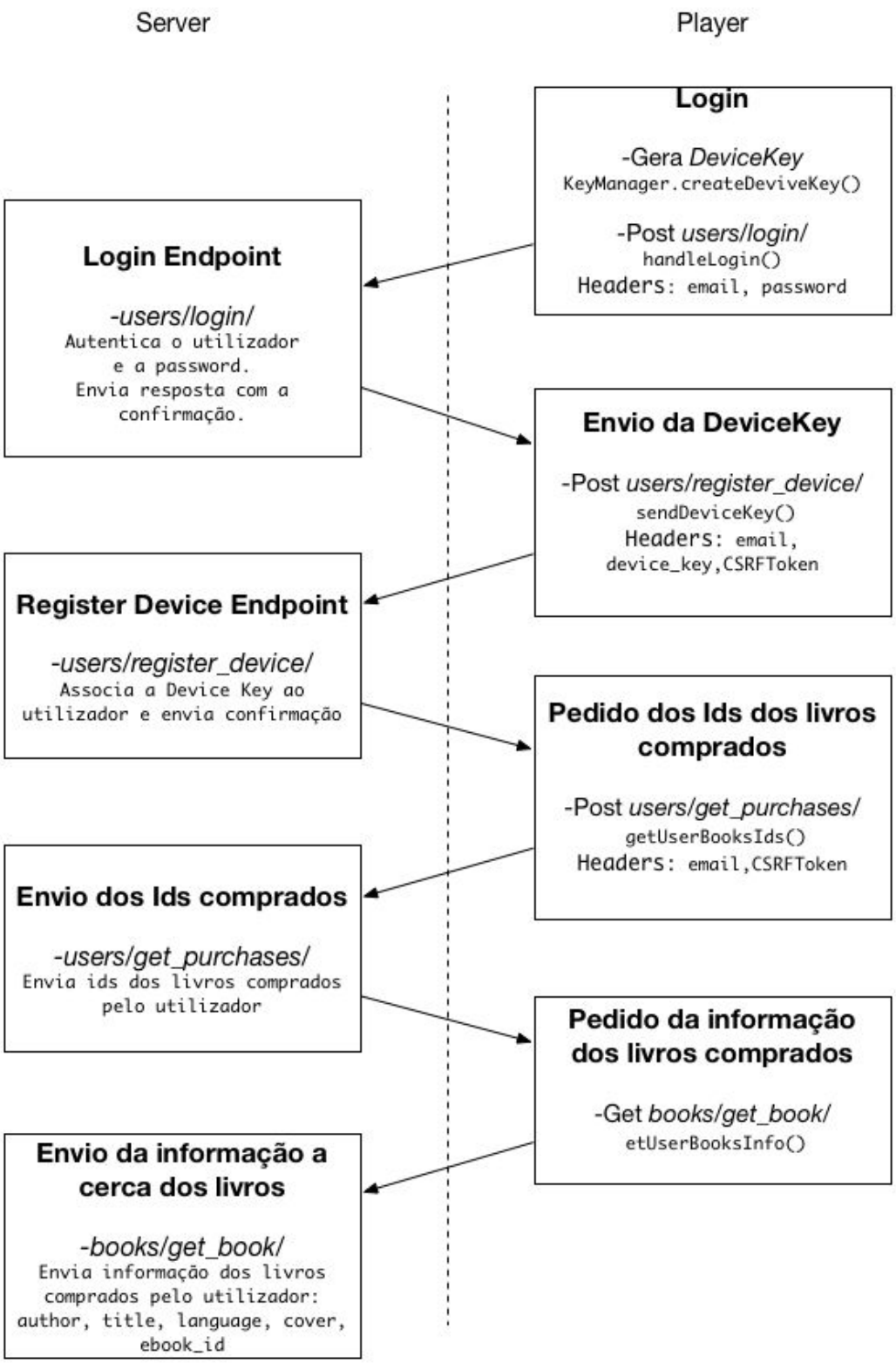
O Player está implementado como uma aplicação Java, utilizando a biblioteca gráfica JavaFX. A comunicação entre a aplicação e o servidor é feita utilizando a biblioteca HttpClient da Apache.

Fluxo do programa:

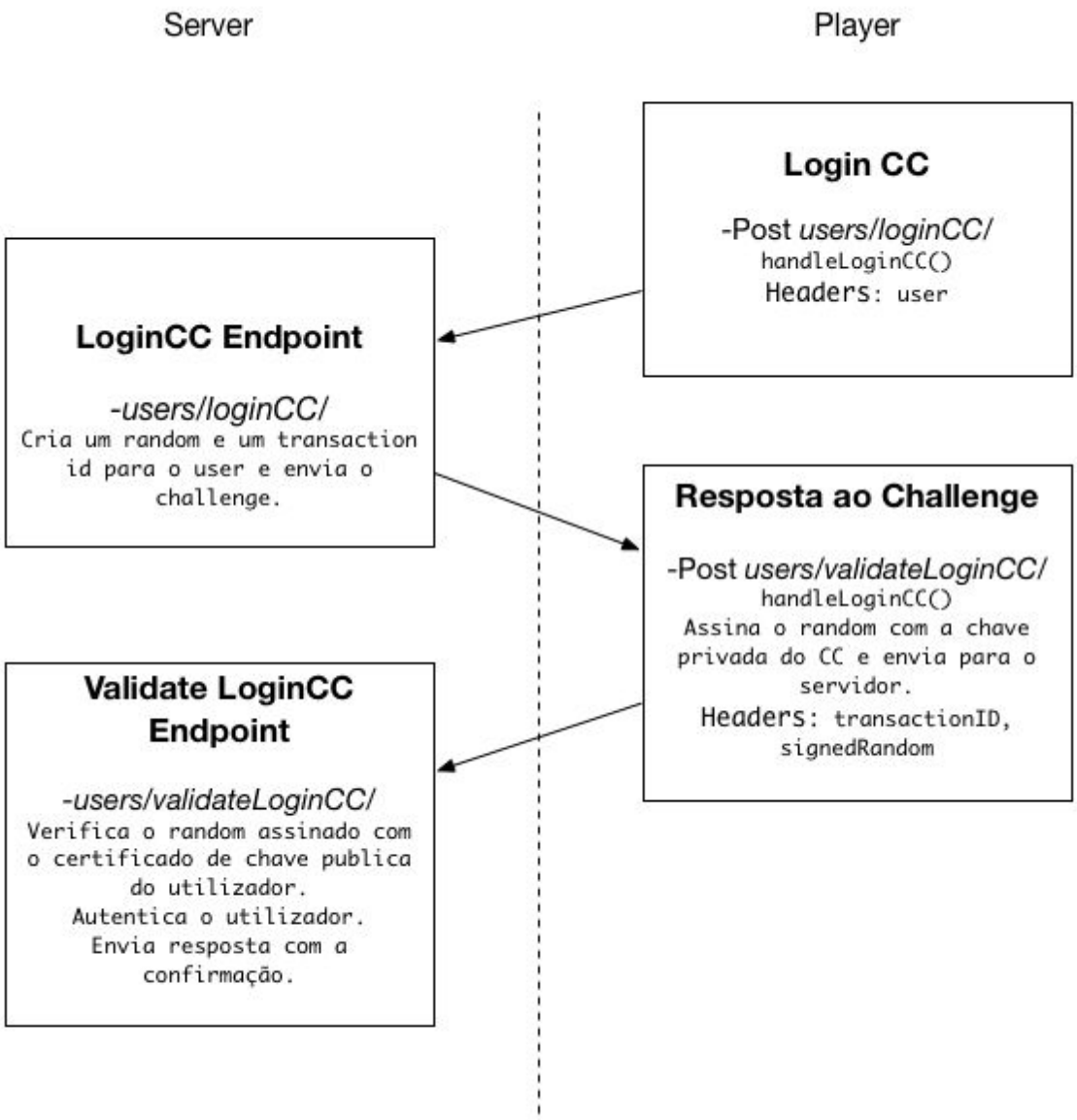
Depois de iniciar a aplicação é criada uma *Device Key* como sendo o número de série do computador utilizado, de modo a identificar unicamente o dispositivo que está a ser utilizado, e, é de seguida apresentada uma janela de login, onde devem ser introduzidos os dados de autenticação previamente usados na criação de uma conta na Web Store ou, se usando o cartão de cidadão, introduz-se o email e o cartão no leitor. Estes dados são enviados para o servidor e no caso de os dados estarem corretos, a *Device Key* é também enviada de modo a identificar o dispositivo em que o utilizador está a usar a aplicação.

De seguida, as informações sobre os livros comprados pelo utilizador são pedidas ao servidor, e é apresentada uma janela com a listagem destes livros. O utilizador pode escolher um livro para ver mais informações e selecioná-lo para ler. O processo de pedido do livro ao servidor bem como toda a interação Server <-> Player vai ser descrita no seguinte diagrama:

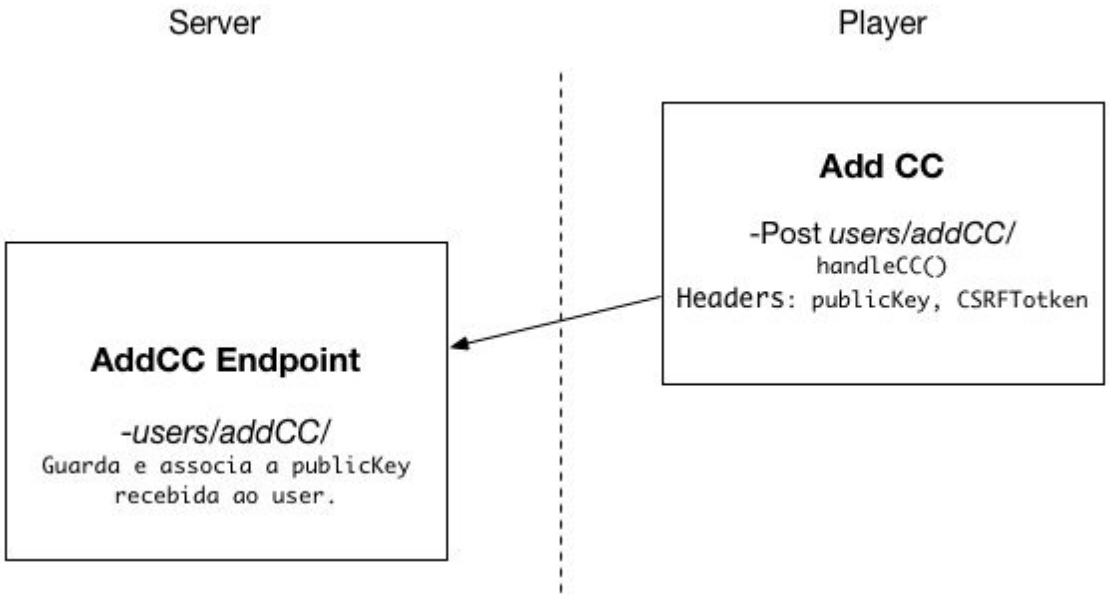
- Login:


Server
Player
Login
-Gera DeviceKey
KeyManager.createDeviveKey()
-Post users/login/
handleLogin()
Headers: email, password
Login Endpoint
-users/login/
Autentica o utilizador e a password. Envia resposta com a confirmação.
Envio da DeviceKey
-Post users/register_device/
sendDeviceKey()
Headers: email, device_key,CSRFToken
Register Device Endpoint
-users/register_device/
Associa a Device Key ao utilizador e envia confirmação
Pedido dos Ids dos livros comprados
-Post users/get_purchases/
getUserBooksIds()
Headers: email,CSRFToken
Envio dos Ids comprados
-users/get_purchases/
Envia ids dos livros comprados pelo utilizador
Pedido da informação dos livros comprados
-Get books/get_book/
etUserBooksInfo()
Envio da informação a cerca dos livros
-books/get_book/
Envia informação dos livros comprados pelo utilizador: author, title, language, cover, ebook_id

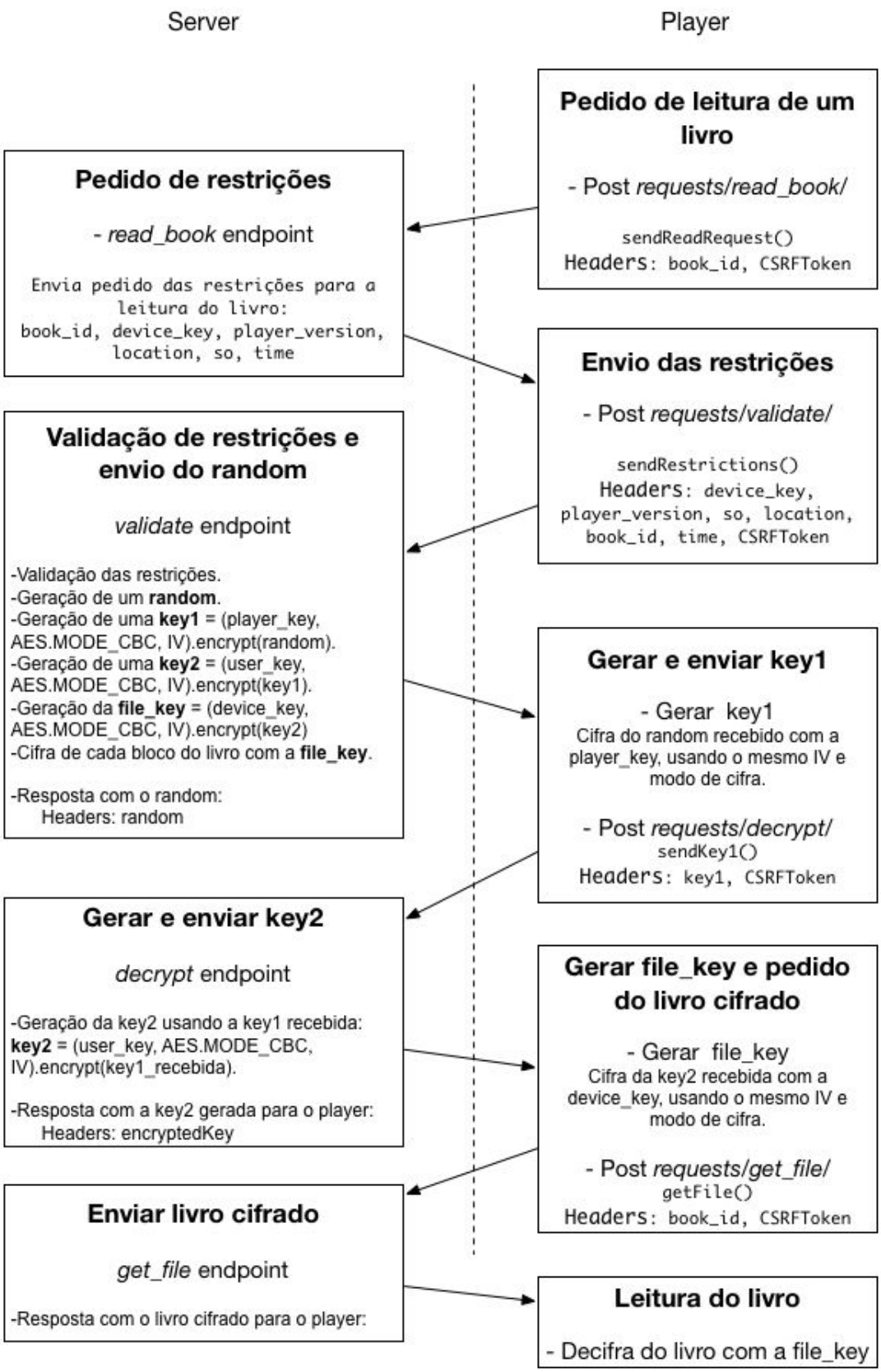
- Login com o CC:



- Adicionar CC:



- Leitura de um livro:



Instalação e execução:

Server

Para executar o servidor nesta segunda entrega esá tudo mais facilitado bastando para isso ter instalado o vagrant e executar “vagrant up” no diretório raiz do projeto. Isto vai criar uma maquina virtual nova e executar o script boostrap.sh que se encontra na mesma pasta. Este script configura a maquina toda.

Após o servidor ser executado, terá de se aceder a https://127.0.0.1:8080/, e carregar em “Sign up FREE” para criar uma conta. De seguida basta comprar os livros desejados.

Player

Para executar o programa, deve-se ir a pasta “m1/player/IEDCS/build/deploy/” e executar o jar que lá se encontra com o comando “java -jar IEDCS.jar”.

Após o player estar em execução basta fazer login com as mesmas credenciais que foram criada na webstore, após ter entrado com sucesso vai ser levado para uma página onde aparecem os livros que já comprou. basta selecionar o pretendido e carregar em “Read Book”. Vai ser levado para um leitor com o livro.

Para se adicionar o cartão de cidadão, depois de o login, carrega-se em File-> AddCC.

Para o login usando o CC, insere-se o email, o cartão no leitor e carrega-se no botão Login CC

Links auxiliares:

Server

Cifrar em python com AES:
http://eli.thegreenplace.net/2010/06/25/aes-encryption-of-files-in-python-with-pycrypto
http://stackoverflow.com/questions/12524994/encrypt-decrypt-using-pycrypto-aes-256

Criar utilizadores e gerir login/logout:
https://docs.djangoproject.com/en/1.8/topics/auth/default/

Testar conexões em https com o servidor de desenvolvimento e com um certificado auto assinado:
http://stackoverflow.com/questions/8023126/how-can-i-test-https-connections-with-django-as-easily-as-i-can-non-https-connec

Definições para o https no Django:
https://docs.djangoproject.com/en/1.8/ref/settings/#std:setting-SECURE_HSTS_INCLUDE_SUBDOMAINS

Validação de assinaturas com RSA:
https://www.dlitz.net/software/pycrypto/api/current/Crypto.Signature.PKCS1_v1_5-module.html

Configurar apache com virtualenv:
http://thecodeship.com/deployment/deploy-django-apache-virtualenv-and-mod_wsgi/

Chroot (foram consultados mais sites/forums não referidos aqui por esquecimento de serem apontados na altura):
https://groups.google.com/forum/#!topic/modwsgi/sOrtcm7JV50
https://code.google.com/p/modwsgi/issues/detail?id=106
https://wiki.ubuntu.com/ModChroot
http://core.segfault.pl/~hobbit/mod_chroot/install.html

Player:

Ignorar verificação dos certificados SSL para chamar endpoints em https:
http://literatejava.com/networks/ignore-ssl-certificate-errors-apache-httpclient-4-4/

Tutorial de inicição ao JavaFX:
http://code.makery.ch/library/javafx-8-tutorial/

Encontrar o IP público:
http://stackoverflow.com/questions/2939218/getting-the-external-ip-address-in-java

Tutorial Apache HttpClient:
http://hc.apache.org/httpclient-3.x//userguide.html

Encontrar localização através do IP:
http://www.mkyong.com/java/java-find-location-using-ip-address/

Encontrar o MAC Adress em java:
http://www.mkyong.com/java/how-to-get-mac-address-in-java/