



UNIVERSIDADE DE AVEIRO

DEPARTAMENTO DE ELECTRÓNICA, TELECOMUNICAÇÕES E
INFORMÁTICA

47232- SEGURANÇA

IEDCS

Identity Enabled Distribution Control System

8240 - MESTRADO INTEGRADO EM ENGENHARIA DE
COMPUTADORES E TELEMÁTICA

António Rafael da
Costa Ferreira
NMec: 67405 | P4G1

Rodrigo Lopes
da Cunha
NMec: 67800 | P4G1

Docente: João Paulo Silva Barraca

Novembro de 2015
2015-2016

Conteúdos

1	Introdução	3
2	Processos apresentados na 1ª apresentação	4
2.1	Processo de cifra do conteúdo do livro	4
2.2	Processo de decifra do conteúdo do livro	5
3	Alterações efetuadas em relação à 1ª apresentação	6
3.1	Processo de cifra do conteúdo do livro	6
3.2	Cifra simétrica e modo	8
3.3	Visualização de um livro	8
3.4	Processo de decifra do conteúdo do livro	9
4	Tecnologias usadas	11
5	Casos de utilização	12
5.1	Registo de um utilizador	12
5.2	Registo de um device	13
5.3	Aquisição de um livro	13
5.4	Restrições na visualização de um livro	13
6	HTTPS	14
7	Problemas identificados na primeira entrega	14
7.1	Soluções	15
8	Segunda entrega, síntese de melhorias	16
9	Pequenas melhorias implementadas	16
9.1	New Secret Key	16
9.2	Django Rest Framework	16
9.3	Melhoria do Debug do Player	16
9.4	Diffie Hellman na visualização de um livro	16
9.5	Copy/ Paste da janela do Player	17
10	Novas funcionalidades	18
10.1	Associar Cartão de Cidadão a uma conta	18
10.2	Desassociar Cartão de Cidadão de uma conta	19
10.3	Login com o Cartão de Cidadão	20
10.4	Logout	21
11	Confinamento	21

11.1	Chroot with Apache	21
11.2	Docker	22
11.3	Conteúdo cifrado no servidor	24
12	Autenticação do Player junto do Servidor	24
12.1	Jar sign	24
12.2	Certificados assinados pela CA do IEDCS	25
12.3	Autenticação usando o Cartão de Cidadão	26
12.4	Identificação do cidadão junto do servidor	26
12.5	Validação do cartão de cidadão usado pelo utilizador	27
13	Instalação	28
14	Conclusão	30

1 Introdução

O trabalho proposto para o projeto da unidade curricular de Segurança é um IEDCS: Identity Enabled Distribution Control System. Para o efeito foi necessário implementar uma Ebook Webstore, um WebService e um Player de reprodução dos Ebooks em formato de texto.

O objetivo deste sistema é garantir a máxima e possível segurança do serviço, utilizando os conhecimentos adquiridos na unidade curricular de Segurança. Para isso são necessários vários processos como por exemplo, a utilização de certificados HTTPS, a cifragem de todo o material existente, derivação de chaves e registo de utilizadores.

O relatório reflete todos os passos e decisões tomadas na criação do sistema, assim como uma análise ao que foi mostrado na primeira apresentação e decisões que se tomaram depois desta, tecnologias utilizadas, descrição dos vários processos existentes e conclusão.

2 Processos apresentados na 1ª apresentação

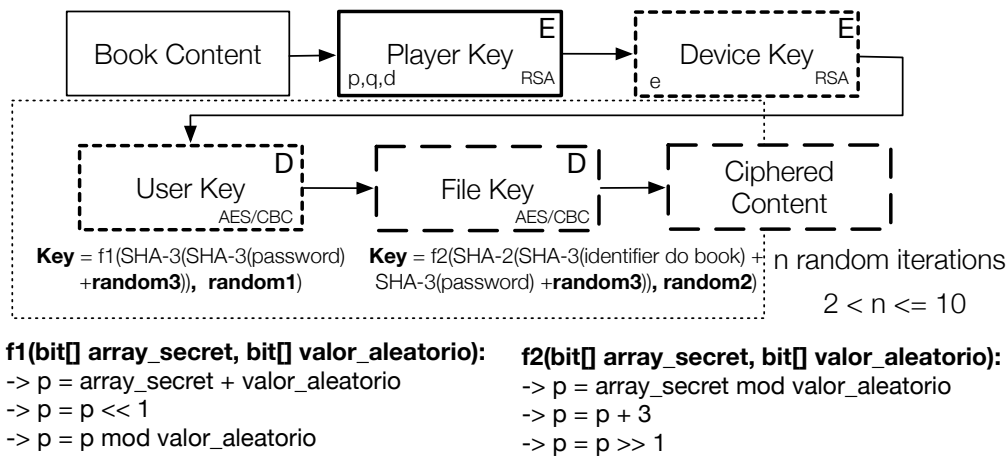
2.1 Processo de cifra do conteúdo do livro

Na primeira apresentação foram concebidos dois processos que foram abandonados após a sua apresentação. O primeiro processo apresentado foi o “Book content cipher process”. Este pretendia fazer uso de cifras simétricas e assimétricas de forma a ser possível obter uma File Key que seria usada para cifrar o conteúdo do livro. Idealmente o conteúdo do livro seria cifrado com a chave privada do Player, depois cifrado com a chave pública do Device e finalmente seria feito sobre esses resultados uma cifra com o User Key e a File Key. O processo de cifra com o User Key e a File Key seriam repetidos n vezes, sendo o n obtido através do random2. Foi pensado também usar duas funções para derivar as keys, uma f_1 que seria usada para derivar a User Key e uma f_2 que seria usada para derivar a File Key.

A ideia de usar uma função de derivação da File Key e da User key é boa, e usando um número de iterações finita também. Isto permite que quem esteja a ouvir na rede tenha mais dificuldades em obter uma chave usando aquilo que conseguiu ouvir, pois teria de conhecer o processo usado pelos intervenientes para conseguir obter as chaves. Diga-se que também não seria difícil perceber o processo usado pelos dois intervenientes pois basta ao atacante usar o serviço e obter um Player e usando ferramentas próprias analisar o código.

A causa de ter-se abandonado este processo para cifrar o conteúdo do livro foi:

- É computacionalmente exigente cifrar com chaves assimétricas conteúdos grandes.
- Normalmente para evitar este esforço computacional faz-se uma assinatura sobre o resultado de uma função de síntese desse mesmo conteúdo do livro.
- As chaves simétricas são muito mais eficientes a nível computacional.
- As funções para derivar as chaves foram abandonadas em detrimento da PBKDF2 que efetua já as iterações idealizadas.
- Não é boa ideia usar as passwords dos utilizadores para cifrar o conteúdo do mesmo pois caso o utilizador mude a password ter-se-ia de derivar as chaves outra vez.



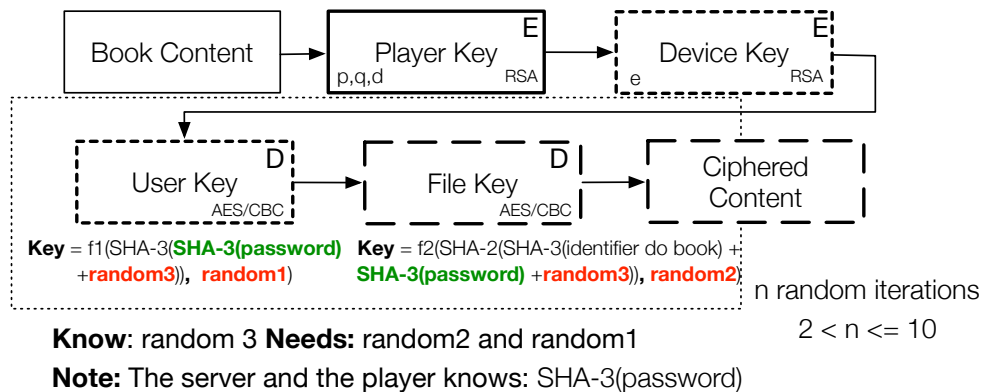
Note: The server and the player knows: $\text{SHA-3}(\text{password})$

Figura 1:
 Processo de cifra usado na 1ª apresentação

2.2 Processo de decifra do conteúdo do livro

O processo de decifra foi desenhado para reverter o processo anunciado em cima. Sendo por isso nada mais do que fazer as operações inversas das realizadas no processo de cifra. As razões para que se tenha abandonado este processo de decifra são as mesmas enunciadas no processo de cifra.

O processo de “challenge” consistia no pedido do *random1* e do *random2* ao servidor, mas, isso não é suficiente para que o processo seja seguro, pois basta ao atacante conhecer o processo de cifra e decifra e interceptar os pedidos ao servidor do *random1* e do *random2*. Para o Player também seria fácil decifrar sem necessitar da cooperação do servidor, pois, bastava que cooperasse uma primeira vez.



f1(bit[] array_secret, bit[] valor_aleatorio):

-> $p = \text{array_secret} + \text{valor_aleatorio}$

-> $p = p \ll 1$

-> $p = p \bmod \text{valor_aleatorio}$

f2(bit[] array_secret, bit[] valor_aleatorio):

-> $p = \text{array_secret} \bmod \text{valor_aleatorio}$

-> $p = p + 3$

-> $p = p \gg 1$

Figura 2:

Processo de decifra usado na 1ª apresentação

3 Alterações efetuadas em relação à 1ª apresentação

No seguimento da primeira apresentação procedeu-se a muitas alterações no modo como o processo de cifra e decifra do conteúdo de um livro seria efetuado.

3.1 Processo de cifra do conteúdo do livro

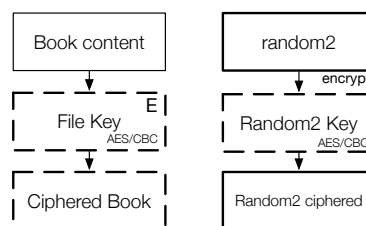


Figura 3:

Processo de cifra de um livro

Neste novo processo, o servidor cifra um valor aleatório, designado por Random2, com uma chave Random2 Key, calculada através da seguinte

forma:

$$\text{Random2 Key} = \text{PBKDF2}(\text{username} + \text{"jnpc"} + \text{book.identifier}, \text{book.identifier}, 1000)$$

Esta key é partilhada pelo servidor e pelo cliente, pois ambos conhecem a forma de como ela é calculada, não sendo preciso enviar a mesma. Apenas o conteúdo de Random2 cifrado é enviado para o cliente. O valor Random2 foi criado, para que através dele tanto o servidor como o cliente possam obter a File Key que será utilizada para cifrar o conteúdo de um livro que não poderá ser guardada do lado do cliente e poderá ser reutilizada para um mesmo utilizador noutra Player ou noutra Device.

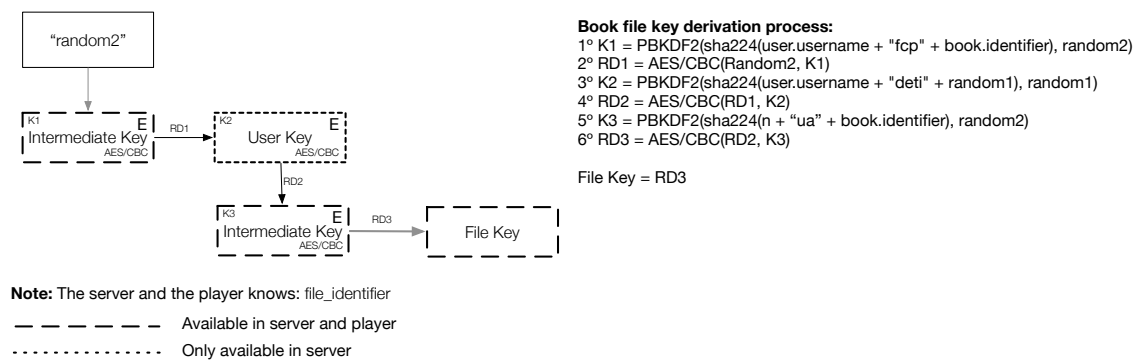


Figura 4:
Processo de derivação da File Key

Para derivar a File Key, como se pode ver na figura 4, o Random2 é cifrado com uma chave K1, que é calculada através do SHA-224 do username, de um texto ("fcp") e do identificador do livro. Este SHA é a password e o Random2 o texto, aplicando-se assim a função PBKDF2 (recomendado na apresentação), com o valor por defeito de 1000 iterações, obtendo-se assim a chave pretendida, K1. O cálculo desta chave K1 é possível de ser efetuado tanto do lado do servidor como do cliente.

Posto isto, o resultado obtido da cifra de Random2 com a chave K1, RD1, vai ser cifrado com uma nova chave K2, sendo esta apenas calculada pelo servidor. Esta é calculada praticamente como a K1, sendo diferente por no SHA-224 utilizar um outro texto ("deti") e um *random1*, em vez de *random2*, apenas disponível no servidor, em vez de usar o identificador do livro. Aplica-se igualmente a função PBKDF2, com o SHA como password e o *random1* como texto, de forma a obter K2 para se poder cifrar o valor de RD1.

Numa última fase deste processo de derivação da File Key, é calculada ainda uma terceira chave, K3, com o mesmo processo das outras chaves K1 e K2, sendo nesta última utilizado no cálculo do SHA um valor n , que corresponde a:

$$n = (\text{soma de todos os dígitos de random2}) \% 10$$

Juntamente com o valor n , utiliza-se o identificador do livro e um texto “ua”. Através deste SHA e do random2 que atuam respetivamente como password e salt, aplica-se uma função PBKDF2 para obter K3.

Por último através da cifra de RD2 (valor obtido da cifra de *random1* com a chave K2) com a chave K3 obtida, obtém-se a File Key, que será utilizada para cifrar o conteúdo do livro e desta forma este conteúdo cifrado será enviado para o cliente.

3.2 Cifra simétrica e modo

Todo o processo de cifra e decifra de um livro e de um random2, foi efetuado através da cifra simétrica por blocos AES com o modo de cifra CBC.

Foi escolhida a cifra simétrica AES, pois cifra por blocos e garante maior rapidez tanto no processo de cifra como de decifra.

O modo de cifra CBC foi selecionado devido à existência de confusão, introduzida por um IV no início do processo de cifra e decifra. Cada bloco cifrado funciona como realimentação para o bloco seguinte, exceto no inicial, onde é utilizado o IV. Isto permite ter acesso aleatório uniforme apenas na decifra, que é o pretendido para o método de paginação, sendo então possível decifrar o bloco que se deseja, sem ter de processar os anteriores.

3.3 Visualização de um livro

Na primeira vez que um utilizador solicita a visualização de um livro é gerado o primeiro Random2 e Random1 que vai ser único para determinado utilizador e livro que tenha adquirido o livro. Estes valores aleatório são guardados na base de dados cifrados com uma chave que depende do utilizador e do livro.

Todos os Initialization Vectors são também guardados na base de dados com o mesmo processo.

Os Random2, Random1 e Initialization Vectors passam todos por um processo de cifra antes de serem guardados na base de dados. Este processo consiste em:

- Gerar um *identifier*, através de um SHA-224 composto por um alias (nome pelo qual se quer identificar o conteúdo, por exemplo “random2”), username e identificador do livro.

- Gerar um valor que será a password na função PBKDF2, através de um SHA-224, composto pelo username, email e último nome.
- Depois de gerados os dois valores acima descritos, recorre-se a uma função PBKDF2, onde o salt é o *identifier*. Esta função terá um número de 1000 iterações por defeito.
- Posto isto, o conteúdo que se deseja guardar na base de dados é cifrado com a chave gerada acima, ficando assim salvaguardado com alguma segurança.

3.4 Processo de decifra do conteúdo do livro

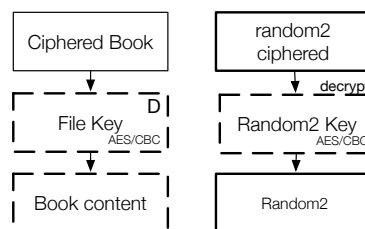


Figura 5:

Processo de decifra de um livro

Tal como no processo de cifra de um livro, a Random2 Key é calculada, mas desta vez no lado do cliente para este conseguir decifrar o random2 que recebeu cifrado, obtendo assim o valor original que vai ser utilizado para o cálculo da File Key usando também um processo de challenge.

O processo de derivação da File Key no lado do servidor e do cliente é feito da mesma maneira, apenas mudando, que quando o Player está a gerar a File Key, precisa de questionar o servidor (figura 6), pois ele é o único capaz de calcular o valor de RD2, obtido através da cifra de RD1 (cifra de Random2 com K1) com uma chave K2 calculada com um random1 que só o servidor conhece (figura 5).

Este "challenge", que é um processo de pergunta e resposta, em que a pergunta tem de vir bem formulada, consiste na pergunta que o Cliente faz ao servidor, como já mencionado e na resposta, com o valor que calculou, para que o cliente prossiga o processo de derivação da File Key.

A pergunta colocada ao servidor tem de ter um tamanho esperado e tem de ser colocada em Base64. Como a pergunta é em Base64 espera-se que a pergunta tenha um valor de 108 caracteres.

O resultado da cifra de RD1 com AES/CBC tem 64 bytes, resultante de o random 2 ter 56 bytes (mais o padding ficam 4 blocos de 16 bytes cada).

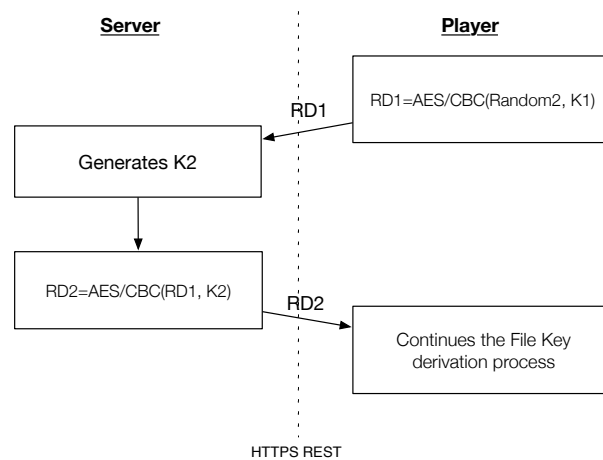


Figura 6:
Challenge entre Player e Servidor na derivação da File Key

Com o IV soma-se 16 bytes. Tem-se então, 80 bytes a enviar para o servidor em Base64, sendo isto 640 bits. Como cada caracter é usado para representar 6 bits, mas como 640 não é múltiplo de 6 é preciso adicionar padding, neste caso, terá de ser adicionado 2 bits para ser múltiplo de 6 bits. Sendo assim 642 resulta em 107 caracteres, como apenas adicionou-se 2 bits apenas será necessário adicionar um '=' no final, resultando em 108 caracteres.

A resposta dada pelo servidor não é verificada se é a correta ou não pois isso poderia resultar em pedidos mal intencionados só para saber qual seria a resposta correta, e por consequência qual seria o valor de entrada correto para aquele livro e utilizador, resultando numa fraqueza no processo de cifra/decifra.

Após este processo de derivação da File Key o Player decifra o conteúdo cifrado do livro de forma a obter o conteúdo original do mesmo, podendo agora apresenta-lo ao utilizador.

Paginação

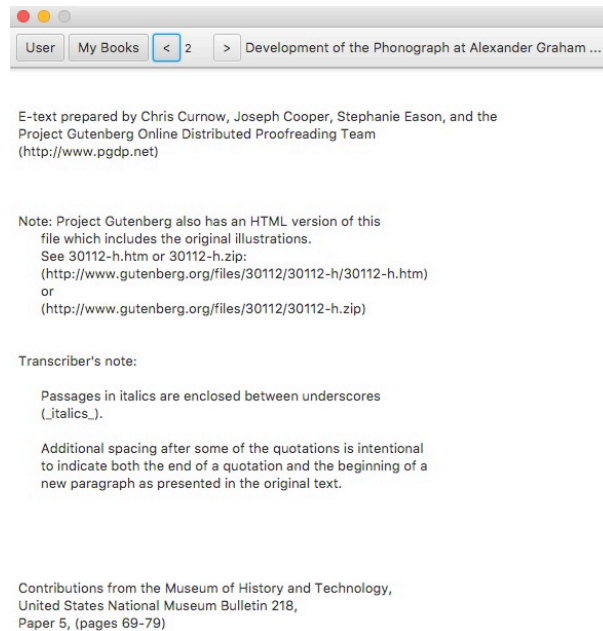


Figura 7:
Paginação ajustada à janela

A paginação foi feita de acordo com o tamanho da janela do Player de modo ao número de linhas ser igual a 31. Caso o último bloco a ser decifrado possua mais que uma linha, é apenas escrita a primeira linha desse bloco, sendo o resto do texto apresentado no início página seguinte.

Convém contudo realçar que a decifra do conteúdo é feita página a página, usando o acesso aleatório do modo CBC, e a cada página apresentada ao utilizador são calculadas todas as chaves necessárias e é efetuado o challenge ao servidor.

4 Tecnologias usadas

As tecnologias usadas no desenvolvimento do IEDCS Server, foram: Django + Python e AngularJS + JS. O Django dá a possibilidade de usar proteção contra CSRF usando um token e um middleware que obriga que o cliente envie este token sempre que faz um pedido para a aplicação.

O AngularJS já tem suporte para enviar o CSRF token, sempre que recebe um *Set-Cookie CSRFToken = XXX* na resposta do servidor, irá gravar esse csrf token num cookie no browser do utilizador e irá enviar sempre que for feito um pedido.

Já o Player foi feito usando Java e JavaFX. No java teve-se de usar uma biblioteca para fazer os pedidos HTTP e HTTPS exterior, neste caso usou-se a org.apache.http. Esta biblioteca não tem suporte para uso de um CSRF de forma correta, por isso, teve-se de implementar uma classe (Requests.java) que gerisse uma ligação HTTP e guardasse o seu estado de sessão. Nesta classe enviou-se um X-CSRFToken no Header do pedido.

No servidor, também teve-se a proteção contra SQL Injection e XSS ataques pois fez-se uso dos métodos disponibilizados pelo Django para manipulação de modelos da base de dados. Já o AngularJS não permite que seja injetado código JS nem HTML sem intenção do programador da aplicação.

Como se usou Python e Java já se teve proteção contra buffer overflows e mesmo assim teve-se cuidado para não fazer implementações que comprometessem o estado tanto do servidor como do player, como por exemplo, uma saída inesperada devido a um erro.

5 Casos de utilização

5.1 Registo de um utilizador

Um registo de um utilizador é feito na página web disponibilizada pelo IEDCS. Ao gravar na base de dados a password é gravada com um segundo PBKDF2 e SHA256, o que torna computacionalmente trabalhoso obter as passwords dos utilizadores caso a base de dados seja exposta. Ao fazer login, tanto no Player como na interface Web é aplicado PBKDF2 com HMAC e SHA1 para caso o HTTPS seja comprometido, as passwords não sejam enviadas em claro pela rede ou caso se esteja a usar HTTP.

$$\text{Password} = \text{PBKDF2}(\text{Password}, \text{Password}, 500)$$

Não existe nenhuma criação da chamada UserKey, esta no entanto, é gerada para cada livro por cada utilizador quando ele solicita pela primeira vez a leitura do mesmo. Optou-se por fazer um UserKey por Livro para dificultar ainda mais a tarefa a quem tente comprometer o sistema de IEDCS.

id	password	last_login	email	username	first_name	last_name	created_at	updated_at
...	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1 1	pbkdf2_sha256\$15000\$aoPM4j5hSUZG\$ITF...	2015-11-12 21:39:38.927...	mail@rafa...	rafaelferreira	Rafael	Ferreira	2015-11-07 23:17:21.586...	2015-11-07 23:17:21.587...

Figura 8:
Password guardada com PBKDF2 e SHA256

5.2 Registo de um device

Quando um player é utilizado pela primeira vez num determinado device, no momento do login de um utilizador, o Device é registado na base de dados com os seguintes parâmetros:

- Identificador único
- Modelo de CPU e PC
- IP
- País
- Hora
- Host name
- Chave pública do Device

No Player é usado a biblioteca `sigar` para obter os dados do device. O País é obtido no servidor através do IP recebido pelo mesmo. A cada utilizador também são associados os vários devices por onde já se conectou ao servidor. Caso no momento do login, o device já exista na base de dados, este é atualizado com os novos dados recebidos, tal como a hora, IP, entre outros já especificados.

5.3 Aquisição de um livro

No momento de aquisição do livro não há nenhuma cifra, apenas um registo na base de dados que o utilizador adquiriu um livro.

5.4 Restrições na visualização de um livro

Para a visualização de um livro criaram-se dois tipos de restrições, por país e por hora de utilização.

Isto permite que os conteúdos não possam ser acedidos a qualquer momento e em qualquer lado caso existam restrições nesse livro.

Gestão de restrições

Para ser mais simples gerir as restrições por parte do administrador do sistema foi criada uma interface de linha de comandos com várias possibilidades. Para adicionar uma restrição a um livro:

```
# python manage.py restriction add restrict_book book_identifier  
restriction_name
```

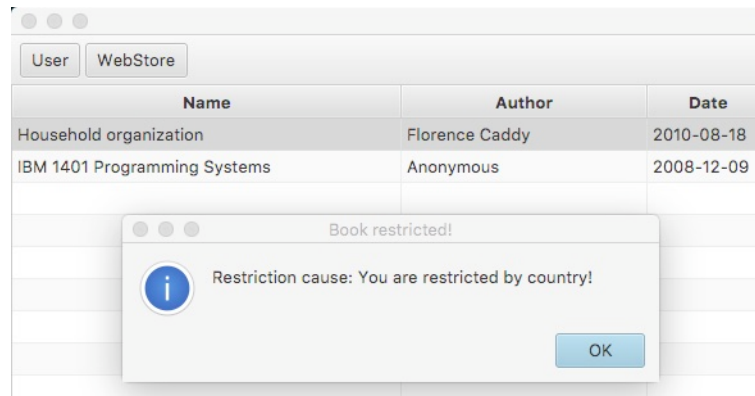


Figura 9:

Restrição apresentada tendo em conta o país do utilizador

Para adicionar uma nova restrição no sistema:

```
# python manage.py restriction add restriction restriction_name  
restrictionFunction cause
```

Para remover uma restrição a um livro:

```
# python manage.py restriction rm restrict_book book_identifier  
restriction_name
```

Para listar os livros disponíveis:

```
# python manage.py restriction list books
```

6 HTTPS

Para criar um certificado criou-se uma chave privada RSA com:

```
openssl req -new -newkey rsa:2048 -nodes -keyout server.key -out server.csr
```

output: /installation/ssl/generating.txt

O certificado foi depois emitido usando a Free SSL.

Como foi criado um domínio que aponta para o endereço *192.168.33.10* que é o endereço local do servidor IEDCS inicializado pelo Vagrant.

7 Problemas identificados na primeira entrega

Nesta primeira entrega, identificaram-se alguns problemas, nomeadamente no corrompimento do HTTPS, através da troca da keystore onde são guardados os certificados confiáveis, podendo assim injetar todos os certificados que se deseja de forma a que seja possível intercetar as comunicações entre o servidor e o Player. Para isso sabe-se que o utilizador pode descompilar o código do Player, analisar o processo de cifra e o código. Com isto ele poderá descobrir que se usou um ficheiro cacerts.keystore para o Java confiar

apenas no certificado usado, pode então modificar essa keystore e o cliente ao fazer download pode ser-lhe dada uma versão corrompida, mas sem estar corrompida no código, apenas no cacerts.keystore. Se estiver presente outro certificado que o atacante domine, ele pode intercetar as comunicações HTTPS e aceder às comunicações que estão a ser trocadas entre a vítima que está a usar o Player e o Servidor e assim, pode, obter o Random2 e derivar o n (este é enviado inicialmente num Header), o book identifier e o rd2 que é enviado pelo servidor no pedido de challenge ao cliente. O texto “ua” é possível de ser obtido descompilando o código. Posto isto, ele pode cifrar com AES e obter a file key. Com isto, consegue-se decifrar o livro cifrado.

Outra possibilidade seria um atacante/cliente que usasse ferramentas de análise de memória para obter o livro cifrado, e todas as outras variáveis que estão expostas em Runtime.

7.1 Soluções

- Assinar o Player com uma chave assimétrica assim como o ficheiro cacerts.keystore.
- Autenticando ambas as entidades no momento em que é negociada o HTTPS entre o cliente e o servidor (2º parte do trabalho).
- Sessão para a file key usada ser sempre diferente tendo em conta a sessão, proibindo a reutilização da file key em vários devices e players.
- Cifra por página, ou seja, a paginação ser realizada no lado do servidor.

8 Segunda entrega, síntese de melhorias

um género de introdução para a segunda entrega do trabalho <=====

9 Pequenas melhorias implementadas

9.1 New Secret Key

Foram tidas em conta algumas considerações sobre o servidor Django. Muitas aplicações Django mantêm a "Secret Key" ¹ a inicial que vem com outras instalações ² da mesma framework, no entanto, para evitar algum problema, decidiu-se gerar uma nova chave privada para este projeto.

9.2 Django Rest Framework

A DRF³ tem uma "browsable API"⁴ que é usada para o utilizador que pretender usar a API conseguir procurar, no entanto expor dessa forma endpoints não é aconselhável, por isso mesmo, foi retirado.

9.3 Melhoria do Debug do Player

Para o Player tinha-se desenvolvido uma interface para que quando a aplicação falha-se, fosse possível fazer debug rapidamente do que se tinha passado, no entanto, tinham ficado falhas para o modo produção, sendo uma delas que em alguns casos era exposto para o utilizador os dados recebidos nos endpoints, sendo isto não aconselhável, melhorou-se a forma como é gerido o debug e o modo produção no Player. Sendo esta melhoria não muito importante, não se irá entrar em detalhes sobre a sua implementação.

9.4 Diffie Hellman na visualização de um livro

Quando um utilizador pede ao servidor para obter o conteúdo de um livro, é agora iniciado um processo de challenge baseado na troca de chaves Diffie Hellman. O utilizador ao fazer o pedido, é efetuado um update da informação do Device, sendo que na resposta do servidor a este update, este devolve dois números partilhados **p** e **g**, e ainda um número **n**, calculado com base nos números primos partilhados e num **número primo secreto**, apenas conhecido pelo servidor. O Player ao receber a resposta do servidor com os números **p**, **g** e **n**, gera um **n** próprio baseado nos números **p** e **g** partilhados e num **segredo** que apenas ele conhece, sendo **n** enviado para o servidor para poder calcular o resultado final do challenge. É calculado

¹ <https://docs.djangoproject.com/en/1.9/ref/settings/#secret-key>

² <http://exfiltrated.com/research-Instagram-RCE.php>

³ <http://www.django-rest-framework.org/>

⁴ <http://www.django-rest-framework.org/topics/browsable-api/>

ainda o resultado final do lado do Player, com base no número partilhado **p**, **n** e ainda no **segredo** do Player. Este resultado final é enviado juntamente com o **n** calculado para o servidor no pedido de `getBook`. O servidor calcula então o resultado final, da mesma maneira que o Player, e compara os valores dos dois resultados finais. Se forem iguais o conteúdo do livro é enviado para o Player, caso contrário não envia nada.

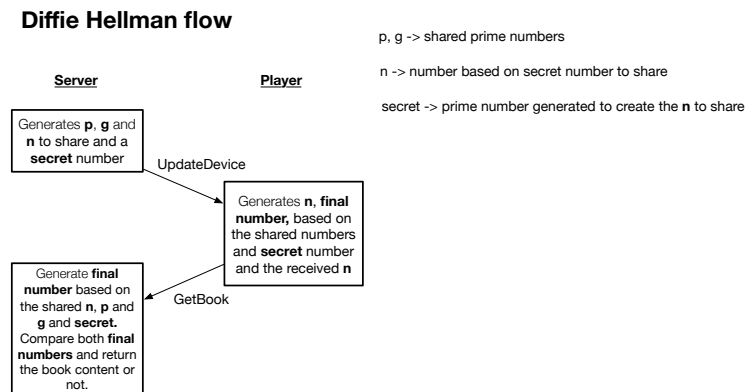


Figura 10:
Processo de Challenge Diffie-Hellman

9.5 Copy/ Paste da janela do Player

Na implementação da primeira entrega do projeto, era possível o copy/paste do texto que era apresentado ao utilizador. Para que isso não acontecesse foram efetuadas algumas alterações no JavaFx, para que não seja possível copiar o texto.

10 Novas funcionalidades

10.1 Associar Cartão de Cidadão a uma conta

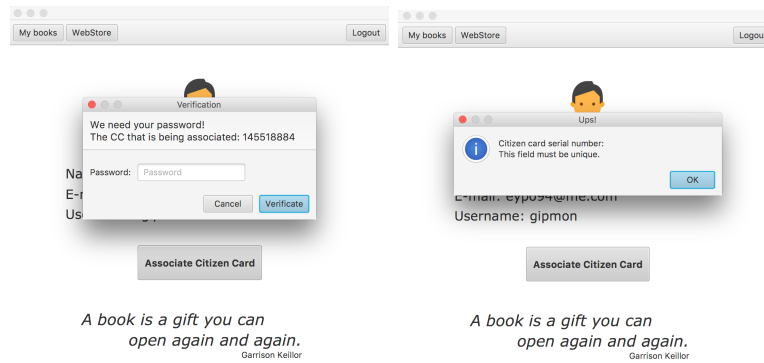


Figura 11:
Associar Cartão de Cidadão e erro ao associar

A cada conta é possível associar um Cartão de Cidadão, para isso o utilizador na sua página pessoal do Player deverá clicar no botão de "Associate Citizen Card" onde lhe será pedido, como medida de segurança, que coloque a sua password da conta de utilizador (Figura 11). Depois disto, no servidor é guardada a chave pública do Cartão de Cidadão e o seu número de identificação, que deverá ser único tal como mostra a figura 11. Depois da associação do cartão é efetuado logout automaticamente, e no próximo login aparecerá um botão de "Disassociate Citizen Card".

10.2 Desassociar Cartão de Cidadão de uma conta

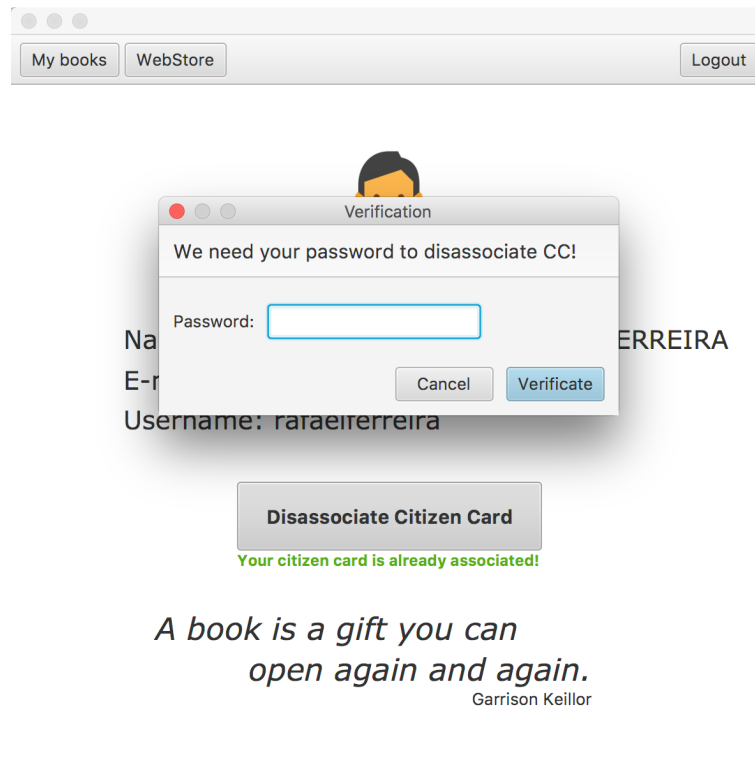


Figura 12:
Desassociar Cartão de Cidadão

É possível, depois de associar um cartão a uma conta, desassociar o mesmo. Para esta função e como medida de segurança, é também pedido ao utilizador que coloque a sua password da conta de utilizador (Figura 12). É removida então a chave pública do Cartão de Cidadão e o número de identificação do mesmo no servidor. Posto isto é efetuado logout automaticamente, e no próximo login será possível associar um outro cartão.

10.3 Login com o Cartão de Cidadão

Depois de se associar um cartão passa a ser possível fazer login com o cartão de cidadão.

Quando não está nenhum cartão de cidadão inserido, é lançada uma mensagem de erro para o utilizador como se pode ver na figura 13. Se um dado cartão não estiver associado a nenhuma conta de utilizador será então lançada a mensagem de erro que está na figura 13.

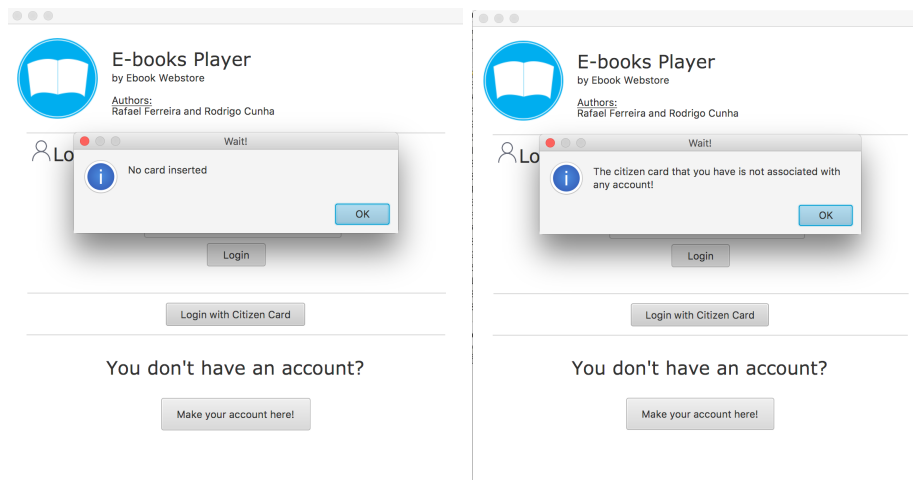


Figura 13:
Nenhum Cartão de Cidadão inserido ou associado

10.4 Logout

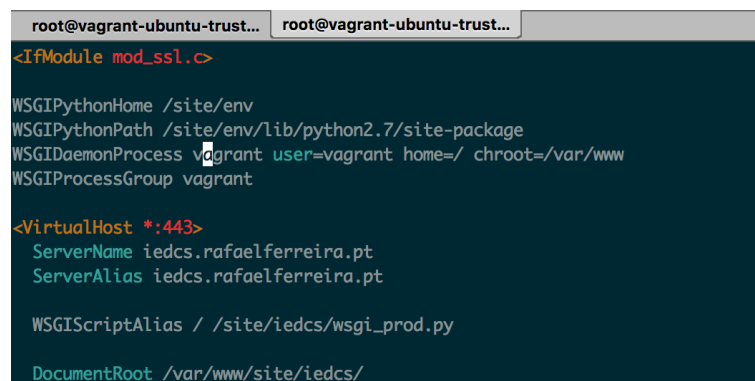
Uma nova funcionalidade que não se tinha implementado na primeira entrega, é a possibilidade de fazer logout para trocar de conta ou sair dela no Player.

11 Confinamento

Foram realizadas várias tentativas para fazer o confinamento da aplicação e limitar ao máximo o que a aplicação poderia fazer caso fosse comprometida e executasse código malicioso. Para isso, inicialmente, realizaram-se várias tentativas para implementar uma solução com *chroot* e para a solução final usou-se Docker.

11.1 Chroot with Apache

O `mod_wsgi`⁵ na versão 0300⁶ anunciou a implementação de uma diretiva que permitia fazer o *chroot* de um *WSGIProcess*. No entanto, esta funcionalidade está muito pouco documentada⁷.



```
root@vagrant-ubuntu-trust... root@vagrant-ubuntu-trust...
<IfModule mod_ssl.c>

WSGIPythonHome /site/env
WSGIPythonPath /site/env/lib/python2.7/site-package
WSGIDaemonProcess vgrant user=vagrant home=/ chroot=/var/www
WSGIProcessGroup vgrant

<VirtualHost *:443>
    ServerName iedcs.rafaelferreira.pt
    ServerAlias iedcs.rafaelferreira.pt

    WSGIScriptAlias / /site/iedcs/wsgi_prod.py

    DocumentRoot /var/www/site/iedcs/
```

Figura 14:
Parte da implementação do *chroot*

No entanto, tentou-se a implementação, representada na figura 14, usando esta diretiva, os resultados obtidos usando a configuração, estão apresentados na figura

Com isto, foi abandonada a implementação de *chroot* com o apache para impedir que o python executa-se código de forma maliciosa e procurou-se outra solução.

⁵ <https://code.google.com/p/modwsgi/>

⁶ <https://code.google.com/p/modwsgi/wiki/ChangesInVersion0300>

⁷ <https://groups.google.com/d/msg/modwsgi/sOrtcm7JV50/YmriFHUmHI8J>

```

Sat Dec 19 14:56:16.212834 2015 [info] [pid 12858:tid 139684949124992] mod_wsgi (pid=12858): Python home BASELINE.
Sat Dec 19 14:56:16.212834 2015 [info] [pid 12858:tid 139684949124992] mod_wsgi (pid=12858): Initializing Python.
ImportError: No module named site
Sat Dec 19 14:56:16.214004 2015 [info] [pid 12849:tid 139684949124992] mod_wsgi (pid=12849): Starting process 'wsgiproxy' with uid=33, gid=33 and threads=15.
Sat Dec 19 14:56:16.215068 2015 [info] [pid 12849:tid 139684949124992] mod_wsgi (pid=12849): Python home BASELINE.
Sat Dec 19 14:56:16.215078 2015 [info] [pid 12849:tid 139684949124992] mod_wsgi (pid=12849): Initializing Python.
Sat Dec 19 14:56:16.222012 2015 [info] [pid 12849:tid 139684949124992] mod_wsgi (pid=12849): Attach interpreter ''.
Sat Dec 19 14:56:18.215206 2015 [info] [pid 12873:tid 139684949124992] mod_wsgi (pid=12873): Python home BASELINE.
Sat Dec 19 14:56:18.215248 2015 [info] [pid 12873:tid 139684949124992] mod_wsgi (pid=12873): Initializing Python.
ImportError: No module named site

```

Figura 15:
Import Error: No module named site

11.2 Docker

Para obter resultados semelhantes a um chroot pensou-se em usar o Docker. No entanto não se pode pensar só em usar Docker, teve-se de pensar numa arquitetura para o servidor e o contentor docker. Para isso, a principal preocupação, foi em ter as chaves privadas do servidor usadas para o HTTPS, fora do contentor, retirando assim possibilidades a que o script python, que disponibiliza o serviço web tivesse possibilidades de aceder a tais chaves.

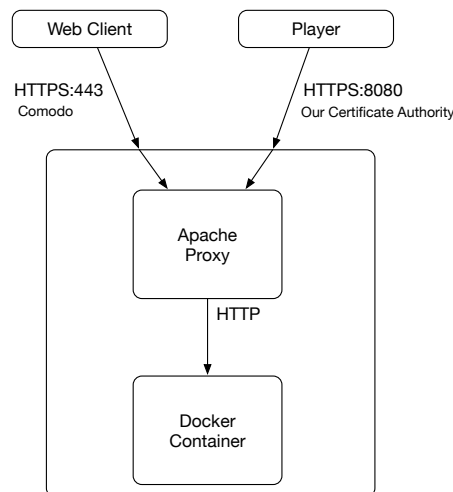


Figura 16:
Arquitetura servidor e docker container

Quando um cliente acede pelo browser ao site da loja, é-lhe fornecido um certificado assinado pela COMODO, Free SSL, que o cliente usará para navegar no site sem problemas. Este pedido, que é negociado com o Apache no servidor principal, é depois redirecionado por HTTP para o Docker container que é responsável por receber e tratar os pedidos junto com o código python (Django). Desta forma, como os pedidos apenas são tratados no container e a negociação para o HTTPS é negociada fora do container, as chaves privadas estão seguras em relação ao container.

Já quando o Player comunica com o servidor, apenas aceita pedidos do

servidor que lhe apresente um certificado específico que foi anteriormente assinado pela nossa entidade certificadora criada para o efeito. Estes pedidos são depois encaminhados de igual modo para o container Docker.

Comportamentos observados

Foram realizados alguns testes com código python inserido de forma intencional no código do servidor e observou-se:

- Apenas tem privilégios para criar diretórios e ficheiros em pastas que pertençam ao utilizador ou grupo "www-data".
- Não tem acesso as chaves privadas usadas para o HTTPS
- Tem acesso à chave do Player e a todas as chaves públicas de cada device
- Consegue ver e listar os diretórios do docker container
- Consegue aceder à base de dados

Proibição de determinados URL no Apache

Foi tida em conta a preocupação para que o utilizador apenas conseguisse usar os endpoints para o Player na porta destinada isso, a porta 8080, para isso foi criada uma regra na porta 443 para que pedidos para os endpoints que são utilizados pelo Player fossem descartados e não fosse feito o forward para o Docker container.

Isolamento da chave privada do Player

Apesar de não ter sido desenvolvido nenhuma solução para o isolamento da chave do Player, por falta de tempo, tinha sido idealizado uma solução baseada num container que apenas guardaria a chave privada do Player e um endpoint para assinar o que lhe fosse pedido para assinar internamente. Retirando assim a chave privada do container onde está a correr o web service Python.

```
<LocationMatch "/api/v1/player/*">  
    Require all denied  
</LocationMatch>
```

Figura 17:

Proibição de acesso a endpoints do player na porta 443

Acesso ao container Docker apenas de dentro do servidor

Para apenas permitir o acesso de dentro do servidor principal ao Docker (para fazer o forward dos pedidos HTTP) usou-se as iptables do Ubuntu para fazer deny a todos os pedidos para a porta 8002. No entanto, como é usado um "Host-only adapter", o IP que o host e a máquina virtual têm são o mesmo, o impossibilitou que fosse testado de maneira mais simples.

```
# iptables -I FORWARD -dport 8002 -j DROP

# iptables -I DOCKER -p tcp -dport 8002 -j DROP

# iptables -I INPUT -p tcp -dport 8002 -j DROP
```

11.3 Conteúdo cifrado no servidor

Os livros são cifrados => cifrado o media => os livros nunca estão guardados de forma não cifrada => explicar as opções que estão no server bootstrap, linha 6

AQUI RODRIGO

12 Autenticação do Player junto do Servidor

Sendo este um requisito explícito do trabalho, os resultados não foram os esperados, pelo menos com a implementação que se foi conseguindo fazer. Usando o Java não foi conseguido negociar de forma correta o TLS para que obrigando o TLS client a autenticar-se junto do servidor, este se autenticasse.

12.1 Jar sign

Para que não seja possível alterações no código fonte do Player, o ficheiro jar deste foi assinado por uma chave guardada na KeyStore JarSignature.KeyStore, de forma a que apenas seja possível correr o código do Player disponibilizado por nós e não por outra fonte desconhecida.

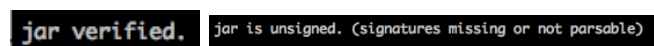


Figura 18:
Verificação da assinatura

Se a assinatura é verificada e validada o programa corre sem qualquer problema, caso contrário, é impossibilitada a execução do mesmo por parte do utilizador.

12.2 Certificados assinados pela CA do IEDCS

Apesar das sucessivas tentativas que foram feitas para que a autenticação da negociação do TLS fosse efetuada com sucesso usando Java, não foi possível implementar algo que o fizesse. Outro problema que se teria se optasse por usar certificados assinados pela CA do IEDCS é que seria necessário gerar uma chave única do Player junto com o Device para cada instalação e enviar a sua chave pública para o servidor que assinaria toda as chaves. Concluiu-se que essa implementação teria poucos benefícios para além do que já foi implementado.

No entanto foi criada a nossa Certificate Authority para emitir certificados para cada chaves de cada versão do Player. Estas são usadas para assinar cada livro que é enviado para o player desde o servidor. A chave privada está no servidor e a chave pública está em cada implementação do Player, hardcoded para que não seja possível alterá-la, dado que o player está assinado.

Encrypt then MAC

A assinatura do conteúdo do livro é feita fazendo uma síntese usando o algoritmo SHA-256 do conteúdo cifrado e depois usando a chave privada do Player é assinada essa síntese e enviada junto com a mensagem do Player. Por sua vez, o Player, usando a chave pública irá verificar se o conteúdo que recebeu corresponde ao que foi assinado. Este mecanismo só pode ser comprometido se a private key do player saia do servidor ou alguém tenha acesso de forma indesejada a esta. Se um atacante conseguir meter no meio e pretender alterar o conteúdo da mensagem precisa da chave privada e pública para alterar o conteúdo além de todo o outro processo de cifra que já foi desenvolvido. A chave pública que está guardada no código fonte do Player não pode ser alterada devido ao Jar Sign que se implementou.

12.3 Autenticação usando o Cartão de Cidadão

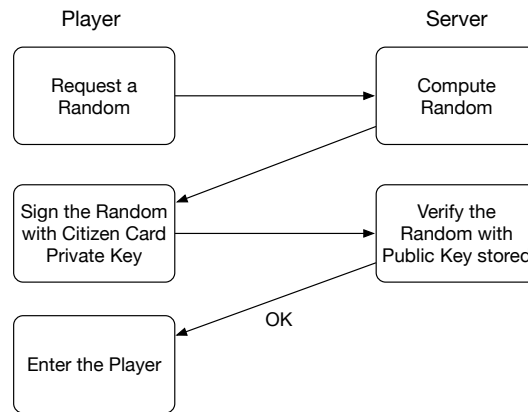


Figura 19:
Processo de autenticação usando o cartão de cidadão

A autenticação do cartão de cidadão baseia-se num pedido ao servidor de um número completamente aleatório e único, uuid4, que é depois enviado na resposta a esse pedido para o Player e na assinatura por parte do Player usando a chave privada de autenticação do Cartão de Cidadão desse mesmo valor único e aleatório.

Esse mesmo número é gravado na base de dados para que quando o Player iniciar o pedido de Login seja verificado se esse valor único existe na base de dados, se não existir, a autenticação falha, se existir e a assinatura corresponder ao valor que se pretendeu que fosse assinado, o utilizador faz Login.

A validação da assinatura é realizado usando a chave pública de autenticação do Cartão de Cidadão que irá verificar a assinatura produzida no Player, e sendo isto tudo no lado do servidor.

O servidor sabe que chave pública usar para cada situação porque é-lhe enviado o número de identificação do Cartão de Cidadão, que também é único. Assumimos que o utilizador associou o seu cartão de cidadão à sua conta posteriormente.

12.4 Identificação do cidadão junto do servidor

Um dos objectivos do projeto era a identificação do cidadão no servidor usando o TLS Handshake e enviando uma "Client Authentication", depois o servidor verificaria se a autenticação ou identificação fornecida estaria correta. Pensou-se isto usando a "Web Authentication" tal como se tinha feito nas aulas. Começou-se por fazer como nas aulas práticas, através do Firefox, sendo que foi logo conseguido sem qualquer problema, contudo não era o que

se queria, pois queria-se que houvesse uma autenticação de ambas as partes no negociação do TLS. Passou-se então à implementação desta autenticação no lado do Player, onde foram efetuadas muitas tentativas de forma a que tal fosse possível. Para esta autenticação idealizada, em que ambas as partes se autenticam, é sempre necessária a KeyStore que possui o certificado do Servidor e a KeyStore PKCS11 proveniente do Cartão de Cidadão utilizada para a autenticação através do mesmo e que contém os certificados e as chaves do cartão. É necessário ainda a criação de um TrustManager que contém os certificados confiáveis e uma KeyManager, utilizada para certificados de autenticação. O primeiro problema surgiu quando na criação de um SSLContext se enviava um array de TrustManager, e apenas o primeiro elemento era utilizado, sendo que quando era carregado primeiro o certificado do Servidor, os pedidos que não requeriam autenticação do cartão de cidadão era efetuados com sucesso, contudo os que requeriam autenticação, davam sempre um erro de "handshake_failure", pois os certificados do Cartão de Cidadão não estavam no primeiro elemento do array de TrustManager. Quando se colocava os certificados do Cartão de Cidadão no primeiro elemento do array e o certificado do Servidor no segundo, então obtínhamos outro erro que não nos permitia sequer fazer pedidos ao servidor, mais uma vez porque seriam apenas utilizados os certificados do primeiro elemento do array. Tentou-se então que o certificado do Servidor estivesse na KeyStore da JVM, mas depois de se ter colocado o certificado nessa mesma KeyStore o erro permaneceu, não conseguindo aceder ao Servidor. Com isto juntaram-se então os certificados todos numa KeyStore, de forma a existir apenas um TrustManager e assim carregar os certificados todos. Com isto deixou-se de ter o erro de handshake failure e passou-se a ter um de erro na verificação da assinatura, que segundo o professor, seria porque a cadeia de certificados não estaria completa. Posto isto pensou-se que poderia ser o facto de se estar a juntar tudo numa única keystore, e então criou-se um TrustManager que juntava os certificados de ambas as KeyStores e verificava as cadeias de certificação, contudo o erro persistia. Numa última alternativa, foram colocados numa KeyStore todos os certificados obtidos pela MakeFile da aula prática que obtinha todos os certificados referentes ao Cartão de Cidadão, assim a cadeia à partida e juntando com os certificados do Cartão, estaria completa. No entanto o erro permaneceu.

12.5 Validação do cartão de cidadão usado pelo utilizador

Depois de vários dias perdidos a investir nesta funcionalidade, decidiu-se desistir da ideia, de forma a ser possível avançar na realização do trabalho.

Para isto através de uma classe criada pelo Professor André Zúquete e editada por nós de forma a adaptar-se ao projeto, antes de cada pedido ao servidor onde é utilizado o Cartão de Cidadão, este é validado, percorrendo toda a cadeia de certificados e CRL's de forma a verificar que nada no cartão está fora de validade, ou foi revogado. Posto isto, caso o cartão seja verificado com sucesso, a operação desejada é efectuada, caso contrário é apresentado o erro ao utilizador.

13 Instalação

Para uso e instalação do sistema, criou-se uma pasta `/installation` que tem um ficheiro *Vagrantfile* que permite criar um servidor ubuntu trusty 64bits, cria uma private network e uma pasta partilhada com o servidor python em `/vagrant/Docker/python/site`. O ficheiro *server-bootstrap.sh* servirá para instalar todas as bibliotecas necessárias, irá instalar o docker e o Apache com os modos necessários para se fazer proxy dos pedidos para o docker container. A configuração do Apache2 para o site que está no ficheiro *iedcs.rafaelferreira.pt.conf* e *default-ssl.conf*, também para instalar os certificados necessários para o HTTPS nas pastas usadas para esse efeito e para criar a base de dados com os livros e restrições exemplo.

O ficheiro *iedcs.rafaelferreira.pt.conf* serve apenas para redirecionar todos os pedidos HTTP para HTTPS. O ficheiro *default-ssl.conf* tem todas as configurações necessárias para o uso de HTTPS e para usar o `mod_proxy` do Apache2 para o Django.

Criou-se um package de settings para o Django para ter definições de development e outras de servidor. Usando o hostname do servidor sabe-se em que modo deve estar, sendo assim nesse servidor iniciado com Vagrant o servidor estará em modo produção, ou seja, não irá lançar output de debug para o utilizador.

Para usar o servidor, além de ter o Vagrant instalado, é apenas necessário fazer:

```
# vagrant up
```

A password caso seja pedida é sempre vagrant e para aceder à máquina é apenas necessário fazer:

```
# vagrant ssh
```

O site pode ser acessível em: <https://iedcs.rafaelferreira.pt/>

Para fazer download do Player apenas é preciso registar na webstore e depois clicar em "Player" que será feito o download do Player. A implementação do Player está apenas disponível para Mac OS X, com uma aplicação

native application tendo como base uma aplicação JavaFX. Testado com o java version "1.8.0_60".

Docker

Quando se inicia a máquina depois de fazer um "vagrant halt" deve-se executar o script `"/vagrant/start-up.sh"`. Se necessitar de fazer reload do apache foi criado um script para esse efeito em `"/vagrant/reload-apache.sh"`.

A Dockerfile está disponível em `/installation/Docker/python`.

14 Conclusão

NOVA CONCLUSÃO

Esta conclusão não é uma conclusão final, pois este projeto será realizado em duas partes. Sendo assim vai-se optar por nesta conclusão dizer principalmente as implementações futuras e problemas encontrados.

O principal problema encontrado, foi fazer a decifra no lado do Java, depois de a cifra estar feita do lado do Python, pois muitas tecnologias disponibilizadas por este não estão disponíveis diretamente no Java, o que nos obriga a utilizar bibliotecas externas o que nem sempre é bom, pois pode-se estar a acrescentar problemas ao sistema.

As implementações futuras que não se conseguiu implementar a tempo nesta fase do projeto, mas que contudo poderão fazer parte da próxima são:

- Uma sessão única sempre o device atualiza os seus dados. Este atualiza os seus dados sempre que o utilizador efetua um inicio de sessão, tenta ler um livro e muda de página. Esta solução faria com que a cifra do conteúdo do livro apenas fosse válida durante a sessão, pois assim que fosse efetuado um outro challenge com o servidor a sessão iria mudar e o Player não iria conseguir fazer a decifra.
- A solução apresentada acima, obriga a que seja feita a cifra página a página, consoante a sessão do Player, Device e utilizador.
- Outra ideia que nos surgiu, foi apresentar ao utilizador as duas páginas iniciais de cada EBook (Sample) de forma gratuita.

Referências

- [1] André Zúquete, *Segurança em Redes Informáticas*, 4ª Edição Aumentada, FCA - Editora de Informática, Lda.
- [2] <http://stackoverflow.com/>
- [3] Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy <http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html>
- [4] <https://support.hyperic.com/display/SIGAR/Home>
- [5] <https://www.digitalocean.com/community/tutorials/how-to-create-a-ssl-certificate-on-apache-for-ubuntu-14-04>
- [6] <http://www.vagrantcloud.com>
- [7] <https://hc.apache.org/>
- [8] PKCS #5 e AES para a implementação em Python <https://tools.ietf.org/html/rfc2898>