universidade de aveiro

DETI | ua

Relatório Projeto

Segurança 2017-2018 1ºS

Professores: André Zúquete, andre.zuquete@ua.pt
João Barraca, jpbarraca@ua.pt

Alunos: André Cardoso, 65069 marquescadroso@ua.pt
Nelson Costa, 42983 nelson.costa@ua.pt

Grupo 02 - P2 31-12-2017

Este relatório tem como objetivo apresentar o contexto, tecnologias, funcionalidades implementadas e referências usadas no desenvolvimento do projeto proposto na unidade curricular Segurança.

1. Contexto:

Este projeto descreve um sistema de mensagens, onde os utilizadores podem comunicar entre si, enviando mensagens diretamente para um servidor ou repositório central que as guarda. Os utilizadores só podem comunicar com o servidor, ou seja, toda a informação deverá obrigatoriamente passar por este antes da mesma chegar ao destinatário.

O sistema de mensagens deverá ser implementada de forma a suportar a confidencialidade, a integridade e autenticação das mensagens e utilizadores, assim como a respetiva confirmação da entrega das mensagens.

2. Introdução:

O sistema de mensagens é composto por dois componentes principais: os **clientes**, que representam os utilizadores do sistema, e o **servidor**, que serve de ponto de encontro para os clientes se interligarem. Como os clientes não interagem diretamente, o servidor dispõe de uma conexão por *socket*, através do qual os clientes se podem ligar e trocar mensagens JSON com outros clientes. O servidor também armazena internamente alguns dados sobre os utilizadores, por exemplo, os identificadores e caixas de mensagens associados, e outras informações criptográficas para a interação com os mesmos. A Figura 1 mostra uma vista geral do sistema de mensagens.

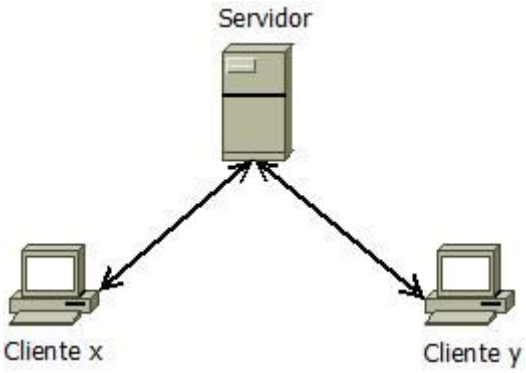

Figura 1 - Vista geral do sistema de mensagens com o servidor e os clientes.

Estes comunicam diretamente com o servidor para enviar e/ou receber as mensagens de outros clientes.

3. Arquitecturas

Este capítulo inclui os modelos de dados utilizados para os componentes do sistema de mensagens referidos anteriormente. As Figuras 2 e 3 ilustram os modelos de dados do servidor e cliente, respectivamente.

3.1 Servidor

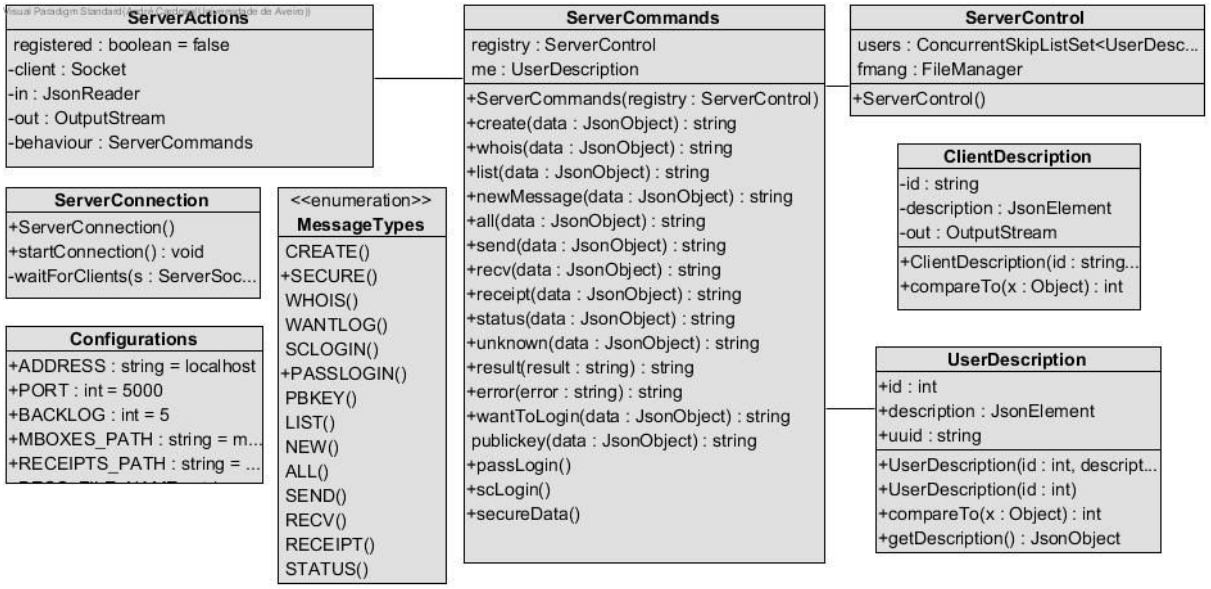

Figura 2 - Modelo de dados do servidor

Algumas alterações foram efectuadas no servidor, principalmente para numa primeira fase para explorar e avaliar o estado em que se encontrava. Mudamos as mensagens de tipo para enumerados adicionamos um nova classe “*ServerComands*” de maneira a facilitar a escalabilidade e compressão do código.

3.2 Cliente

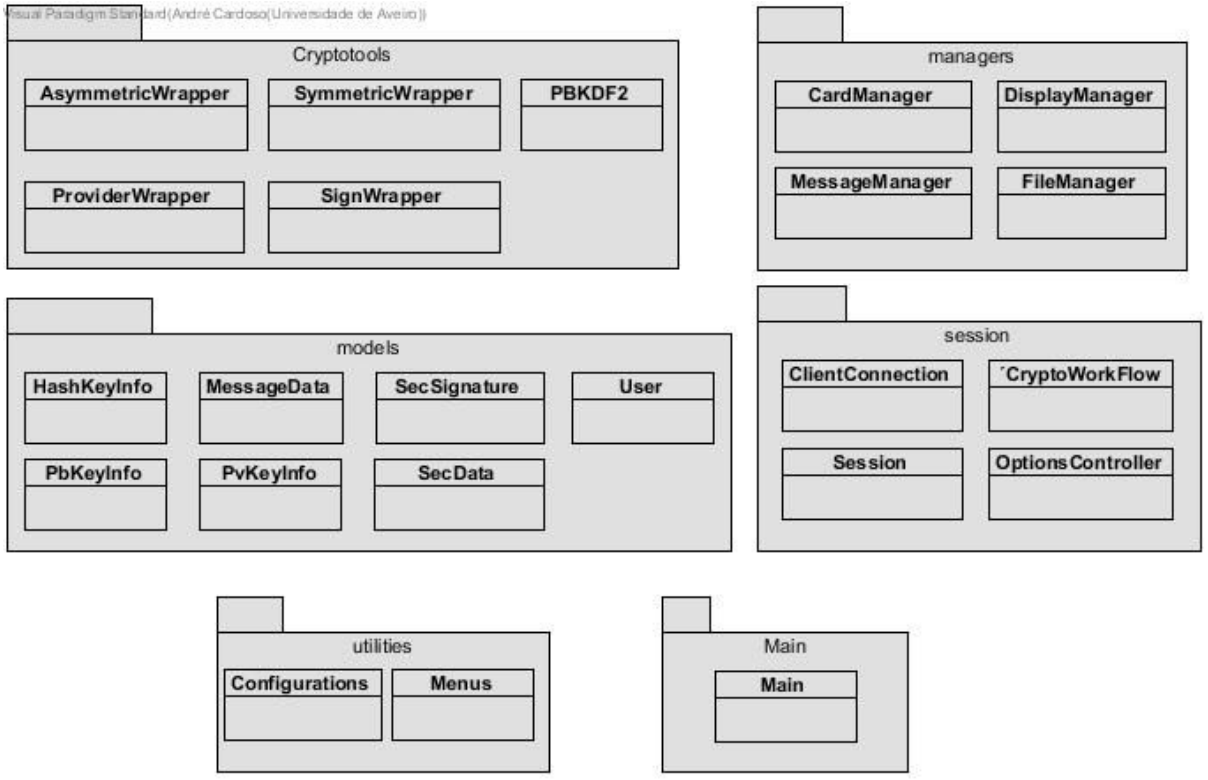

Figura 3 - Modelo de dados do cliente

Projecto encontra-se organizado em vários packages, segue-se uma descrição sobre eles:

- "models": encontram-se as várias classes que que contêm informação que o utilizador necessita para efectuar as várias ações;
- "managers": contém contem as classes responsáveis por lidar com a comunicação com o leitor de cartões, ficheiros e formatar a informação para apresentar ao utilizador.;
- "session": contém as classes responsáveis para fazer a ligação com o servidor, interpretar e executar a ordem do utilizador e executar os fluxos, descritos no capítulo 5, de manipulação de dados;
- "cryptotools": contém as operações criptográficas necessárias para este projecto. Sendo elas as operações de cifras assimétricas, simétricas, assinaturas, ligação ao "provider" do Cartão de Cidadão e a leituras dos respectivos certificados e o uso de password derivadas de funções, "Password Based Key Derivation Function 2" (PBKDF2);
- "utilites": contém as configurações, isto é, valores fixos para o decorrer do programa;

3.3 Modelo de Dados do Utilizador

O utilizador possui um conjunto de dados necessários para efetuar as mais variadas operações pretendidas com as devidas validações. Modelos de dados seguinte apresenta esse conjunto de dados.

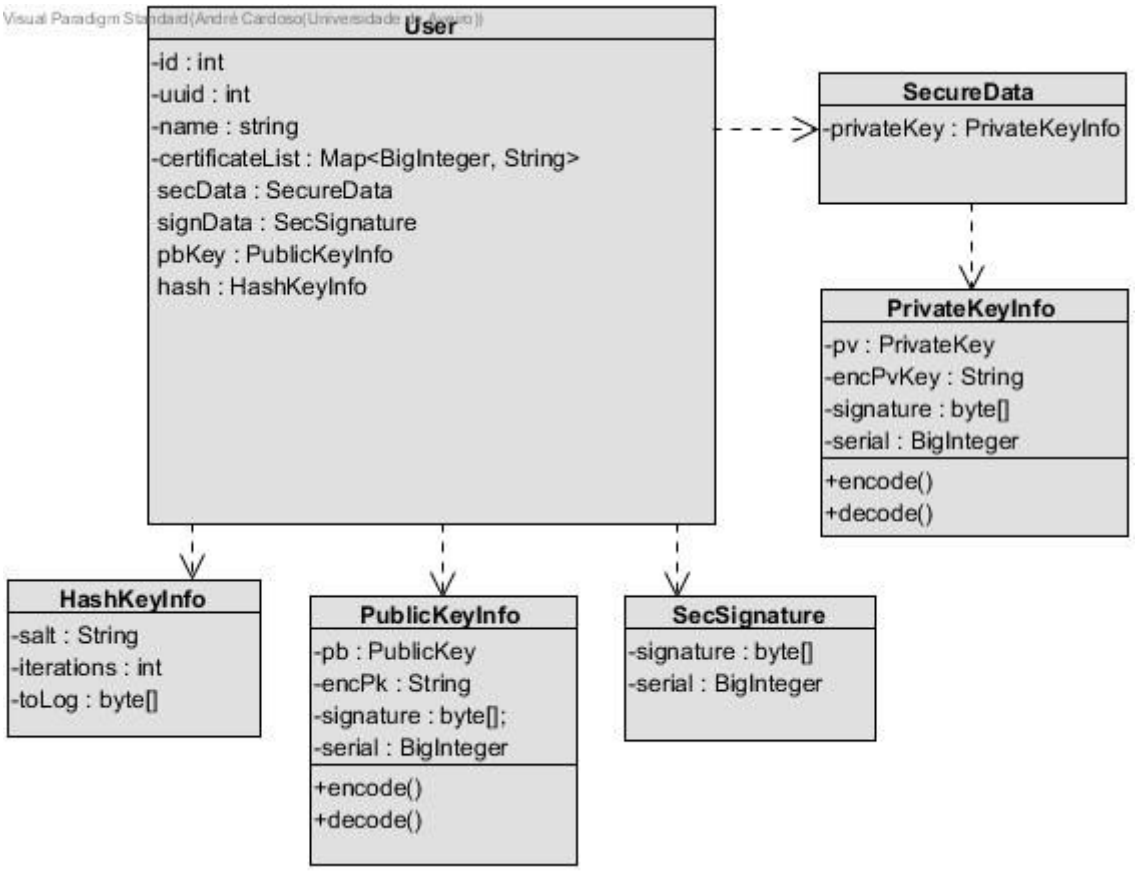

Figura 4. Diagrama de Classe da informação do utilizador.

- Id: identificador interno no servidor;
- uuid: identificador único universal, utilizamos o número do cartão de cidadão;
- CertificateList: lista de certificados que existe no cartão de cidadão;
- SecureData: estrutura de dados contém toda a informação que é privada para o utilizador. Tal como chave privada, Diffie-Hellman value, … Este objecto é serializado e cifrado com a chave gerada através da password derivada de um chave, PBKDF2.
- SecureSignature: estrutura de dados que contém a assinatura e o número de série do certificado usado.
- PublicKeyInfo: estrutura de dados que contém a informação pública do utilizador. Neste caso é a chave RSA pública, respectiva assinatura e o número de série do certificado usado.
- HashKeyInfo: quando é criado uma conta, é necessário gerar um chave PBKDF2, para tal é preciso o valor Salt e número de iterações. Esta estrutura de dados guarda

estes valores. Aqui também é guardado o ID do utilizador no servidor cifrado com com essa mesma chave.

4. Fluxo das Ações

Este capítulo inclui as funcionalidades de segurança implementadas nos componentes (servidor e cliente) do sistema de mensagens.

4.1 Criação de Conta

A criação de uma conta no sistema só pode ser efetuada usando o Cartão de Cidadão (CC), ou seja, o utilizador deve inserir o seu CC num leitor de cartões antes de criar uma conta. A Figura 5 mostra o diagrama de sequências do processo de criação de conta entre o utilizador e servidor.

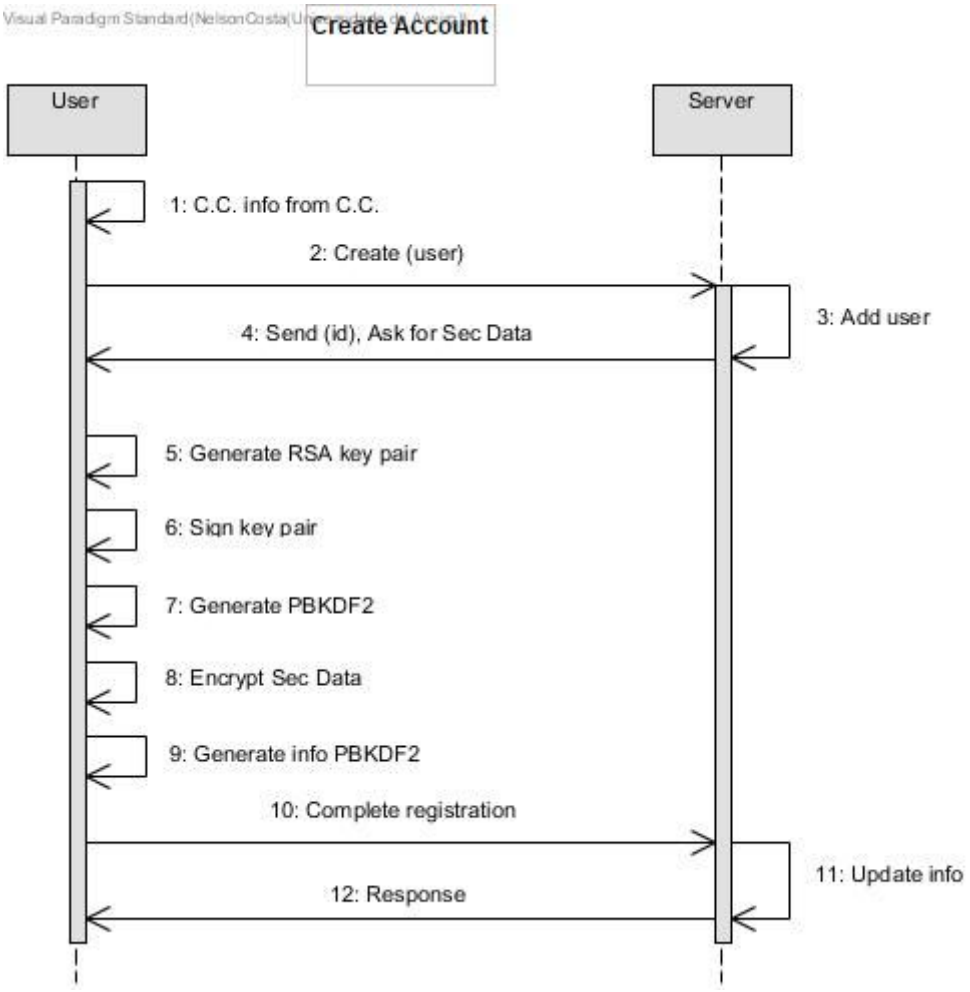

Figura 5 - Processo de criação de conta

Como mostra a Figura 5, o utilizador deve inicialmente introduzir o seu CC num leitor de cartões. A seguir, é efetuada a leitura e extração dos dados relativos ao Certificado de autenticação contidos no CC, por exemplo, o nome e número identificação civil que irá servir de identificador único (UUID) para o utilizador.

Após a extração da informação do utilizador, é enviada uma mensagem CREATE para o servidor que processa os dados recebidos, cria uma nova conta e gera um identificador interno (ID) para o cliente que lhe é depois devolvido. Depois desta informação gravada no servidor é pedido ao cliente informação relativa aos processos criptográficos que se irão principalmente no envio e receção de mensagens.

O cliente gera um par de chaves assimétricas RSA e assina as chaves pública e privada com o seu certificado de autenticação. Depois, é gerado um *hash* PBKDF2 da palavra-passe previamente pedida ao cliente e que irá servir como chave de sessão. Esta chave irá ser usada juntamente com o algoritmo simétrico AES para cifrar uma estrutura de dados serializada que contém toda a informação privada do utilizador, assim como irá cifrar o ID do utilizador, caso este queira fazer login sem cartão de cidadão. A cifra desta estrutura de dados é ela também assinada.

Toda esta informação é enviada através da mensagem do tipo “SecData”e é guardada no servidor.

4.2 Login

O login do utilizador só pode ser efetuado se este tiver uma conta previamente criada. As Figuras 6 e 7 mostram as duas formas possíveis para o cliente se credenciar usando C.C.

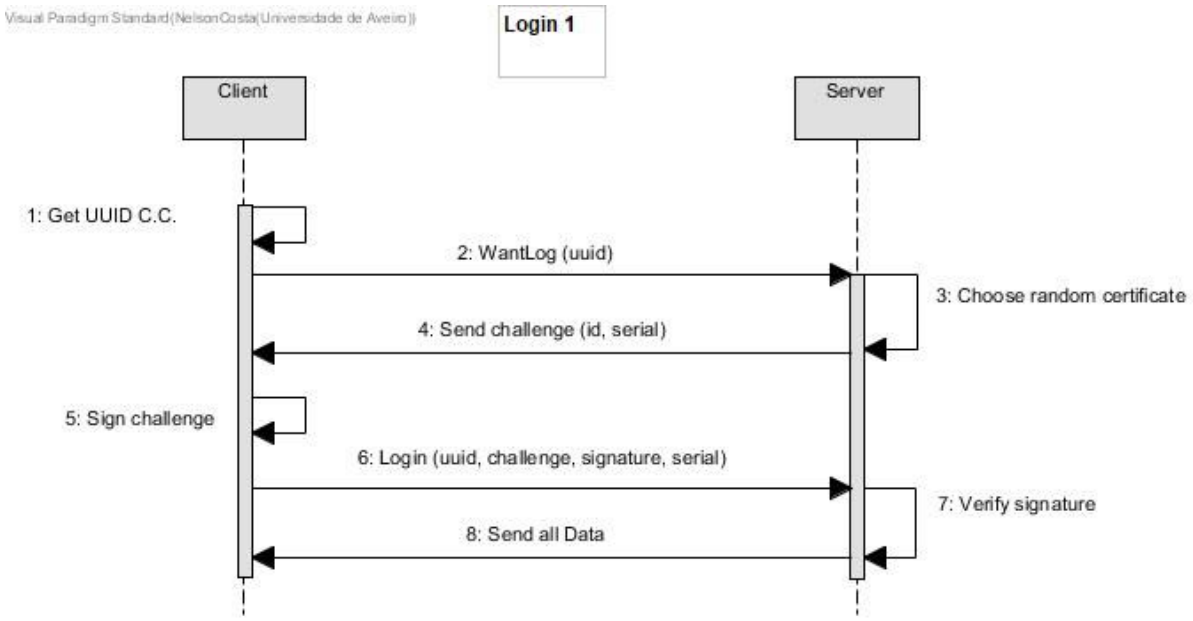

Figura 6 - 1º Processo de login

Na Figura 6, o cliente introduz o seu CC num leitor de cartões e envia uma mensagem “WANTLOG” com o seu UUID para o servidor. Este gera um desafio que é depois enviado para o cliente como resposta ao seu pedido. O cliente recebe o desafio e assina-o com o seu certificado de autenticação. A seguir, o cliente pede login ao servidor enviando-lhe uma mensagem LOGIN com o seu UUID, o desafio, o desafio assinado e o

número série do seu certificado de autenticação. O servidor recebe o pedido de login e verifica a assinatura do certificado de autenticação do cliente. Se a assinatura for válida, o servidor responde ao cliente notificando-o de que o login foi bem sucedido.

4.3 Enviar Mensagem

Para enviar uma mensagem, o cliente tem de estar obrigatoriamente credenciado no sistema. O envio de uma mensagem é efetuado através das mensagens do tipo “send”. A Figura 8 mostra o diagrama de sequências para o processo de envio deste tipo de mensagens.

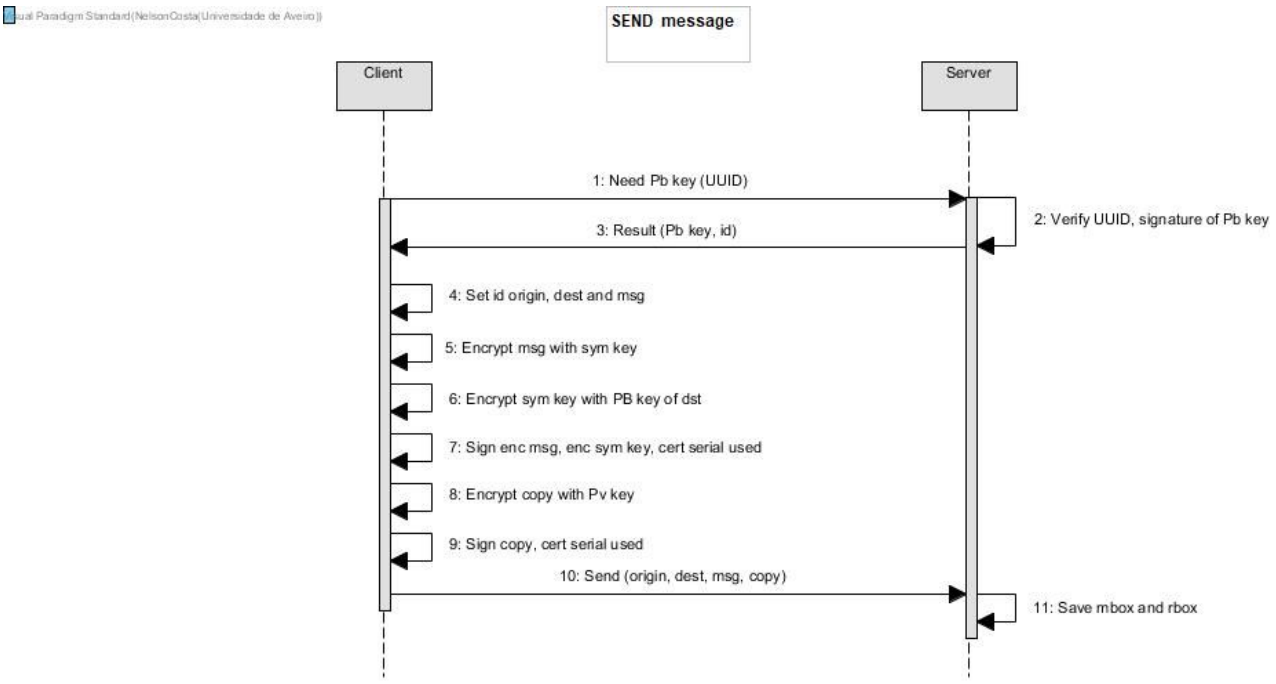

Figura 8 - Processo de envio de mensagens do tipo “send”

Na Figura 8, é pedido ao cliente o UUID do destinatário e a mensagem a enviar. A seguir, o cliente envia para o servidor uma mensagem com o UUID para obter a chave pública do destinatário. O servidor recebe a mensagem, verifica a existência do UUID e a assinatura da chave pública do destinatário. Esta chave é depois enviada ao cliente juntamente com o ID interno do destinatário. A seguir, o cliente cifra a mensagem a enviar com a chave pública do destinatário, a cópia da mesma com a sua chave privada e envia uma mensagem SEND para o servidor juntamente com o seu ID interno e ID interno do destinatário. O servidor recebe a mensagem SEND e guarda a mensagem do cliente na *mbox* do destinatário e a cópia dessa mensagem na *rbox* do destinatário.

4.5 Receber Mensagem

Para receber uma mensagem, o cliente tem de estar obrigatoriamente credenciado no sistema e enviar para o servidor mensagens do tipo “recv”. Neste tipo de mensagens também se incluem as mensagens do tipo “receipt”.

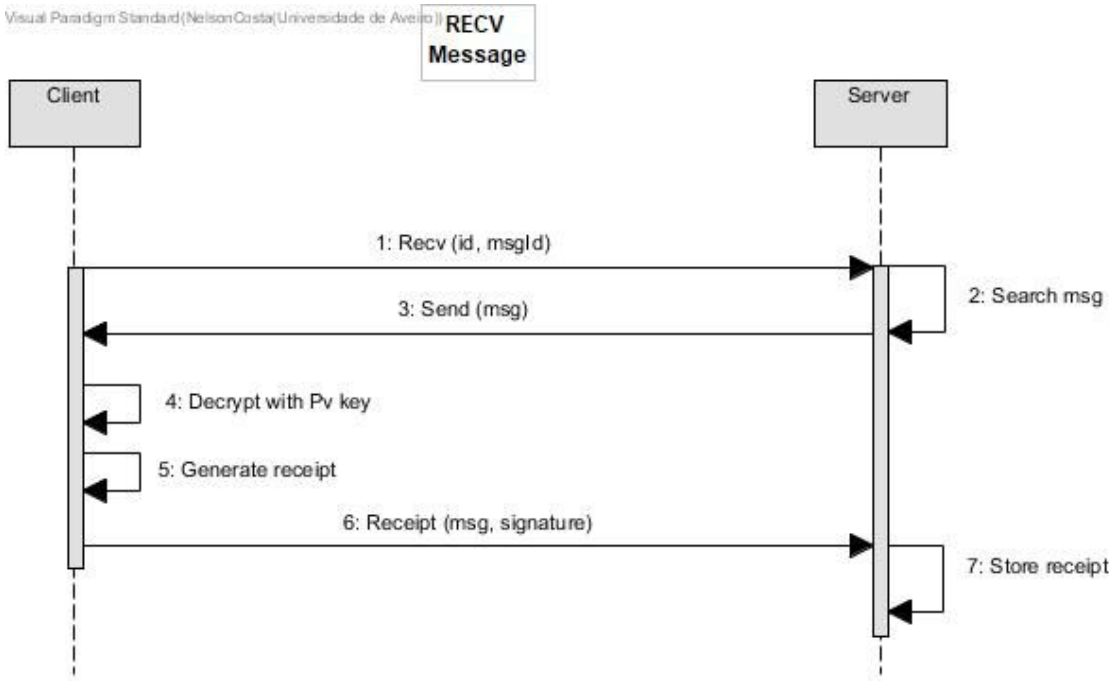

Figura 9 - Processo de envio de mensagens do tipo “recv”

Na Figura 9, é inicialmente pedido ao cliente o identificador da mensagem que pretende visualizar. A seguir, o cliente envia para o servidor uma mensagem do tipo “recv” com o seu ID interno e o identificador da mensagem. Como resposta ao pedido do cliente, o servidor envia para o cliente a mensagem e o ID interno do seu remetente. A seguir, o cliente decifra a mensagem com a sua chave privada para depois conseguir visualizá-la.

O envio das mensagens do tipo “receipt” tem como objetivo criar um recibo logo após a decifra e visualização da mensagem. Esta mensagem tem como objetivo fazer um registo sempre que um utilizador acede a uma mensagem, ou seja, uma mensagem pode vir a ter vários recibos. Estes recibos contêm a informação do identificador do utilizador que visualizou a mensagem, do identificador da mensagem, texto da mensagem assinado pelo recetor e a hora de acesso.

4.6 Outras Ações

Para o envio das mensagens do tipo ”list”, o utilizador tem de estar obrigatoriamente credenciado no sistema. Este tipo de mensagem tem como objetivo listar todos os clientes com *message boxes* no servidor.

```
============== Users List ====================
|  UUID: 134029844 - NELSON FILIPE DOS SANTOS COSTA
|  UUID: 80697232 - REGINA MARIA DA COSTA MARQUES CARDOSO
|  UUID: 141423072 - ANDR? FILIPE MARQUES CARDOSO
==============================================
```

Figura 10 - Resposta do servidor à mensagem do tipo “list”

Na Figura 10, o cliente “NELSON FILIPE DOS SANTOS COSTA” enviou uma mensagem do tipo “list” para o servidor que respondeu com a lista de todos os clientes e respetivos UUIDs do sistema. De notar que é também listado o nome e UUID do remetente da mensagem, neste caso, do cliente “NELSON FILIPE DOS SANTOS COSTA”.

Para o envio das mensagens do tipo ”new”, o utilizador tem de estar obrigatoriamente credenciado no sistema. Este tipo de mensagem tem como objetivo listar todas as novas mensagens do cliente.

```
============== New Messages ==================
|    3_1
|    3_2
==============================================
```

Figura 11 - Resposta do servidor à mensagem do tipo “new”

Na Figura 11, o cliente que enviou a mensagem do tipo “new” tem 2 novas mensagens no seu *mbox*. O primeiro dígito da mensagem corresponde ao ID interno do remetente e o segundo dígito corresponde a ordem das mensagens do mesmo remetente, começando em 1. Neste caso, o cliente com ID interno 3 enviou duas novas mensagens para o cliente que enviou a mensagem “new”.

Para o envio das mensagens do tipo ”all”, o utilizador tem de estar obrigatoriamente credenciado no sistema. Este tipo de mensagem tem como objetivo listar todas as mensagens no *mbox* do cliente.

```
============== Message Box ===================
=Received:
        3_1
        3_2
=Sended:
        3_1
        3_2
==============================
```

Figura 12 - Resposta do servidor à mensagem do tipo “all”

Na Figura 12, o cliente que enviou a mensagem do tipo “all” tem neste momento duas mensagens enviadas e outras duas recebidas. Neste caso, o cliente com ID interno 3 enviou duas mensagens para si próprio.

Para o envio das mensagens do tipo ”status”, o utilizador tem de estar obrigatoriamente credenciado no sistema. Este tipo de mensagem tem como objetivo verificar o estado das mensagens enviadas e se estas têm recibos associados ou não.

Inicialmente, é pedido ao cliente o identificador da mensagem que pretende verificar. Depois, o cliente envia para o servidor uma mensagem do tipo “status” com o seu ID interno e o identificador da mensagem. O servidor, por sua vez, procura a mensagem e envia para o cliente os respetivos recibos, se existirem.

5. Conclusão

Com este trabalho foi possível estudar alguns dos conceitos e mecanismos de segurança úteis para que um sistema de troca de mensagens assíncrona onde a segurança da comunicação entre os seus componentes, i.e., o servidor e os múltiplos clientes, devia ser assegurada. O planeamento deste trabalho foi relativamente simples de elaborar e projetar em papel, mas encontramos maiores dificuldades na implementação do código, i.e., pôr em prática todos os pontos especificados no planeamento visto que nem tudo estava nos guiões práticos e slides teóricos da disciplina Segurança. Apesar disso, foi possível implementar a maioria dos conceitos e assim atingir o principal objetivo deste trabalho.

6. Referências

Escolha de algoritmos:
https://stackoverflow.com/questions/5554526/comparison-of-des-triple-des-aes-blowfish-encryption-for-data

Gson:
http://www.baeldung.com/jackson-vs-gson
http://www.baeldung.com/gson-deserialization-guide
https://www.javadoc.io/doc/com.google.code.gson/gson/2.8.2

Implementação do Log:
https://howtodoinjava.com/log4j2/

Usb events:
https://www.programcreek.com/java-api-examples/index.php?api=javax.usb.event.UsbServicesEvent

Smartcard:
https://www.javaworld.com/article/2077101/learn-java/smart-cards--a-primer.html?page=2

Exemplo do uso de smartcard e certificados:
https://www.programcreek.com/java-api-examples/index.php?source_dir=virtual-pki-card-master/se-pki-client/src/org/nick/sepkiclient/Main.java

Encoded:
http://xacmlinfo.org/2013/11/30/how-to-certificate-retrieve-x509-certificate-as-data/

AES Example:
https://howtodoinjava.com/security/java-aes-encryption-example/

Doc. Oracle:
https://docs.oracle.com/javase/8/docs/technotes/guides/security/p11guide.html
https://docs.oracle.com/javase/7/docs/jre/api/security/smartcardio/spec/javax/smartcardio/CardTerminal.html
https://docs.oracle.com/javase/8/docs/technotes/guides/security/certpath/CertPathProgGuide.html

CRL List:
http://www.nakov.com/blog/2009/12/01/x509-certificate-validation-in-java-build-and-verify-chain-and-verify-clr-with-bouncy-castle/

TLS/SSl:
http://blog.palominolabs.com/2011/10/18/java-2-way-tlsssl-client-certificates-and-pkcs12-vs-jks-keystores/index.html