



**universidade de aveiro**

Departamento de Eletrónica, Telecomunicações e Informática

## **Segurança**

# IEDCS: Identity Enabled Distribution Control System

Curso	[8240] MI em Engenharia de Computadores e Telemática
Disciplina	[47232] Segurança
Ano letivo	2015/2016

Alunos	[64090] Rui Lebre [68129] Tomás Rodrigues
Prática	P1
Docente	Professor João Paulo Barraca

Aveiro, 16 de Novembro de 2015

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Abordagem</b>	<b>2</b>
2.1	Projeção inicial . . . . .	2
2.2	Alterações à projeção . . . . .	3
<b>3</b>	<b>Base de dados</b>	<b>4</b>
3.1	Tabelas . . . . .	4
3.2	Stored Procedures . . . . .	5
3.3	Triggers . . . . .	7
<b>4</b>	<b>Modelo Cliente-Servidor</b>	<b>9</b>
4.1	IEDCS Cliente/Player . . . . .	10
4.1.1	Descrição . . . . .	10
4.1.2	Implementação . . . . .	11
4.2	IEDCS Servidor . . . . .	12
4.2.1	Descrição . . . . .	12
4.2.2	Implementação . . . . .	13
<b>5</b>	<b>Cifragem de mensagens trocadas</b>	<b>14</b>
5.1	Modos de cifra . . . . .	14

5.2	Chaves . . . . .	15
<b>6</b>	<b>Conclusões</b>	<b>17</b>
<b>7</b>	<b>Notas finais</b>	<b>17</b>
<b>8</b>	<b>Bibliografia</b>	<b>19</b>

# 1 Introdução

Pretende-se através deste relatório expor sob forma escrita, o nosso desempenho, metodologia e objetivos alcançados no desenvolvimento de um Digital Rights Management (DRM), proposto no âmbito da unidade curricular de Segurança incidente na componente de avaliação prática.

A solução apresentada visa a distribuição de ebooks apenas a utilizadores que tenham adquirido os direitos de utilização (i.e. leitura) de um ou vários títulos, em que o armazenamento no servidor e transmissão de documentos para o cliente é sempre feita de forma segura e confidencial. Tais requisitos serão implementados aplicando os conceitos já adquiridos ao longo da UC em questão, nomeadamente, nas áreas da criptografia, gestão de chaves públicas e autenticação.



Figura 1: Digital Rights Management

## 2 Abordagem

### 2.1 Projeção inicial

Na apresentação inicial, a projeção feita apontava para o desenvolvimento de um serviço web usando Java. Estas opções foram inicialmente idealizadas dada a proximidade tida com a linguagem de programação JAVA e ser necessário usar serviços acessíveis e bem documentados por forma a otimizar o tempo dispendido no verdadeiro objetivo do projeto da Unidade Curricular, a criptografia.

No que toca à base de dados, dada, mais uma vez, a familiarização com MS SQL Server, optou-se por avançar neste campo o que, tal como se terá em conta mais à frente, não se concretizou. Quanto, ainda no campo da base de dados, ao esquema relacional, este encontra-se definido mais à frente, na Figura 1.

Por último, vai ser desenvolvido o tema das chaves. Ao longo dos próximos temas vão ser apresentadas as chaves usadas no projeto, passando a enumerar: User Key, Device Key, Player Key e File Key. Inicialmente, pensou-se que ambos os cliente e servidor teriam acesso a todas as chaves, o que não se tornou verdade no decorrer do desenvolvimento. A User Key é uma chave única gerada para cada utilizador e que permite a cifragem de um ficheiro para um utilizador específico; a Device Key permite a cifragem do mesmo ficheiro atrás mencionado para o dispositivo em questão, tornando-se, assim, impossível a sua reprodução numa outra máquina; do mesmo raciocínio da Device Key, podemos inferir que a Player Key permite cifrar os ficheiros para um reprodutor em específico, além de se conseguir validar a sua implementação através da comparação da chave com as chaves presentes na base de dados; por último, mas não menos importante, temos a File Key: esta chave é gerada aleatoriamente para cada transferência de um ficheiro e

a sua descoberta por parte do player depende intrínsecamente de cooperação com o servidor. Com isto, quer dizer, que um reprodutor, mesmo tendo acesso ao ficheiro, não o consegue reproduzir sem ter ligação com o servidor.

## 2.2 Alterações à projeção

As alterações profundas em relação à projeção têm o seu início no meio de comunicação. Isto é, foi prevista a utilização de serviços REST para a implementação da interface do servidor para com os players. No entanto, devido a haver facilidade dos membros do grupo 8 na implementação das comunicações por sockets de texto plano, optou-se por esta mudança e, assim, eliminar a possibilidade de atrasos no cumprimento de prazos e metas que a aprendizagem de REST implicaria e que, naturalmente, não iria ser tido em atenção no sufrágio avaliativo a que o trabalho vai estar sujeito, uma vez que não se trata, de todo, este o seu objetivo.

Atendendo à base de dados, alterou-se o Sistema de Gestão de Base de Dados (SGBD) de MS SQL Server para MySQL. A razão desta alteração prende-se ao facto da portabilidade do servidor para diferentes sistemas operativos. Visto que a sintaxe entre os dois SGBDs é idêntica e as prováveis diferenças de *performance*, a este nível académico, é pouco relevante, pensa-se que apenas existem vantagens nesta mudança.

Ao contrário do explícito na apresentação, que mencionava o uso de cifras DES e trocas de chaves utilizando Diffie-Hellman, as alterações relevantes no campo da criptografia foram a utilização de AES em lugar de DES, dada a otimização do Advanced Encryption Standard nos CPUs atuais e tamanho reduzido das chaves DES. Posteriormente, verificou-se que o uso de Diffie-Hellman se tornou obsoleto e não se justificava a sua utilização.

## 3 Base de dados

### 3.1 Tabelas

Na figura abaixo podemos observar a base de dados que suporta o nosso projeto e as relações entre as tabelas.

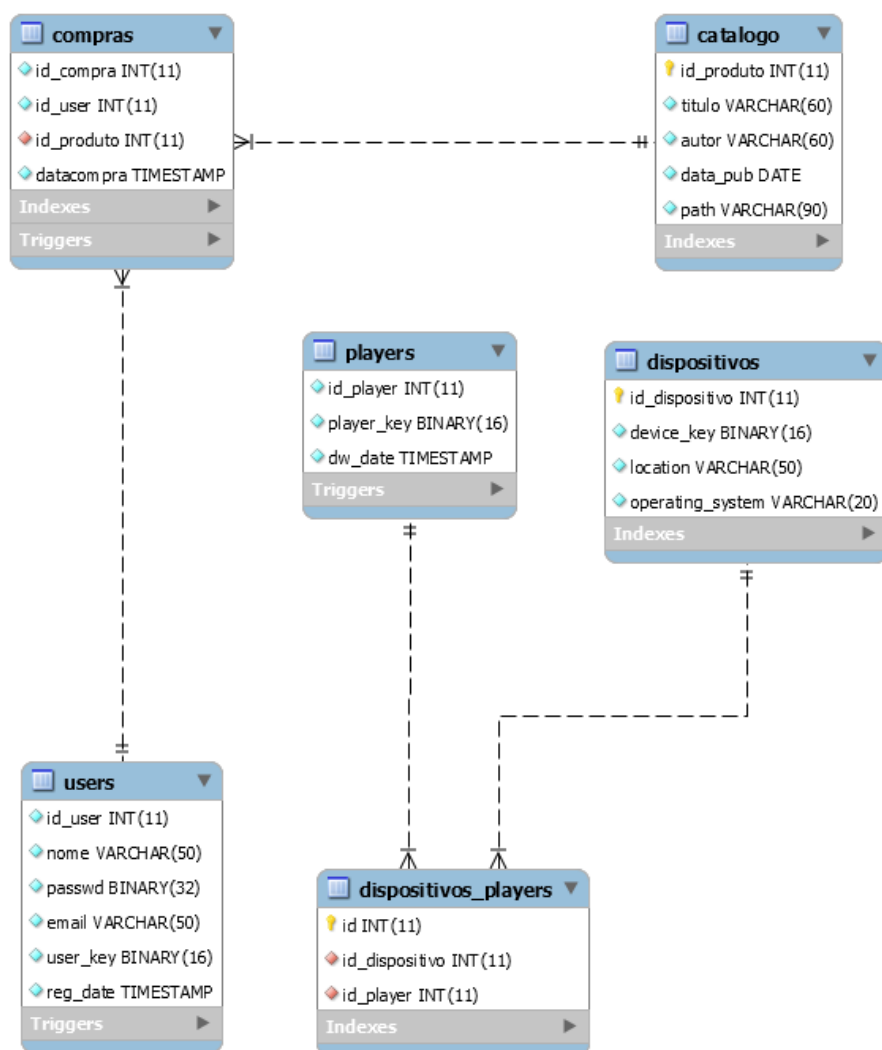


Figura 2: Base de dados

## 3.2 Stored Procedures

Os Stored Procedures são instruções frequentemente utilizadas pela base de dados. Facilitam a interacção com a aplicação (mais rápidos, menos sujeitos a erros, facilmente alteráveis sem necessidade de alterar a aplicação).

No decorrer do projeto, foi opção aplicar *Stored Procedures* a todas as interações entre o servidor e a base de dados, limitando assim as operações permitidas e, por conseguinte, solidificando a segurança neste setor.

Enumerando os *Stored Procedures* utilizados, temos:

### **changeEmail**

Dado um utilizador, atual email do utilizador e email pretendido, se todos os dados forem corretos, altera o email associado ao usuário em questão

### **changePw**

Dado um utilizador, atual *password* e *password* pretendida se todos os dados forem corretos, altera a *password* associada ao usuário em questão

### **checkOnUserCart**

Dado um utilizador e um item ID, retorna **True** se o utilizador adquiriu o item em questão e **False** caso contrário

### **checkPlayerExistance**

Dada uma have de um Player, retorna **True** se o player é autorizado e **False** caso contrário

### **getCatalog**

Retorna o catálogo existente na base de dados

### **getPath**



Dado um Item ID, retorna o caminho no sistema de ficheiros do servidor para o ficheiro correspondente ao item

**getUserKey**

Dado um utilizador, retorna a User Key correspondente

**purchaseItem**

Dado um utilizador e um Item ID, procede à aquisição por parte do utilizador especificado ao item pretendido

**rebuyItem**

Dado um utilizador e um Item ID, procede à renovação do numero de transferências permitidas do item para o dado utilizador

**registerDevice**

Procede ao registo de um novo dispositivo onde o Player foi executado, caso esse dispositivo ainda não esteja registado; regista a chave correspondente, a localização e o sistema operativo

**registerPlayer**

Regista um novo Player na base de dados. Este registo é feito pelo servidor aquando o download do Player

**registerProduct**

Regista um novo produto no catálogo do servidor

**registerUser**

Regista um novo utilizador na base de dados

**retrievePath**

Dado um item, retorna o seu caminho no servidor, decrementando em 1 unidade o número de transferencias permitidas do item para o utilizador especificados

### searchCatalog

Retorna os resultados de uma procura por um nome no catalogo presente na base de dados

### searchUserCatalog

Retorna os títulos comprados pelo utilizador especificado

### validateUser

Dado um utilizador e respetiva palavra-chave (sob a forma de um *digest-hash*), retorna a sua validade perante os dados contidos na base de dados

Na figura 2 temos vários exemplos de Stored Procedures presentes na base de dados.

```
CREATE DEFINER='root'@'localhost' PROCEDURE `validateUser`(IN _nome varchar(50), IN _passwd binary(32), out b boolean)
BEGIN
    DECLARE counter INT;
    set counter = 0;
    SELECT COUNT(users.id_user) INTO counter FROM users WHERE users.nome=_nome AND users.passwd=_passwd;

    if counter > 0 then
        set b = true;
    else
        set b = false;
    end if;

    select b;
END ;;

CREATE DEFINER='root'@'localhost' PROCEDURE `getCatalog`()
BEGIN
    SELECT * FROM catalogo;
END ;;

CREATE DEFINER='root'@'localhost' PROCEDURE `registerProduct`(in _titulo varchar(50), in _autor varchar(30), in _data date)
BEGIN
    insert into catalogo (id_produto, titulo, autor, data_pub) values (0, _titulo, _autor, _data);
END ;;
```

Figura 3: Exemplos de stored procedures

## 3.3 Triggers

Os *Triggers* são utilizados na nossa base de dados para garantir que a inserção de dados na BD é feita corretamente. Oferecem também um melhor feedback sobre alterações efectuadas na base de dados.

Enumerando os *Triggers* utilizados, temos:

**date\_trigger\_compras**

Quando é inserida uma nova compra na base de dados, altera a data de compra para a data atual do SGBD

**date\_trigger\_player**

Quando é feita a transferência de um novo player, a data de *download* é alterada para a data atual do SGBD

**date\_trigger\_users**

Quando é feito o registo de um novo utilizador, a data de registo é alterada para a data atual do SGBD

Alguns exemplos podem ser contemplados a seguir:

```
CREATE DEFINER='root '@localhost ' TRIGGER date_trigger_compras  
BEFORE INSERT ON compras FOR EACH ROW  
    SET NEW.datacompra = current_timestamp()
```

```
CREATE DEFINER='root '@localhost ' TRIGGER date_trigger_players  
BEFORE INSERT ON players FOR EACH ROW  
    SET NEW.dw_date = current_timestamp()
```

```
CREATE DEFINER='root '@localhost ' TRIGGER date_trigger_users  
BEFORE INSERT ON users FOR EACH ROW  
    SET NEW.reg_date = current_timestamp()
```

## 4 Modelo Cliente-Servidor

A arquitetura do sistema baseia-se num modelo cliente-servidor. Há um servidor que recebe comunicações num par IP/porta fixo e que por cada ligação dá-se a criação de um *thread* em que é atribuído um novo porto de comunicação através de um socket seguro através de SSL/TLS e que fará o controlo da ligação do cliente em questão com o sistema. Focando agora no cliente, ao ser executado, este é ligado ao servidor e posteriormente ao respetivo *handler*. Partindo deste ponto, é apresentado ao utilizador um ecrã onde se dará início a toda a interação.

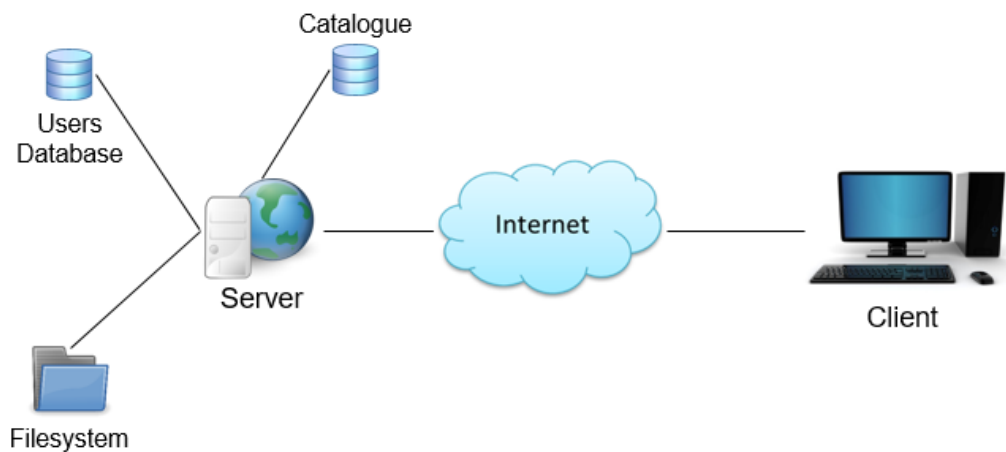


Figura 4: Arquitetura geral

## 4.1 IEDCS Cliente/Player

### 4.1.1 Descrição

Os clientes depois de estarem registrados e devidamente logados podem efetuar compras, ver os seus produtos adquiridos, usá-los e editar as suas informações pessoais. O diagrama de use-cases dos clientes pode ser observado na figura abaixo:

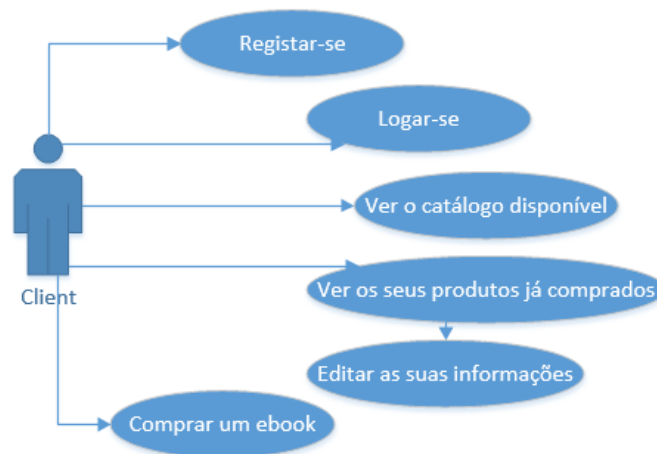


Figura 5: Diagrama de use cases do Cliente

Aqui estão alguns exemplos do que o client vê ao utilizar as opções que tem à sua escolha depois de se loggar no servidor:

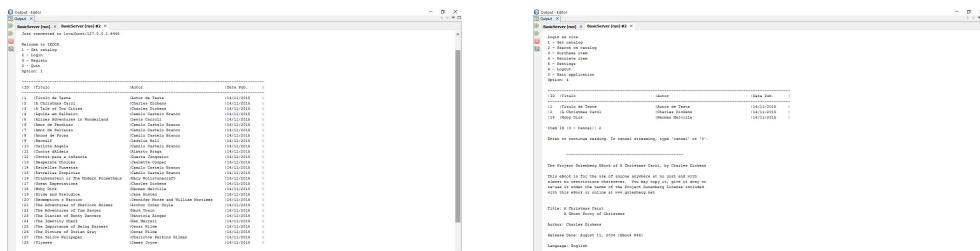


Figura 6: Ver catálogo e ler um ebook já comprado

### 4.1.2 Implementação

Para o desenvolvimento do cliente foram desenvolvidas duas principais classes. Começando pela classe `Client.java`, este contém o main do cliente como também instruções de execução. Aqui é inicializado a ligação para com o servidor através de um socket SSL/TLS e, para isso, usando um certificado auto-certificado para acerto de chaves.

Após esta inicialização, é executado o método `run()` que iniciará as comunicações com o servidor, acionado assim toda a interação com o utilizador.

A figura abaixo dá-nos uma boa prespetiva de tudo o que o acontece desde o login do cliente até à compra e utilização dum e-book:

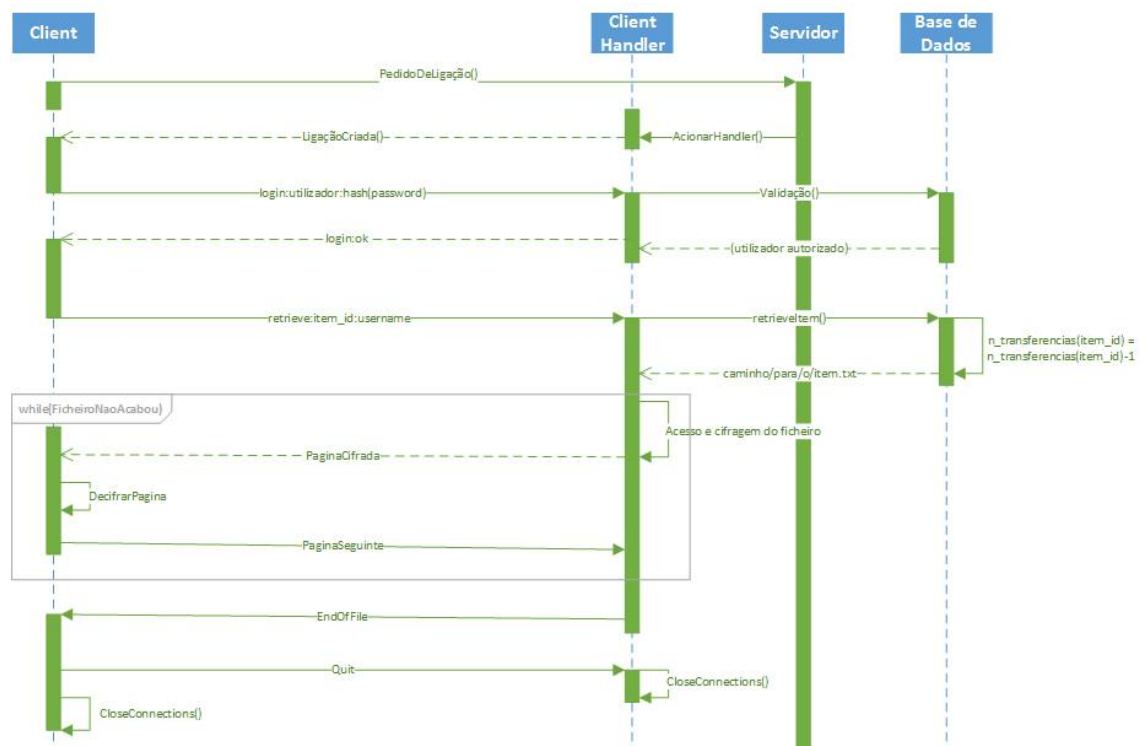


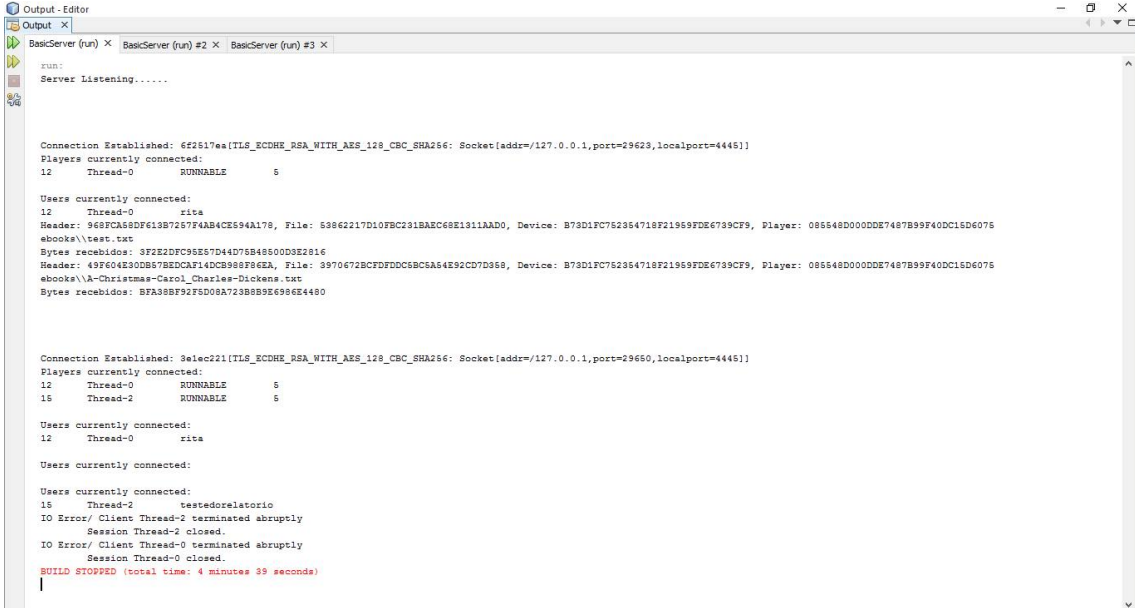
Figura 7: Diagrama de Sequência do Cliente

## 4.2 IEDCS Servidor

### 4.2.1 Descrição

O servidor tem acesso à base de dados vista anteriormente e contém um catálogo de onde os clientes podem escolher o produto a comprar. Este servidor dá acesso a vários utilizadores através do lançamento de um *handler* (*thread*) para cada player que pede ligação.

Quando é feita uma ligação ao servidor, esta deve conter no socket uma mensagem codificada em Base64 (ver Notas Finais) constituída por *device\_id:player\_id:localizacao:s\_operativo* em que posteriormente: é feita uma verificação da existência do Player na base de dados e envio da respetiva autorização; é feito o registo do dispositivo na base de dados, caso este não esteja já referenciado nela.



```
Output - Editor
Output X
BasicServer (run) X BasicServer (run) #2 X BasicServer (run) #3 X

run:
Server Listening.....

Connection Established: 662517ea[TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256: Socket[addr=/127.0.0.1,port=29623,localport=4445]]
Players currently connected:
12 Thread-0 RUNNABLE 5

Users currently connected:
12 Thread-0 rita
Header: 969P2AS8DF613B75774AB4CE594A179, File: 53862217D10FBC231BAEC68E1311AAD0, Device: B73D1FC752354718F21959F0E6739CF9, Player: 085548D00DD0E7487B99F40DC15D6075
ebooks\test.txt
Bytes recebidos: 3F2E2DFC95E57D4D75B48500D3E2816
Header: 49F604E30DB57BEDCAF14DCB989F86EA, File: 3970672BCFDFDDCS8CA54E92CD7D358, Device: B73D1FC752354718F21959F0E6739CF9, Player: 085548D00DD0E7487B99F40DC15D6075
ebooks\A-Christmas-Carol-Charles-Dickens.txt
Bytes recebidos: BFA38BF92F5D08A723B8B95E696E4480

Connection Established: 3e1ec221[TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256: Socket[addr=/127.0.0.1,port=29650,localport=4445]]
Players currently connected:
12 Thread-0 RUNNABLE 5
15 Thread-2 RUNNABLE 5

Users currently connected:
12 Thread-0 rita

Users currently connected:
15 Thread-2 testedorelatorio
IO Error/ Client Thread-2 terminated abruptly
Session Thread-2 closed.
IO Error/ Client Thread-0 terminated abruptly
Session Thread-0 closed.
BUILD STOPPED (total time: 4 minutes 39 seconds)
|
```

Figura 8: Estado do servidor momentaneamente

#### 4.2.2 Implementação

A implementação do servidor foi dividida em três principais partes:

1. **Server** Contém a main do cliente como também instruções de execução e inicia o servidor SSL através da instânciação da classe `SSLServer`
2. **SSLServer** Servidor que recebe os vários pedidos de clientes e lança um *thread* por cada pedido recebido
3. **CLientHandler Thread** que trata de cada ligação ao cliente



## 5 Cifragem de mensagens trocadas

### 5.1 Modos de cifra

Foi usado o modo CBC (Cipher Block Chaining) na cifra e decifra de todos os conteúdos e AES que é um algoritmo bastante seguro e rápido, tal como já referido em cima. Usa blocos e chaves com 128 bits ou mais (128 no caso do presente projeto) e é adaptado para processadores modernos, por exemplo, *smartcards* o que proporciona bons meios para a segunda parte deste projeto.

O modo de operação CBC funciona como explicado na figura abaixo e tem as seguintes características:

1. Modo que implementa uma cifra sequencial auto-sincronizável com uma cifra por blocos.
2. IV deve ser único por cada utilização (c.f. reutilização de chaves em cifras sequenciais). E.g. pode ser enviado em claro.
3. Um erro num bit do criptograma afecta o bit respectivo no bloco e todos do bloco seguinte.
4. Sequência de chave depende do IV, chave da cifra e de todo o texto limpo já cifrado.

# Cipher Block Chaining (CBC)

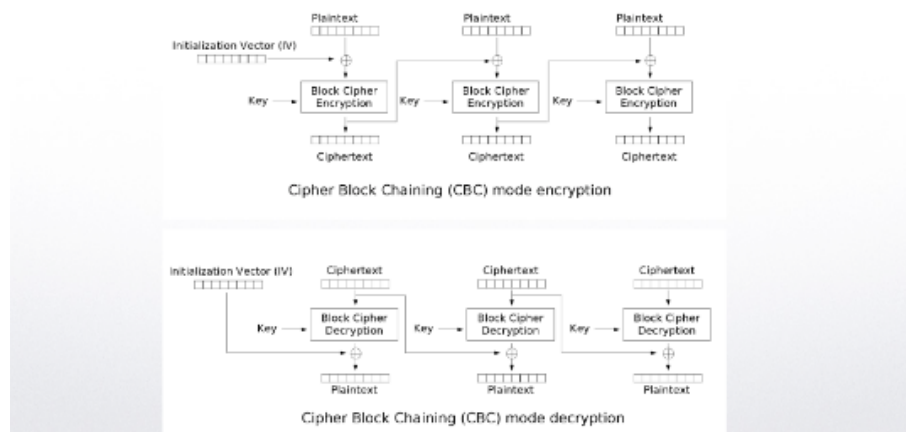


Figura 9: Modo de operação CBC

## 5.2 Chaves

### 1. User key:

É criada quando um utilizador se regista no servidor e fica no servidor;

Usada na encriptação de um ficheiro para esse utilizador em específico;

### 2. Device key:

É criada a partir de especificações (Serial Number ou UUID) do dispositivo em questão;

Gerada no player e no servidor através do envio dos SN e UUID;

### 3. Player key:

Gerada através de um Player ID que por sua vez é gerado aquando do download do player;

Confere validade do player em questão;

#### 4. File key:

Gerada quando ocorre um pedido de transferencia;

É apagada assim que é enviada/utilizada;

As Device e Player Keys são chaves que os dois componentes contêm. Quando o cliente faz o pedido ao servidor por um ficheiro, antes do envio do ficheiro cifrado com a File Key, é enviado um cabeçalho criptográfico que contém:

$$C-Pk-(D-Uk-(C-Dk-(File\ Key)))$$

em que **C** corresponde à operação de cifrar, **D** à operação de decifrar, **Pk** Player Key, **Uk** User Key e **Dk** Device Key.

Quando o cliente recebe o cabeçalho criptográfico, realiza as operações inversas às realizadas pelo servidor, descritas acima: o primeiro passo passa por decifrar usando a Pk. Seguidamente, pede a colaboração ao servidor, uma vez que não se encontra na presença da Uk, que procede à operação de C-Uk e envia o resultado novamente para o cliente. O cliente, por fim, procede a D-Dk e obtém a File Key que irá ser usada para decifrar o ficheiro.

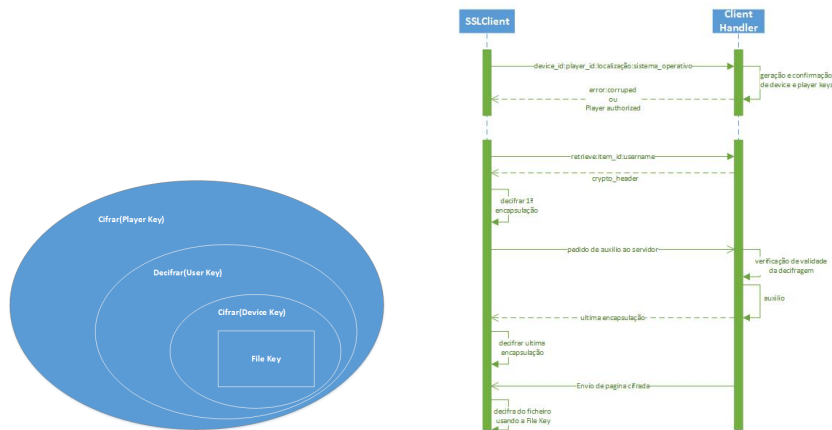


Figura 10: Criação do *Cryptographic Header* e Diagrama de Sequência de troca de chaves

## 6 Conclusões

Chegado ao final deste relatório, é nossa intenção efetuar uma retrospectiva da evolução do mesmo, tendo em conta os problemas com que nos deparámos, e principais conclusões retiradas.

Ao contrário do que idealizámos, foi necessário ter em conta vários outros aspetos, bem como alterar os métodos e modos de cifra, experimentado e desistindo de vários algoritmos até termos uma solução robusta preparada para ser publicada.

É de nossa ideia que o trabalho agora concluído e entregue fortaleceu bastante os nossos conhecimentos acerca do uso intensivo da criptografia para a segurança dos sistemas informáticos atuais.

Muito embora o trabalho seja entregue nesta primeira fase, temos consciência de que existem *bugs* a serem resolvidos de modo a construir uma aplicação mais completa na segunda fase.

## 7 Notas finais

1. **Execução do cliente** java IEDCS\_Player options
2. **Execução do servidor** java IEDCS\_Player options
3. **Bibliotecas externas** Foram usadas bibliotecas externas, nomeadamente commons-cli-1.3.1 para tratamento de parâmetros de entrada de consola e mysql-connector-java-5.0.8 para ligação ao servidor MySQL. As bibliotecas poderão ser encontradas em [https://code.ua.pt/svn/projeto\\_seguranca/dependencies](https://code.ua.pt/svn/projeto_seguranca/dependencies)
4. **Catálogo** Para exemplificação, estão disponíveis ebooks em formato .txt de variados conceituados autores. A biblioteca está disponível em

[https://code.ua.pt/svn/projeto\\_seguranca/ebooks](https://code.ua.pt/svn/projeto_seguranca/ebooks)

5. **Certificado** Uma vez que foi necessária a criação de um certificado para utilizar na ligação SSL/TLS, foi gerada uma KeyStore num certificado através do comando `$keytool -genkey -keystore myCliKeystore -keyalg RSA` [http://code.ua.pt/projects/projeto\\_seguranca/repository/raw/mySrvKeystore](http://code.ua.pt/projects/projeto_seguranca/repository/raw/mySrvKeystore)
6. **Base de Dados** Está disponível em `Dump.sql`
7. **Base64** Todos os dados em que era necessária a comunicação por arrays de bytes foram codificados em Base64 utilizando a biblioteca `java.util.Base64` disponível na versão JDK8 do Java
8. **Javadoc** Todo o código produzido contém documentação na forma de javadoc que pode ser gerada

## 8 Bibliografia

### Referências

- [1] Commons Cli. URL: <https://commons.apache.org/proper/commons-cli>
- [2] Get the hard disk serial number or Motherboard serial number. URL: <http://www.rgagnon.com/javadetails/java-0580.html>
- [3] How to get a unique computer identifier in Java. URL: <http://stackoverflow.com/questions/1986732/how-to-get-a-unique-computer-identifier-in-java-like-disk-id-or-motherboard-id>
- [4] SSL Client Authentication with smart card works in Java 6 but fails in Java 7. URL: <http://stackoverflow.com/questions/15503514/ssl-client-authentication-with-smart-card-works-in-java-6-but-fails-in-java-7>
- [5] NetworkInterface - Oracle. URL: <http://docs.oracle.com/javase/6/docs/api/java/net/NetworkInterface.html>
- [6] KeyStore <http://docs.oracle.com/javase/7/docs/api/java/security/KeyStore.html>
- [7] Signature <http://docs.oracle.com/javase/7/docs/api/java/security/Signature.html>
- [8] Java PKCS11 Reference Guide <https://docs.oracle.com/javase/7/docs/technotes/guides/security/p11guide.html>
- [9] JAVA + SSL Tutorial [http://stilius.net/java/java\\_ssl.php](http://stilius.net/java/java_ssl.php)
- [10] Sun SSL Socket Server : SSL Socket « Security « Java Tutorial [http://www.java2s.com/Tutorial/Java/0490\\_\\_Security/SunSSLSocketServer.htm](http://www.java2s.com/Tutorial/Java/0490__Security/SunSSLSocketServer.htm)

- [11] SSL Socket « Security « Java Tutorial [http://www.java2s.com/Tutorial/Java/0490\\_\\_Security/0860\\_\\_SSL-Socket.htm](http://www.java2s.com/Tutorial/Java/0490__Security/0860__SSL-Socket.htm)
- [12] Code Examples For Creating SSL Sockets <http://juliusdavies.ca/commons-ssl/ssl.html>
- [13] How to capture traffic for a specific program? <https://ask.wireshark.org/questions/3725/how-to-capture-traffic-for-a-specific-program>
- [14] Generating ID unique to a particular computer <http://stackoverflow.com/questions/3644722/generating-id-unique-to-a-particular-computer>
- [15] Generating unique IDs <http://www.javapractices.com/topic/TopicAction.do?Id=56>
- [16] Get a unique identifier <http://www.rgagnon.com/javadetails/java-0518.html>
- [17] How to get hard disk serial number using java <http://www.coderanch.com/t/532384/java/java/hard-disk-serial-number-java>
- [18] How to get DiskID <http://www.coderanch.com/t/476332/java/java/DiskID>
- [19] how to get hard disk serial number using java <http://stackoverflow.com/questions/5482947/how-to-get-hard-disk-serial-number-using-java>
- [20] Get motherboard serial Number of local system. <http://www.coderanch.com/t/426976/java/java/motherboard-serial-Number-local-system>

- [21] Get id of motherboard and hard disk in java for linux operating system? <http://stackoverflow.com/questions/21634510/get-id-of-motherboard-and-hard-disk-in-java-for-linux-operating-system>
- [22] SOL15292: Troubleshooting SSL/TLS handshake failures <https://support.f5.com/kb/en-us/solutions/public/15000/200/sol15292.html>
- [23] SSLServerSocket: getSupportedProtocols() : SSLServerSocket « javax.net.ssl « Java by API <http://www.java2s.com/Code/JavaAPI/javax.net.ssl/SSLServerSocketgetSupportedProtocols.htm>
- [24] GitHub - Hardware4Mac <https://github.com/sarxos/secure-tokens/blob/master/src/main/java/com/github/sarxos/securetoken/impl/Hardware4Mac.java>
- [25] GitHub - Hardware4Win <https://github.com/sarxos/secure-tokens/blob/master/src/main/java/com/github/sarxos/securetoken/impl/Hardware4Win.java>
- [26] GitHub - Hardware4Nix <https://github.com/sarxos/secure-tokens/blob/master/src/main/java/com/github/sarxos/securetoken/impl/Hardware4Nix.java>
- [27] PBKDF2 (Password-Based Key Derivation Function 2) <https://en.wikipedia.org/wiki/PBKDF2>
- [28] Java – find location using Ip Address <http://www.mkyong.com/java/java-find-location-using-ip-address/>
- [29] MySQL ON DELETE CASCADE Deletes Data From Multiple Tables <http://www.mysqltutorial.org/mysql-on-delete-cascade/>