Faculdade de Ciências da Universidade de Lisboa
Departamento de Informática
Disciplina de Segurança e Confiabilidade
Ano letivo de 2020/2021

# RELATÓRIO DO PROJETO "SEITCHIZ"

## RESUMO DO PROJETO

*"A construção de uma aplicação distribuída a ser executada numa sandbox, focando-se essencialmente nas funcionalidades da aplicação. O trabalho consistiu na concretização do sistema SeiTchiz (uma versão portuguesa do Instagram), que é um sistema do tipo cliente-servidor que permite que os utilizadores (clientes) utilizem um servidor central para partilhar fotografias e comunicar com outros utilizadores. O sistema suporta dois modos de funcionamento. No modo mural (feed), os utilizadores colocam fotografias no seu perfil público guardado no servidor e podem seguir quaisquer outros utilizadores, vendo no seu mural as fotografias que estes publicaram. O serviço também permite a colocação de likes nestas fotografias. No modo conversação, podem enviar mensagens para grupos privados de utilizadores e ler as mensagens enviadas para os grupos a que pertencem. (…)"*

## COMO COMPILAR E CORRER O PROJECTO

(ESTE PROJETO FOI TESTADO EM AMBIENTE LINUX E MACOS)

- 45678 é o Porto usado para a conexão
- Todas as keystores têm como password: 456789
- Existem 3 utilizadores criados (João, Rafael, Ana) com as suas respetivas keystores (JoãoStore, RafaelStore, AnaStore) e certificados.
- Os testes foram feitos com estes 3 utilizadores.

  É possível fazer a ligação com o RafaelStore e Rafael, com o JoãoStore e João e com AnaStore e Ana, substituindo nos parâmetros do cliente na keystore e userID respetivamente.

  O Servidor usa como parâmetros: <porto> <keystore> <keystore-password>
  O Cliente usa como parâmetros: <ip:porto> <truststore> <keystore> <keystore-password> <clientID>

**COM POLÍTICAS E COM .JAR:**

Para correr o servidor deve colocar-se na pasta SegC-grupo49-proj1-2 e correr o seguinte comando:

```
java -cp bin -Djava.security.manager -Djava.security.policy=server.policy -jar server.jar 45678 servidorStore 456789
```

Para correr o cliente deve colocar-se na pasta SegC-grupo49-proj1-2 e correr o seguinte comando: (Entrando com o Rafael)

```
java -cp bin -Djava.security.manager -Djava.security.policy=client.policy -jar client.jar 127.0.0.1:45678  trustore.cliente RafaelStore 456789 Rafael
```

**SEM POLÍTICAS E SEM .JAR:**

Para correr o servidor deve colocar-se na pasta SegC-grupo49-proj1-2 e correr o seguinte comando:

java -cp bin Server.SeiTchizServer 45678 servidorStore 456789

Para correr o cliente deve colocar-se na pasta SegC-grupo49-proj1-2 e correr o seguinte comando: (Entrando com o Rafael)

java -cp bin Client.SeiTchiz 127.0.0.1:45678 trustore.cliente RafaelStore 456789 Rafael

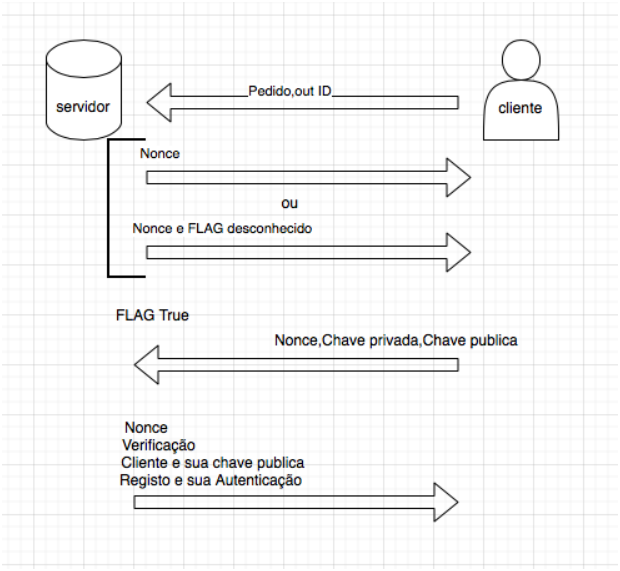
**NO ECLIPSE E COM POLÍTICAS:**

**1º** Importar a pasta do projeto
**2º** Ir a classe .java do servidor a SeiTchizServer e fazer: run as -> Run configuration -> Arguments -> 45678 servidorStore 456789 nos Argumentos e na VM -> -Djava.security.manager
-Djava.security.policy=server.policy
**3º** Fazer o mesmo na classe .java do cliente a SeiTchiz e fazer: run as
-> Run configuration -> Arguments -> 127.0.0.1:45678 truststore.client RafaelStore 456789 Rafael nos Argumentos e na VM -> -Djava.security.manager
-Djava.security.policy=client.policy

## SOLUÇÕES IMPLEMENTADAS

De forma a conseguirmos criar uma solução que atendesse a todos os pontos que nos eram pedidos nesta fase do trabalho desenvolvemos múltiplas estratégias:

- Para tentar compreender a autenticação desta fase do trabalho, foi concebido um desenho como esboço na qual depois esta foi implementada.




- Foi criada uma pasta auxiliar para esta fase, a SeiTchizKeys, onde contém vários métodos necessários para o desenvolvimento desta fase do projeto. A classe KeyStoreAux guarda a chave publica e privada a partir da Keystore. A classe Keys trata de resolver a encriptação e decriptação. A classe KeyServer trata de ir buscar os certificados do cliente e guardá-los. A classe KeyClient

trata de ir buscar um Certificado e devolver a sua chave e trata também de gerar uma chave simétrica.

- Ao ser gerado um novo grupo, é criado uma pasta GroupsKeys onde irá conter um .txt com o identificador e a chave desse grupo.

- Ao adicionar um elemento a um grupo, como é explicado na secção: "Limitações do Trabalho", em baixo, foi partido do suposto que o utilizador a adicionar já existe, é então adicionado ao grupo, criando uma pasta de Users desse grupo onde contém o .txt desse utilizador adicionado contendo o seu identificador e a chave e um .txt onde contem os membros desse grupo, juntamente com o caminho para o ficheiro desse utilizador onde tem a chave.

- Ao remover um elemento a um grupo, como é explicado na secção: "Limitações do Trabalho", em baixo, foi partido do suposto que utilizador a remover já existe. Ao remover, este apaga da lista de membros do grupo, e do grupo.

- De forma a conseguir uma forma do utilizador poder utilizar o comando "post" como é pedido no trabalho, deixamos uma pasta (postDirectory) que é criada no momento da compilação do servidor. Essa pasta tem uma imagem de exemplo chamada "teste.png" e todas as imagens que se queira publicar no servidor, deverão ser colocadas nessa pasta primeiro.

## LIMITAÇÕES DO TRABALHO

- Ao adicionar um membro ao grupo (comando "addu"), partimos do suposto que este membro já existia (já autenticado, estando no ficheiro users.txt, alteramos a linha onde se vê que o utilizador existe na autenticação para "true", na primeira do primeiro trabalho essa linha funcionava corretamente, mas nesta versão não conseguimos entender a razão então colocamos essa variável a "true" de forma a conseguir implementar este método, a mesma situação acontece com o "removeu".

- No método post, é enviado um ficheiro/foto para o servidor, a foto a enviar deve ser colocada/criada, na pasta postDirectory (criada depois de se inicializar a conexão). Ao chamar o método post, tem que se chamar a foto/ inserir a foto com a sua extensão (por exemplo, se criar a foto "teste.png", tem que chamar post com teste.png (p teste.png)).

- No trabalho enviado, vai a pasta postDirectory já criada, contendo assim uma foto como teste(teste.png), caso queira testar desta forma. Caso queira apagar o postDirectory e esta foto, terá na mesma de depois de criada a conexão entre o cliente e o servidor, ele cria então a pasta postDirectory mas terá então que criar uma foto como teste, como mencionado no ponto anterior.

- O método msg não foi implementado nesta fase, contendo as funcionalidades(implementadas) da fase anterior.

Realizado por:
**Grupo 49**
João Miranda nº 47143
Manuel Tovar nº 49522
Manuel Lopes nº 49023