

Segurança e Confiabilidade

2016/2017

**Relatório e Discussão – Etapa 2**

Grupo 005

Autores

Francisco João Guimarães Coimbra de Almeida Araújo, n.º 45701

João Marques de Barros Mendes Leal, n.º 46394

Joana Correia Magalhães Sousa, n.º 47084

Nota Introdutória:

Todos os objetivos foram compridos com sucesso.

Problemas encontrados

Durante a realização do projeto as partes onde surgiram mais problemas foram no checkMac, na cifra e decifra com password. Outra situação onde tivemos problemas foi no início do projeto, ao tentar perceber onde é que as funcionalidades de segurança tinham de ser implementadas. Visto o nosso projeto estar dividido em várias classes diferentes acabámos por optar pela criação de uma classe SecurityHandler que implementa todos os métodos necessárias para o cumprimento dos objetivos. Em seguida esses métodos são invocádos pelas classes que necessitam das funções por eles implementadas.

Segurança da aplicação criada

De modo a autenticar os utilizadores, o servidor guarda as passwords de cada utilizador num ficheiro cifrado com a password introduzida quando o servidor arranca. Este ficheiro tem a sua integridade assegurada recorrendo a um MAC. De modo a garantir a autencidade dos intervenientes, o servidor envia um nonce ao cliente, que este vai concatenar a uma sintese da password do utilizador e enviar de volta ao servidor. O servidor prossegue a comparar essa sintese com uma que ele calcula recorrendo ao ficheiro das passwords.

A integridade dos repositórios é assegurada recorrendo ao algoritmo HmacSHA256, tal como o ficheiro das passwords e os ficheiros das permissões.

Para garantir a autenticidade do servidor e a confidencialidade da comunicação, recorremos a canais de comunicação seguros, TLS/SSL. Para o uso destes canais de segurança, configurámos as chaves de cliente e do servidor e o certificado deste último. Para esse efeito existe uma truststore com o certificado do servidor e uma keystore com a chave privada do servidor.

Com o objetivo de impedir a personificação do utilizador ou que o conteúdo dos repositórios seja observado indevidamente, utilizámos criptografia hibrida, ie, usámos chaves simétricas e assimétricas e assinaturas digitais. Após gerar a assinatura digitial do ficheiro que pretende que o servidor guarde ele envia-a e esta é guardada pelo servidor num ficheiro com extensão .sig. Em seguida o cliente gera uma chave simétrica recorrendo ao algoritmo AES, cifra-a recorrendo à chave pública do servidor e envia-a ao servidor. O servidor decifra a chave simétrica recorrendo à sua chave privada. Agora que ambos partilham a chave simétrica o cliente envia o ficheiro cifrado com a chave simétrica. O servidor também vai guardar a chave simetrica que partilha com o cliente no repositorio do mesmo. Para decifrar o ficheiro, o servidor realiza o processo anteriormente descrito no sentido inverso. Primeiro vai buscar a chave simétrica cifrada ao ficheiro adequado e decifra-a recorrendo à sua chave privada. Em seguida o servidor envia o conteúdo ficheiro pedido cifrado juntamente com a chave simétrica que decifrou. O cliente procede a decifrar o conteúdo ficheiro com a chave simétrica e compara o resultado com a assinatura do ficheiro que recebeu do servidor.

O acesso que é dado a um utilizador sobre um repositório e os ficheiros que lá estão armazenados e as operações necessárias, para que após esse ser retirado o utilizador não consiga aceder a nenhum dos ficheiros desse repositório, já tinham sido introduzidas no projeto anterior.

Código Fonte:

ClientServerHandler:

/\*\*Grupo sc005

\* Francisco JoÃ£o GuimarÃ£es Coimbra de Almeida AraÃºjo nÂº45701

\* Joana Correia MagalhÃ£es Sousa nÂº47084

\* JoÃ£o Marques de Barros Mendes Leal nÂº46394

\*/

**import** java.io.File;

**import** java.io.FileFilter;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.security.PrivateKey;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**import** javax.crypto.SecretKey;

/\*\*

\* A ideia desta classe eh tratar das interaccoes do myGit com o myGitServer

\* **@author** Utilizador

\*

\*/

**public** **class** ClientServerHandler {

**private** **static** **final** **int** ***PUSH\_REP*** = 20;

**private** **static** **final** **int** ***PUSH\_FILE*** = 30;

**private** **static** **final** **int** ***PULL*** = 40;

**private** **static** **final** **int** ***SHARE*** = 50;

**private** **static** **final** **int** ***REMOVE*** = 60;

**private** String username;

**private** String passwd;

**private** Messager msg;

/\*\*

\* Construtor de ClientServerHandler

\* **@param** username - nome do cliente

\* **@param** passwd - password do cliente

\*/

**public** ClientServerHandler(String username,String passwd){

**this**.username = username;

**this**.passwd = passwd;

msg = **new** Messager();

}

/\*\*

\* Envia para o servidor os dados relativamente sobre o cliente em questao

\* **@param** outStream - objeto por onde escreve ao servidor

\*/

**public** **void** sendInitInfo(ObjectOutputStream outStream){

**try** {

outStream.writeObject(username);

} **catch** (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Metodo que cria um novo utilizador caso este nao exista

\* **@param** outStream - objeto por onde escreve ao servidor

\* **@param** inStream - objeto por onde le ao servidor

\* **@param** sc - security handler, trata de toda a seguranca envolvida para este caso

\*/

**public** **int** addUser(ObjectOutputStream outStream, ObjectInputStream inStream,SecurityHandler sc){

**try** {

**short** resp;

outStream.flush();

**if**((resp = inStream.readShort()) == 1){ //se nao existir

System.***out***.println("--O utilizador " + username + "vai ser criado");

System.***out***.println("Confirmar password do utilizador " + username);

Scanner reader = **new** Scanner(System.***in***);

String comp = reader.nextLine();

reader.close();

**if**( passwd.equals(comp)){

msg.confirm(outStream);

//enviamos a password normalmente

outStream.writeObject(passwd);

outStream.flush();

**if**(inStream.readShort() == 1){ //ler

System.***out***.println("--O utilizador " +username+ " foi"

+ " criado com sucesso");

**return** 1;

}

**else**{

System.***out***.println("--Ouve um erro na criacao"

+ " do utilizador");

**return** -1;

}

}

**else**{

msg.reject(outStream);

System.***out***.println("Erro:Ocorreu um erro a criar o utilizador");

**return** -1;

}

}

**else** **if** (resp == -1){ //lemos o reject

//aqui recebemos o nonce

String nonce = (String) inStream.readObject(); //lemos uma string

System.***out***.println("Nonce = " + nonce);

//enviamos a password hashada

**byte**[] hashedPasswd = sc.hash(passwd);

**byte**[] hashedNonce = sc.hash(nonce);

outStream.writeInt(hashedPasswd.length); //enviamos um inteiro

outStream.write(hashedPasswd); //ebnviamos um array de bytes

outStream.writeInt(hashedNonce.length); //enviamos um inteiro

outStream.write(hashedNonce); //enviamos um array de bytes

outStream.flush();

**if**(inStream.readShort() == -1){

System.***out***.println("Erro:Password errada!");

**return** -1;

}

**return** 0;

}**else**

System.***out***.println("Erro de Seguranca no servidor!");

} **catch** (Exception e) {

e.printStackTrace();

}

**return** -1;

}

/\*\*

\* Funcao que faz push de um ficheiro para o repositorio que se encontra no servidor

\* que tem o mesmo nome que o repositorio que se encontra localmente

\* **@param** file - ficheiro ao qual fazemos push

\* **@param** filename - nome do ficheiro ao qual vamos fazer push

\* **@param** outStream - objeto por onde escreve ao servidor

\* **@param** inStream - objeto por onde le ao servidor

\* **@param** sc - security handler, trata de toda a seguranca aqui envolvida

\* **@param** privKey - chave privada utilizada pelo security handler

\*/

**public** **void** push\_file(File file,String filename,

ObjectOutputStream outStream, ObjectInputStream inStream,

SecurityHandler sc,PrivateKey privKey){

**try** {

System.***out***.println("Vamos enviar um fichiero");

outStream.writeInt(***PUSH\_FILE***);

outStream.writeObject(filename);

outStream.flush();

**if**(inStream.readShort() == -1){

System.***out***.println("Erro: Nao tem permissao para entrar nesse ficheiro");

**return**;

}

**if**(inStream.readShort() == -1){

System.***out***.println("Erro: Repositorio nao existe");

**return**;

}

//enviar ambos

**if**(filename.contains(".sig") || filename.contains(".key.server")){

System.***out***.println("Nao pode enviar .sig ou .key.server");

msg.reject(outStream);

**return**;

}

msg.confirm(outStream);

**if**(msg.notModified(file,outStream, inStream)){

//enviar a assinatura

**byte**[] sg = sc.signature(file, privKey);

outStream.writeInt(sg.length);

outStream.write(sg);

SecretKey key = sc.getRandomSecretKey();

outStream.writeObject(key);

sc.SendEncryptWithPassword(file,key,msg,outStream);

System.***out***.println("-- O ficheiro " + file.getName() + " foi"

+ " enviado para o servidor");

}

**else**

System.***out***.println("-- O ficheiro " + file.getName() + "ja "

+ "se encontra actualizado no servidor");

} **catch** (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* Funcao que faz push de todos os ficheiros que se encontram no repositorio para o servidor

\* com o repositorio com o mesmo nome

\* Caso exista um ficheiro no servidor mas nao localmente, apaga-se o ficheiro do servidor

\* **@param** rep - repositorio ao qual vamos fazer push

\* **@param** repName - nome do repositorio ao qual vamos fazer push

\* **@param** outStream - objeto por onde escreve ao servidor

\* **@param** inStream - objeto por onde le ao servidor

\* **@param** sc - security handler, trata de toda a seguranca aqui envolvida

\* **@param** privKey - chave privada que eh utilizada pelo security handler

\*/

**public** **void** push\_rep(File rep,String repName,ObjectOutputStream outStream,

ObjectInputStream inStream,SecurityHandler sc,PrivateKey privKey){

**try** {

outStream.writeInt(***PUSH\_REP***);

**int** total = 0;

outStream.writeObject(repName);

**short** resp;

**if**((resp = inStream.readShort()) == 1){

total ++;

System.***out***.println("-- O repositorio " +repName+" foi criado no"

+ " servidor");

}

**else** **if**(resp == 0){

System.***out***.println("Erro: nao pode criar repositorios "

+ "nas paginas de outros utilizadores");

**return**;

}

File[] files = rep.listFiles( **new** FileFilter(){

@Override

**public** **boolean** accept(File pathname) {

**return** pathname.isFile();

}

});

outStream.flush();

**if**(inStream.readShort() == -1){

System.***out***.println("Erro: Nao tem acesso a esse repositorio");

**return**;

}

**int** size = files.length;

outStream.writeInt(size); //numero de ficheiros que vamos enviar

outStream.flush();

**for**(File fl: files){

outStream.writeObject(rep.getName() + "/" +fl.getName());

outStream.flush();

**if**(msg.notModified(fl,outStream, inStream)){

**byte**[] sg = sc.signature(fl, privKey);

outStream.writeInt(sg.length);

outStream.write(sg);

SecretKey key = sc.getRandomSecretKey();

outStream.writeObject(key);

sc.SendEncryptWithPassword(fl,key,msg,outStream);

total ++;

**short** caso = inStream.readShort();

**if**(caso == 1)

System.***out***.println("-- O ficheiro " + fl.getName() +

" foi adicionado ao servidor mas ja existia historico");

**else** **if**(caso == 0)

System.***out***.println("-- O ficheiro " + fl.getName() +

" vai ser adicionado ao servidor");

**else**

System.***out***.println("-- O ficheiro " + fl.getName() +

" foi actualizado no servidor");

}

} //end of for

**short** delSize = inStream.readShort();

**for**(**int** i = 0; i < delSize; i++){

**try** {

total ++;

System.***out***.printf("-- O ficheiro %s foi apagado do servidor\n"

,(String)inStream.readObject());

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

}

}

**if** (total == 0)

System.***out***.println("-- Nao foi mudificado nada no lado do"

+ " servidor");

}**catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Funcao que faz pull de um ficheiro de um servidor para o repositorio local

\* **@param** filename - nome do ficheiro ao qual vamos fazer pull

\* **@param** outStream - objeto por onde escreve ao servidor

\* **@param** inStream - objeto por onde le ao servidor

\* **@param** sc - security handler, trata de toda a seguranca aqui envolvida

\*/

**public** **void** pull(String filename,

ObjectOutputStream outStream, ObjectInputStream inStream,

SecurityHandler sc){

**try** {

outStream.writeInt(***PULL***);

outStream.writeObject(filename);

outStream.flush();

**int** answ;

**long** date;

**if**((answ = inStream.readShort()) == 1) //DIRETORIO

pull\_rep(filename, inStream, outStream,sc);

**else** **if**(answ == 0){

System.***out***.println(filename.substring(0,filename.lastIndexOf("/")));

**if**(!**new** File(filename.substring(0,filename.lastIndexOf("/"))).exists()){

System.***out***.println("Erro:Diretorio nao encontrado");

msg.reject(outStream);

**return**;

}

**else**

msg.confirm(outStream);

**if**(inStream.readShort() == -1){

System.***out***.println("Erro: Nao tem acesso a esse diretorio/ficheiro");

**return**;

}

date = inStream.readLong();

String fl = filename.split("/")[filename.split("/").length - 1];

**if**(lastModified(filename) < date){

msg.confirm(outStream);

File file = msg.receiveFile(filename,inStream);

**byte**[] Key = **new** **byte**[inStream.readInt()];

inStream.read(Key);

sc.decryptFile(file, sc.getKeyFromArray(Key));

file.setLastModified(date);

//File sign = msg.receiveFile(filename + ".sig", inStream);

System.***out***.println("-- O ficheiro " + fl +

" foi copiado do servidor");

}

**else**{

System.***out***.println("-- O ficheiro " +

fl + " encountra se actualizado");

outStream.writeShort(-1);

}

}

**else**

System.***out***.println("Erro: Ficheiro nao encontrado");

} **catch** (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* Funcao que faz pull de todos os ficheiros que se encontram num repositorio do servidor

\* para o repositorio local com o mesmo nome

\* **@param** filename - nome no repositorio ao qual vamos fazer pull

\* **@param** inStream - objeto por onde le ao servidor

\* **@param** outStream - objeto por onde escreve ao servidor

\* **@param** sc - security hanlder, trata de toda a seguranca aqui envolvida

\* **@throws** Exception

\*/

**public** **void** pull\_rep(String filename,ObjectInputStream inStream,

ObjectOutputStream outStream,SecurityHandler sc) **throws** Exception{

**if**(inStream.readShort() == -1){

System.***out***.println("Erro: nao tem acesso a esse diretorio");

**return**;

}

**boolean** ourRep = inStream.readBoolean();

**if**(!ourRep && !**new** File(filename).exists())

System.***out***.printf("-- O repositorio %s do utilizador %s foi"

+ " copiado do servidor\n",filename.split("/")[0],username);

**long** date;

**int** total = 0,size = inStream.readInt();

String fl,totalName;

File file;

**final** ArrayList<String> recNames = **new** ArrayList<String>();

**for**(**int** i = 0; i < size; i++){

fl = (String) inStream.readObject();

totalName = getTotalName(filename,fl);

recNames.add(totalName.split("/")[totalName.split("/").length -1]);

date = inStream.readLong();

**if**(lastModified(totalName) < date){

msg.confirm(outStream);

file = msg.receiveFile(totalName,inStream);

**byte**[] Key = **new** **byte**[inStream.readInt()];

inStream.read(Key);

sc.decryptFile(file, sc.getKeyFromArray(Key));

file.setLastModified(date);

//File sign = msg.receiveFile(filename + ".sig", inStream);

**if**(ourRep)

System.***out***.println("-- Copiamos o ficheiro " + fl + " do servidor");

total ++;

}

**else**{

msg.reject(outStream);

}

}

**if**(!ourRep)

System.***out***.println("-- Copiamos o repositorio do servidor");

size = inStream.readInt();

**for**(**int** i = 0; i < size; i ++){

String name = (String) inStream.readObject();

totalName = getTotalName(filename,name);

**if**(**new** File(totalName).exists()){

System.***out***.println("-- O ficheiro " + name + " existe"

+ " localmente mas foi eliminado do servidor");

total++;

}

}

**if**(total == 0)

System.***out***.println("-- Nenhuma alteracao a informar");

}

/\*\*

\* Funcao que devolve o nome completo, ou seja, o diretorio de um ficheiro

\* **@param** filename - nome do ficheiro ao qual vamos depois devolver o diretorio completo

\* **@param** fl

\* **@return** o diretorio do ficheiro com nome filename

\*/

**private** String getTotalName(String filename,String fl){

**if**(filename.split("/").length == 1){

**new** File(filename).mkdir();

**return** filename + "/" + fl;

}

**if**(filename.split("/").length == 2){

String[] folderNames = filename.split("/");

**if**(folderNames[0].equals(username)){

**new** File(filename).mkdir();

**return** folderNames[1] +"/"+fl;

}

**new** File(folderNames[0]).mkdir();

**if**(!**new** File(filename).exists()){

System.***out***.printf("-- Vamos copiar o diretorio %s "

+ "do utilizador %s.\n",folderNames[1],folderNames[0]);

}

**new** File(filename).mkdirs();

**return** filename+"/"+fl;

}

**else** //tamanho 3

**return** filename;

}

/\*\*

\* Funcao que vai permitir um utilizador partilhar o seu repositorio com um outro

\* utilizador

\* **@param** outStream - objeto por onde escreve ao servidor

\* **@param** myRep - nome do repositorio que vai ser partilhado

\* **@param** userTo - nome do utilizador nao criador do repositorio que vai ganhar acesso ao

\* dito repositorio

\* **@param** inStream - objeto por onde le ao servidor

\*/

**public** **void** share(ObjectOutputStream outStream,String myRep,String userTo,

ObjectInputStream inStream){

**try** {

outStream.writeInt(***SHARE***);

outStream.writeObject(myRep);

outStream.writeObject(userTo);

**if**(inStream.readShort() != 1){

System.***out***.println("Erro: Nao pode partilhar com o proprio utilizador");

**return**;

}

outStream.flush();

**if**(inStream.readInt() == 1){

**int** ans;

**if**((ans = inStream.readInt()) == 1)

System.***out***.println("-- O repositorio "+myRep+" foi "

+ "partilhado com o utilizador " + userTo);

**else** **if**(ans == 0)

System.***out***.printf("Erro: Utilizador %s ja tinha acesso"

+ " ao repositorio %s\n",userTo,myRep);

**else**

System.***out***.println("Erro: Ocorreu um erro");

}

**else**

System.***out***.println("Erro: O user "+userTo+" nao existe");

} **catch** (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Funcao que permite retirar as permissoes de acesso de um utilizador sobre um

\* repositorio

\* **@param** outStream - obejto por onde escreve ao servidor

\* **@param** myRep - nome do repositorio ao qual vai retirar as permissoes de acesso

\* **@param** userTo - nome do utilizador nao criador que vai perder as permissoes de acesso

\* **@param** inStream - objeto por onde le ao servidor

\*/

**public** **void** remove(ObjectOutputStream outStream,String myRep,

String userTo,ObjectInputStream inStream){

**try** {

outStream.writeInt(***REMOVE***);

outStream.writeObject(myRep);

outStream.writeObject(userTo);

outStream.flush();

**if**(inStream.readInt() == 1){

**int** ans;

**if**((ans = inStream.readInt()) == 1)

System.***out***.println("-- O utilizador "+userTo+" foi "

+ "removido doo repositorio " + myRep);

**else** **if**(ans == 0)

System.***out***.printf("Erro: Utilizador %s nao tinha acesso"

+ " ao repositorio %s\n",userTo,myRep);

**else**

System.***out***.println("Erro: Ocorreu um erro ao fazer o remove");

}

**else**

System.***out***.println("Erro: O user " +userTo + " nao existe");

} **catch** (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Funcao que indica a data da ultima vez que um ficheiro foi alterado

\* **@param** fileName - nome do ficheiro ao qual vamos avaliar

\* **@return** qual foi a ultima vez que um ficheiro foi alterado, -1 caso nao exista

\*/

**private** **long** lastModified(String fileName){

File file = **new** File(fileName);

**return** file.exists() ? file.lastModified() : -1;

}

}

Messager:

/\*\* Grupo sc005

\* Francisco JoÃ£o GuimarÃ£es Coimbra de Almeida AraÃºjo nÂº45701

\* Joana Correia MagalhÃ£es Sousa nÂº47084

\* JoÃ£o Marques de Barros Mendes Leal nÂº46394

\*/

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.nio.ByteBuffer;

**public** **class** Messager {

**private** **static** **final** **int** ***SIZE*** = 1024;

/\*\*

\* Funcao que trata do envio do ficheiro

\* **@param** file - ficheiro que vai ser enviado

\* **@param** outStream - objeto por onde escreve ao recetor

\* **@throws** IOException

\*/

**public** **void** sendFile(File file, ObjectOutputStream outStream) **throws** IOException{

FileInputStream fp = **new** FileInputStream(file.getPath());

**byte**[] aEnviar = **new** **byte**[***SIZE***];

outStream.write(ByteBuffer.*allocate*(4).putInt((**int**)file.length()).array(),0,4); //passar o tamanho total

**int** n;

**while**((n=fp.read(aEnviar,0,***SIZE***))>=0){

outStream.write(aEnviar,0,n);

}

outStream.flush();

fp.close();

}

/\*\*

\* Funcao que trata de rececao de um ficheiro

\* Recebemos fileSize, e os bytes do ficheiro, 1024 de cada vez

\* **@param**: fileName - Nome do ficheiro que vai ser recebido

\* **@throws** IOException

\*/

**public** File receiveFile(String fileName,ObjectInputStream inStream) **throws** IOException{

**byte**[] by = **new** **byte**[4];

**byte**[] fileBytes = **new** **byte**[***SIZE***];

File file = **new** File(fileName);

FileOutputStream fos = **new** FileOutputStream(fileName);

inStream.read(by,0,4);

**int** fileSize = ByteBuffer.*wrap*(by).getInt();

**int** n;

**while**(file.length()< fileSize){

n = inStream.read(fileBytes, 0, 1024);

fos.write(fileBytes, 0, n);

}

fos.close();

**return** file;

}

/\*\*

\* Funcao que trata do push do ficheiro

\* **@param** file - ficheiro ao qual vamos fazer o push

\* **@param** outStream - objeto por onde escreve ao recetor

\* **@param** inStream - objeto por onde le ao recetor

\* **@return** true se o push foi feito com sucesso; false caso contrario

\* **@throws** IOException

\*/

**public** **boolean** basic\_push(File file,

ObjectOutputStream outStream,

ObjectInputStream inStream) **throws** IOException{

outStream.writeLong(file.lastModified());

outStream.flush();

**if**(inStream.readShort() == 1){

sendFile(file, outStream);

**return** **true**;

}

**else**

**return** **false**;

}

/\*\*

\* Funcao que indica se um dado ficheiro foi alterado ou nao

\* **@param** file - ficheiro a ser avaliado

\* **@param** outStream

\* **@param** inStream

\* **@return** true se o ficheiro nao foi alterado; false caso contrario

\* **@throws** IOException

\*/

**public** **boolean** notModified(File file,ObjectOutputStream outStream,

ObjectInputStream inStream) **throws** IOException{

outStream.writeLong(file.lastModified());

outStream.flush();

**return** inStream.readShort() == 1;

}

/\*\*

\* Funcao que confirma a rececao do ficheiro

\* **@param** outStream - obejto por onde escreve ao recetor

\*/

**public** **void** confirm(ObjectOutputStream outStream){

**try** {

outStream.writeShort(1);

outStream.flush();

} **catch** (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Funcao que rejeita a rececao do ficheiro

\* **@param** outStream - objeto por onde escreve ao recetor

\*/

**public** **void** reject(ObjectOutputStream outStream){

**try** {

outStream.writeShort(-1);

outStream.flush();

} **catch** (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Funcao que trata do erro ocorrido na parte de seguranca

\* **@param** outStream

\*/

**public** **void** securityError(ObjectOutputStream outStream){

**try** {

outStream.writeShort(-10);

outStream.flush();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

myGit

/\*\*Grupo sc005

\* Francisco JoÃ£o GuimarÃ£es Coimbra de Almeida AraÃºjo nÂº45701

\* Joana Correia MagalhÃ£es Sousa nÂº47084

\* JoÃ£o Marques de Barros Mendes Leal nÂº46394

\*/

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.net.Socket;

**import** java.security.KeyStore;

**import** java.security.PrivateKey;

**import** java.security.PublicKey;

**import** java.security.cert.Certificate;

**import** java.util.Scanner;

**import** javax.net.SocketFactory;

**import** javax.net.ssl.SSLSocketFactory;

**public** **class** myGit{

**private** **static** **int** *ped*;

**private** **static** **boolean** *missingPass* = **false**;

**private** **static** String *username*;

**private** **static** String *serverAddress*;

**private** **static** String *password*;

**private** **static** String *filename*;

**private** **static** String *userTo*;

**private** **final** **static** **int** ***INIT*** = 0;

**private** **final** **static** **int** ***ADD\_USER*** = 10;

**private** **final** **static** **int** ***PUSH*** = 20;

**private** **final** **static** **int** ***PULL*** = 30;

**private** **final** **static** **int** ***SHARE*** = 40;

**private** **final** **static** **int** ***REMOVE*** = 50;

**private** **static** SecurityHandler *sc* = **new** SecurityHandler("Vou ter de tirar daqui a password");

**private** **static** PublicKey *pubKey*;

**private** **static** PrivateKey *privKey*;

**public** **static** **void** main(String[] args) **throws** FileNotFoundException {

*ped* = *isValid*(args);

myGit client = **new** myGit();

client.startClient();

FileInputStream kfile;

**try** {

kfile = **new** FileInputStream("clientkeystore.dd");

KeyStore kstore = KeyStore.*getInstance*("JKS");

//ate aqui tudo bem

kstore.load(kfile,"qwerty".toCharArray());

**if**(kstore.containsAlias("client"))

System.***out***.println("Nao contem");

Certificate cert = kstore.getCertificate("client");

**if**(cert == **null**)

System.***out***.println("Esta null");

*pubKey* = cert.getPublicKey();

*privKey* = (PrivateKey) kstore.getKey("client",

"qwerty".toCharArray());

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Funcao que inicia o cliente

\*/

**public** **void** startClient (){

ClientThread newClientThread = **new** ClientThread();

newClientThread.start();

}

/\*\*

\* Funcao que trata de ligacao do cliente com o servidor

\* **@param** ServerAddress - o endereco do servidor ao qual vai fazer a ligacao

\* **@return** a socket da ligacao estabelecida

\*/

**public** Socket startConnection(String ServerAddress){

Socket sSoc = **null**;

System.*setProperty*("javax.net.ssl.trustStore", "clientkeystore.dd");

SocketFactory sf = SSLSocketFactory.*getDefault*( );

String srvAdrs[] = ServerAddress.split(":");

**try** {

sSoc = sf.createSocket(srvAdrs[0],Integer.*parseInt*(srvAdrs[1]));

} **catch** (IOException e) {

System.***out***.println("Erro: Servidor nao encontrado!");

System.*exit*(-1);

}

**return** sSoc;

}

//Threads utilizadas para comunicacao com o servidor

**class** ClientThread **extends** Thread {

**private** Socket socket = **null**;

/\*\*

\* Trata da iteracao entre o cliente e o servidor

\*/

**public** **void** run(){

**try** {

**if**(*ped* == ***INIT***)

*init*(*filename*);

**else** **if**(*ped* == -1)

System.***out***.println("Erro: comando nao reconhecido");

**else**{

socket = startConnection(*serverAddress*);

ObjectOutputStream outStream = **new** ObjectOutputStream(socket.getOutputStream());

ObjectInputStream inStream = **new** ObjectInputStream(socket.getInputStream());

Scanner reader = **new** Scanner(System.***in***);

**if**(*missingPass*){

System.***out***.println("-- Porfavor diga a password");

*password* = reader.nextLine();

}

ClientServerHandler csh = **new** ClientServerHandler(*username*,*password*);

//enviar logo aquilo que vai enviar obrigatoriamente

csh.sendInitInfo(outStream); //acho que vou ter de tirar daqui coisas, por exemplo nao precisamos de enviar a password logo

//preciso de fazer o log in

**int** addUs;

**if**((addUs = csh.addUser(outStream, inStream,*sc*)) == -1)

**return**;

**switch**(*ped*){

**case** ***ADD\_USER***:

**if**(addUs == 0)

System.***out***.println("-- O utilizador " + *username*

+ " ja existe");

System.***out***.println("Vai para aqui");

outStream.writeInt(-1);

outStream.flush();

**break**;

**case** ***PUSH***:

System.***out***.println("Entrou no caso do push");

File file = **new** File(*filename*);

**if**(file.exists()){

**if**(file.isDirectory())

csh.push\_rep(file,*filename*,outStream, inStream, *sc*, *privKey*);

**else**{

**if**(*filename*.split("/").length == 1){

outStream.writeInt(-1);

System.***out***.println("ERRO: Tem de especificar o diretorio");

**break**;

}

csh.push\_file(file,*filename*, outStream, inStream,*sc*,*privKey*);

}

}**else**{

System.***out***.println("Eroo: Esse ficheiro nao existe");

outStream.writeInt(-1);

}

**break**;

**case** ***PULL***:

csh.pull(*filename*, outStream, inStream,*sc*);

**break**;

**case** ***SHARE***:

csh.share(outStream, *filename*, *userTo*, inStream);

**break**;

**case** ***REMOVE***:

csh.remove(outStream, *filename*, *userTo*, inStream);

**break**;

}

reader.close();

outStream.close();

inStream.close();

socket.close();

}

} **catch** (Exception e) {

System.***out***.println("Erro:nao se conseguiu conectar ao servidor");

}

}

}

/\*\*

\* Funcao que verifica se os argumentos inseridos sao validos

\* **@param** arg - os argumentos que vao ser avaliados

\* **@return**

\*/

**public** **static** **int** isValid(String[] arg){

**int** min = 0,tam = arg.length;

**if**(tam < 2 || tam > 8)

**return** -1;

**if**(arg[0].equals("-init") && tam == 2){

*filename* = arg[1];

**return** ***INIT***;

}

*username* = arg[0];

*serverAddress* = arg[1];

**if**(tam == 2){

*missingPass* = **true**;

**return** ***ADD\_USER***;

}

**else** **if** (tam >=4)

*password* = arg[3];

**else**

**return** -1;

**if**(tam >= 4){

**if**(!arg[2].equals("-p")){

*missingPass* = **true**;

min = 2;

}

**if**(tam + min == 4)

**return** ***ADD\_USER***;

**if**(tam + min == 6){

*filename* = arg[5 - min];

**if**(arg[4 - min].equals("-push"))

**return** ***PUSH***;

**else** **if**(arg[4 - min].equals("-pull"))

**return** ***PULL***;

}

**else** **if**(tam + min == 7){

*filename* = arg[5 - min];

*userTo* = arg[6 - min];

**if**(arg[4 - min].equals("-share"))

**return** ***SHARE***;

**else** **if**(arg[4 - min].equals("-remove"))

**return** ***REMOVE***;

}

}

**return** -1;

}

/\*\*

\* Funcao que trata da criacao de um repositorio local

\* **@param** folderName - nome do repositorio a ser criado

\*/

**public** **static** **void** init(String folderName){

**if**(**new** File(folderName).mkdirs())

System.***out***.println("-- O repositorio " + folderName + " foi criado localmente");

**else**

System.***out***.println("-- Erro na criacao do " + folderName);

}

}

repCatalog:

/\*\*Grupo sc005

\* Francisco JoÃ£o GuimarÃ£es Coimbra de Almeida AraÃºjo nÂº45701

\* Joana Correia MagalhÃ£es Sousa nÂº47084

\* JoÃ£o Marques de Barros Mendes Leal nÂº46394

\*/

**import** java.io.BufferedReader;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.IOException;

/\*\*

\* Cada utilizador vai ter no seu dir de util um ficheiro, que nao pode ser acedido, que

\* dita quem pode ver que repositorios no seu dir do genero

\*

\* <repositorio>:<util1;util2;util3>

\*

\* **@author** Utilizador

\*

\*/

**public** **class** repCatalog{

**private** String fileName;

/\*\*

\* Construtor do repCatalog

\* **@param** fileName

\*/

**public** repCatalog(String fileName){

**this**.fileName = fileName;

}

/\*\*

\* Cria um novo repositorio

\* **@param** repName - nome do repositorio a ser criado

\* **@param** Creator - nome do criador do repositorio a ser criado

\* **@param** sc - security hanlder, trata de toda a seguranca aqui envolvida

\* **@throws** IOException

\*/

**public** **void** addRep(String repName,String Creator,SecurityHandler sc) **throws** IOException{

**new** File(repName).mkdir();

File usersFile = **new** File(Creator + "/" + fileName);

usersFile.createNewFile();

**try** {

sc.createMAC(usersFile);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Adiciona um utilizador ah lista daqueles que o criador partilha o seu repositorio

\* **@param** sharer - Nome do criador quer partilhar o seu repositorio

\* **@param** repName - Nome do repositorio que o criador quer partilhar

\* **@param** userToShare - O nome do utilizador a partilhar com

\* **@param** sc - security handler, trata de toda a seguranca aqui envolvida

\* **@throws** IOException

\*/

**public** **short** addUser(String sharer,String repName,

String userToShare,SecurityHandler sc) **throws** IOException{

**try** {

System.***out***.println("Estamos no add user");

**if**(sc.checkMAC(**new** File(sharer+"/"+fileName)) == 0){

System.***out***.println("Erro no mac!!!!!!!!");

**return** 0;

}

**if**(addUserToFile(sharer,userToShare,repName, sc)){

sc.createMAC(**new** File (sharer+ "/" +fileName)); //nao esta a fazer o update

**return** 1;

}

sc.createMAC(**new** File (sharer+ "/" +fileName)); //nao esta a fazer o update

} **catch** (Exception e1) {

e1.printStackTrace();

}

**return** -1;

}

/\*\*

\* Funcao auxiliar do addUser

\* **@param** sharer - Nome do criador quer partilhar o seu repositorio

\* **@param** userToShare - O nome do utilizador a partilhar com

\* **@param** repName - Nome do repositorio que o criador quer partilhar

\* **@param** sc - security handler, trata de toda a seguranca aqui envolvida

\* **@return** true se a funcao foi feit com sucesso; false caso contrario

\*/

**private** **boolean** addUserToFile(String sharer,String userToShare,String repName,

SecurityHandler sc){

File file = **new** File(sharer + "/" +fileName);

File tempFile = **new** File(sharer + "/myTempFile.txt");

System.***out***.println("Entrou no add user");

**try** {

sc.decryptFile(file);

BufferedReader reader = **new** BufferedReader(**new** FileReader(file));

BufferedWriter writer = **new** BufferedWriter(**new** FileWriter(tempFile));

**boolean** isArleadyThere = **false**,foundRep = **false**;

String currentLine;

String[] line,users;

StringBuilder userLine = **new** StringBuilder();

**while**((currentLine = reader.readLine()) != **null**) {

line = currentLine.split(":");

**if**(line[0].equals(repName)){

foundRep = **true**;

users = line[1].split(";");

**for**(String user:users){

**if**(user.equals(userToShare))

isArleadyThere = **true**;

userLine.append(user + ";");

}

**if**(!isArleadyThere)

userLine.append(userToShare+";");

userLine.deleteCharAt(userLine.length() - 1);

writer.write(repName +":"+userLine.toString()+

System.*getProperty*("line.separator"));

}

**else**

writer.write(currentLine + System.*getProperty*("line.separator"));

}

**if**(!foundRep)

writer.write(repName +":"+userToShare + System.*getProperty*("line.separator"));

writer.close();

reader.close();

**if**(!file.delete())

System.***out***.println("Could not delete file");

//Rename the new file to the filename the original file had.

**if** (!tempFile.renameTo(file))

System.***out***.println("Could not rename file");

sc.encrypthFile(file);

**return** isArleadyThere;

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** **false**;

}

/\*\*

\* Funcao que indica se um utilizador eh criador de um dado repositorio

\* **@param** sharer - nome de utilizador a avaliar

\* **@param** repName - nome do repositorio a ser utilizado para fazer a verificacao

\* **@return** true se sharer eh criador do repositorio repName; false caso contario

\*/

**public** **boolean** isCreator(String sharer,String repName){

**return** **new** File(sharer + "/" + repName).exists();

}

/\*\*

\* Funcao que remove um utilizador da lista daqueles que o criador quer

\* partilhar o seu repositorio

\* **@param** repName - nome do repositorio ao qual o utilizador vai ser removido

\* **@param** remover - nome do criador do repositorio

\* **@param** user - nome do utilizador a ser removido

\*/

**public** **int** removeUser(String sharer,String userToRemove,

String repName,SecurityHandler sc){

File file = **new** File(sharer + "/" +fileName);

File tempFile = **new** File(sharer + "/myTempFile.txt");

**try** {

**if**(sc.checkMAC(file) == 0)

**return** 0;

sc.decryptFile(file);

**int** wasThere = -1;

BufferedReader reader = **new** BufferedReader(**new** FileReader(file));

BufferedWriter writer = **new** BufferedWriter(**new** FileWriter(tempFile));

String currentLine;

String[] line,users;

StringBuilder userLine = **new** StringBuilder();

**while**((currentLine = reader.readLine()) != **null**) {

line = currentLine.split(":");

**if**(line[0].equals(repName)){

users = line[1].split(";");

**for**(String user:users){

**if**(user.equals(userToRemove))

wasThere = 1;

**else**

userLine.append(user + ";");

}

**if**(userLine.length() > 0){

userLine.deleteCharAt(userLine.length() - 1);

writer.write(repName +":"+userLine.toString()+

System.*getProperty*("line.separator"));

}

}

**else**

writer.write(currentLine + System.*getProperty*("line.separator"));

}

writer.close();

reader.close();

**if**(!file.delete())

System.***out***.println("Could not delete file");

//Rename the new file to the filename the original file had.

**if** (!tempFile.renameTo(file))

System.***out***.println("Could not rename file");

sc.encrypthFile(file);

sc.createMAC(file);

**return** wasThere;

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** -1;

}

/\*\*

\* Funcao que indica se um certo utilizador tem acesso a um dado repositorio

\* **@param** repName - nome do repositorio

\* **@param** userToLook - nome do utilizador que vamos avaliar

\* **@return** true se userToLook tem acesso ao repName; false caso contrario

\* **@throws** IOException

\* **@requires** File repName exists

\*/

**public** **boolean** hasAccess(String repName, String userToLook,SecurityHandler sc) **throws** IOException{

System.***out***.println("Entrou no hasAcess");

String username = repName.split("/")[0];

System.***out***.println("repName: " + repName);

File file = **new** File(username+"/"+fileName);

**return** hasAccess(file,repName,userToLook,sc);

}

/\*\*

\* Funcao auxiliar de hasAccess

\* **@param** file - nome do ficheiro que se encontra no repositorio

\* **@param** repName - nome do repositorio

\* **@param** user - nome do utilizador a ser avaliado

\* **@return** true se user tem acesso a repName; false caso contrario

\*/

**private** **boolean** hasAccess(File file,String repName,String user,SecurityHandler sc){

**try** {

System.***out***.println("repName: " + repName);

sc.decryptFile(file);

FileReader fr = **new** FileReader(file);

BufferedReader br = **new** BufferedReader(fr);

String line;

**while**((line = br.readLine()) != **null**){

System.***out***.println(line);

String[] folderNames = repName.split("/");

**if**(line.split(":")[0].equals(folderNames[1]) ){

String[] users = line.split(":")[1].split(";");

**for**(String fileUser:users){

**if**(fileUser.equals(user)){

fr.close();

br.close();

sc.encrypthFile(file);

**return** **true**;

}

}

}

}

fr.close();

br.close();

sc.encrypthFile(file);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** **false**;

}

/\*\*

\* Funcao que indica a data da ultima vez que o ficheiro foi alterado

\* **@param** user - nome do criador do repositorio

\* **@param** myRep - nome do repositorio onde se encontra o ficheiro

\* **@param** file - nome do ficheiro que queremos verificar a data

\* **@return** a data em que o ficheiro foi modificado, -1 caso nao exista

\*/

**public** **long** lastModified(String user,String myRep,String file){

File fl = **new** File(user+"/" +myRep + "/" + file);

**return** fl.exists() ? fl.lastModified() : -1;

}

}

SecurityHandler:

/\*\*Grupo sc005

\* Francisco JoÃ£o GuimarÃ£es Coimbra de Almeida AraÃºjo nÂº45701

\* Joana Correia MagalhÃ£es Sousa nÂº47084

\* JoÃ£o Marques de Barros Mendes Leal nÂº46394

\*/

**import** java.io.BufferedInputStream;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.math.BigInteger;

**import** java.nio.charset.StandardCharsets;

**import** java.nio.file.Files;

**import** java.security.MessageDigest;

**import** java.security.NoSuchAlgorithmException;

**import** java.security.PrivateKey;

**import** java.security.SecureRandom;

**import** java.security.Signature;

**import** java.util.Arrays;

**import** javax.crypto.Cipher;

**import** javax.crypto.CipherInputStream;

**import** javax.crypto.CipherOutputStream;

**import** javax.crypto.Mac;

**import** javax.crypto.SecretKey;

**import** javax.crypto.SecretKeyFactory;

**import** javax.crypto.spec.IvParameterSpec;

**import** javax.crypto.spec.PBEKeySpec;

**import** javax.crypto.spec.PBEParameterSpec;

**import** javax.crypto.spec.SecretKeySpec;

**public** **class** SecurityHandler {

**private** **byte**[] ivBytes = {0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,

0x19,0x1A,0x1B,0x1C,0x1D,0x1E,0x1F,0x20};

**private** IvParameterSpec ivSpec = **new** IvParameterSpec(ivBytes);

**private** PBEParameterSpec spec = **new** PBEParameterSpec(ivBytes, 20, ivSpec); //esta a dar mal no linux

**private** SecretKey key;

**private** String password;

/\*\*

\* Construtor de SecurityHandler

\* **@param** password - password que o utilizador defuniu ao iniciar o servidor

\*/

**public** SecurityHandler(String password) {

**this**.password = password;

}

/\*\*

\* Metodo auxiliar para criar um MAC

\* **@param** password - password utilizado para criar uma chave

\* **@param** rec\_file - ficheiro utilizado para criar o MAC

\* **@return** um MAC num array de bytes

\* **@throws** Exception

\*/

**private** **byte**[] macFunction (String password,File rec\_file) **throws** Exception{

File file = **new** File(rec\_file.getPath());

Mac mac = Mac.*getInstance*("HmacSHA256");

SecretKey key = **new** SecretKeySpec(password.getBytes(), "HmacSHA256");

mac.init(key);

FileInputStream fis = **new** FileInputStream(rec\_file.getPath());

**byte**[] b = **new** **byte**[(**int**) file.length()];

fis.read(b);

mac.update(b);

fis.close();

**return** mac.doFinal();

}

/\*\*

\* Cria um MAC a um ficheiro especifico

\* **@param** rec\_file - ficheiro a ser utlizado para criar o MAC

\* **@throws** Exception

\*/

**public** **void** createMAC(File rec\_file) **throws** Exception {

String loc = rec\_file.getPath().split("\\.")[0];

SecretKey key = **new** SecretKeySpec(password.getBytes(), "HmacSHA256");

ObjectOutputStream oosk = **new** ObjectOutputStream(**new** FileOutputStream(loc + "macKey.key"));

ObjectOutputStream oos = **new** ObjectOutputStream( **new** FileOutputStream(loc + ".mac") );

oos.write(macFunction(password,rec\_file));

**byte**[] keyEncoded = key.getEncoded();

oosk.write(keyEncoded);

oosk.flush();

oos.flush();

oosk.close();

oos.close();

}

/\*\*

\* Metodo que verifica se um dado MAC eh valido

\* **@param** rec\_file - o ficheiro utilizado para comparar o seu MAC

\* anteriormente criado

\* **@return** 1 se o MAC eh valido; 0 caso contrario

\* **@throws** Exception

\*/

**public** **int** checkMAC(File rec\_file)

**throws** Exception{

String name = rec\_file.getPath().split("\\.")[0];

File macToComp = **new** File(name+ ".mac");

//para obter a chave secreta

File macKey = **new** File(name + "macKey.key");

ObjectInputStream fisMac = **new** ObjectInputStream(

**new** FileInputStream(name + "macKey.key"));

**byte** [] keyEncoded = **new** **byte**[(**int**) macKey.length()];

fisMac.read(keyEncoded);

SecretKeySpec key = **new** SecretKeySpec(keyEncoded, "HmacSHA256");

//criar um novo mac para comparar com o original

Mac mac = Mac.*getInstance*("HmacSHA256");

mac.init(key);

FileInputStream fis = **new** FileInputStream(rec\_file.getPath());

**byte**[] b = **new** **byte**[(**int**) rec\_file.length()];

fis.read(b);

mac.update(b);

fis.close();

**byte**[] newMac = macFunction(password,rec\_file);

fisMac.close();

//comparacao dos macs

**byte**[] fileBytes = Files.*readAllBytes*(macToComp.toPath());

**byte**[] originalMac = Arrays.*copyOfRange*(fileBytes, 6, fileBytes.length);

**if**(originalMac.length != newMac.length)

**return** 0;

**if**(!Arrays.*equals*(originalMac,newMac))

**return** 0;

**return** 1;

}

/\*\*

\* Funcao que cria uma chave com o algoritmo PBEWithHmacSHA256AndAES\_128

\* **@return** uma SecretKey

\* **@throws** Exception

\*/

**public** SecretKey getRandomSecretKey() **throws** Exception{

PBEKeySpec keySpec = **new** PBEKeySpec(password.toCharArray());

SecretKeyFactory kf = SecretKeyFactory.*getInstance*("PBEWithHmacSHA256AndAES\_128");

**return** kf.generateSecret(keySpec);

}

/\*\*

\* Metodo que obtem uma secretkey a partir de um array de bytes

\* **@param** key - um array de bytes por onde vai ser obtido a chave secreta

\* **@return** uma secretkey resultante

\*/

**public** SecretKey getKeyFromArray(**byte**[] key){

**return** **new** SecretKeySpec(key,0,key.length,"PBEWithHmacSHA256AndAES\_128");

}

/\*\*

\* Funcao que envia o ficheiro ecriptado

\* **@param** file - ficheiro que vai ser ecriptado e depois enviado

\* **@param** key - chave a ser utilizada para ecriptar

\* **@param** msg - messager que vai ser utilizado para tratar do envio

\* **@param** out - o resultado da ecriptacao

\*/

**public** **void** SendEncryptWithPassword(File file,SecretKey key,Messager msg,ObjectOutputStream out){

Cipher c;

**try** {

c = Cipher.*getInstance*("PBEWithHmacSHA256AndAES\_128");

c.init(Cipher.***ENCRYPT\_MODE***, key, spec);

FileInputStream fis;

FileOutputStream fos;

CipherOutputStream cos;

fis = **new** FileInputStream(file);

fos = **new** FileOutputStream(file.getName() + ".cif");

cos = **new** CipherOutputStream(fos, c);

**byte**[] b = **new** **byte**[1024];

**int** i = fis.read(b);

**while** (i != -1) {

cos.write(b, 0, i);

i = fis.read(b);

}

cos.close();

fis.close();

File fl = **new** File(file.getName() + ".cif");

msg.sendFile(fl, out);

fl.delete();

}**catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Funcao que tratar da decriptacao de um ficheiro usando uma secretkey dada

\* **@param** file - ficheiro que vai seu decriptado

\* **@param** key - chave a ser utilizada para a decriptacao

\* **@throws** Exception

\*/

**public** **void** decryptFile(File file,SecretKey key) **throws** Exception{

Cipher c = Cipher.*getInstance*("PBEWithHmacSHA256AndAES\_128");

c.init(Cipher.***DECRYPT\_MODE***,key,spec);

String path = file.getPath();

FileInputStream fis = **new** FileInputStream(file);

FileOutputStream fos = **new** FileOutputStream(path + ".temp");;

CipherInputStream cis = **new** CipherInputStream(fis, c);

File newFile = **new** File(path + ".temp");

**byte**[] b = **new** **byte**[16];

**int** i = cis.read(b);

**while** (i != -1) {

fos.write(b, 0, i);

i = cis.read(b);

}

fis.close();

fos.close();

cis.close();

**if**(!file.delete())

System.***out***.println("Nao apagou");

newFile.renameTo(file);

}

/\*\*

\* Funcao que trata de ecriptacao de um ficheiro

\* **@param** file - ficheiro a ser ecriptado

\* **@throws** Exception

\*/

**public** **void** encrypthFile(File file) **throws** Exception{

PBEKeySpec keySpec = **new** PBEKeySpec(password.toCharArray());

SecretKeyFactory kf = SecretKeyFactory.*getInstance*("PBEWithHmacSHA256AndAES\_128");

key = kf.generateSecret(keySpec);

Cipher c = Cipher.*getInstance*("PBEWithHmacSHA256AndAES\_128");

c.init(Cipher.***ENCRYPT\_MODE***,key,spec);

String path = file.getPath();

FileInputStream fis = **new** FileInputStream(file);

FileOutputStream fos = **new** FileOutputStream(path + ".temp");;

CipherInputStream cis = **new** CipherInputStream(fis, c);

File newFile = **new** File(path + ".temp");

**byte**[] b = **new** **byte**[16];

**int** i = cis.read(b);

**while** (i != -1) {

fos.write(b, 0, i);

i = cis.read(b);

}

fis.close();

fos.close();

cis.close();

**if**(!file.delete())

System.***out***.println("Nao apagou");

newFile.renameTo(file);

}

/\*\*

\* Funcao que trata da decriptacao de um ficheiro

\* **@param** file - ficheiro a ser decriptado

\* **@throws** Exception

\*/

**public** **void** decryptFile(File file) **throws** Exception{

PBEKeySpec keySpec = **new** PBEKeySpec(password.toCharArray());

SecretKeyFactory kf = SecretKeyFactory.*getInstance*("PBEWithHmacSHA256AndAES\_128");

key = kf.generateSecret(keySpec);

SecretKeySpec secretSpec = **new** SecretKeySpec

(key.getEncoded(), "PBEWithHmacSHA256AndAES\_128");

Cipher c = Cipher.*getInstance*("PBEWithHmacSHA256AndAES\_128");

c.init(Cipher.***DECRYPT\_MODE***,secretSpec,spec);

String path = file.getPath();

FileInputStream fis = **new** FileInputStream(file);

FileOutputStream fos = **new** FileOutputStream(path + ".temp");;

CipherInputStream cis = **new** CipherInputStream(fis, c);

File newFile = **new** File(path + ".temp");

**byte**[] b = **new** **byte**[16];

**int** i = cis.read(b);

**while** (i != -1) {

fos.write(b, 0, i);

i = cis.read(b);

}

fis.close();

fos.close();

cis.close();

**if**(!file.delete())

System.***out***.println("Nao apagou");

newFile.renameTo(file);

}

/\*\*

\* Funcao que gera um nonce aleatoriamente

\* **@return** um valor aleatorio

\*/

**public** String generateNonce(){

SecureRandom random = **new** SecureRandom();

**return** **new** BigInteger(130,random).toString();

}

/\*\*

\* Funcao que trata da sintese da mensagem

\* **@param** value - mensagem a ser utilizada para tratar da sintese

\* **@return** a sinetese da mensagem num array de bytes

\* **@throws** NoSuchAlgorithmException

\*/

**public** **byte**[] hash (String value) **throws** NoSuchAlgorithmException{

MessageDigest md = MessageDigest.*getInstance*("SHA-256");

**return** md.digest(value.getBytes(StandardCharsets.***UTF\_8***));

}

/\*\*

\* Funcao que trata da assinatura do ficheiro

\* **@param** M - ficheiro pelo qual se vai realizar a assinatura

\* **@param** privKey - chave secreta utilizada para fazer a assinatura

\* **@return** uma assinatura num array de bytes

\* **@throws** Exception

\*/

**public** **byte**[] signature(File M,PrivateKey privKey) **throws** Exception{

Signature s = Signature.*getInstance*("SHA256withRSA");

s.initSign(privKey);

FileInputStream fis = **new** FileInputStream(M);

BufferedInputStream bufin = **new** BufferedInputStream(fis);

**byte**[] buffer = **new** **byte**[1024];

**int** len;

**while** ((len = bufin.read(buffer)) >= 0)

s.update(buffer, 0, len);

bufin.close();

fis.close();

**return** s.sign();

}

}

userCatalog:

/\*\*

\* Grupo sc005

\* Francisco JoÃ£o GuimarÃ£es Coimbra de Almeida AraÃºjo nÂº45701

\* Joana Correia MagalhÃ£es Sousa nÂº47084

\* JoÃ£o Marques de Barros Mendes Leal nÂº46394

\*/

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.security.NoSuchAlgorithmException;

**import** java.util.Arrays;

/\*\*

\* Ideia, usar o ficheiro como o catalogo de utilizadores

\* Podemos usar User U ou

\* String username && String password

\* **@author** Utilizador

\*/

**public** **class** userCatalog {

**private** **static** File *passwords*;

**public** userCatalog(){

*passwords* = **new** File("passwords.txt");

**try** {

*passwords*.createNewFile();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Adiciona um utilizador ao ficheiro passwords

\* **@param** username - nome do utilizador a adicionar

\* **@param** password - utilizado para o security handler

\* **@param** sc - security handler, trata de toda a seguranca aqui envolvida

\* **@param** password - password do utililizador a adicionar

\* **@throws** IOException

\*/

**public** **void** addUser(String username, String password,SecurityHandler sc){

StringBuilder Stb = **new** StringBuilder();

**try** {

Stb.append(username + ":" + password + System.*getProperty*("line.separator"));

**if**(existsUser(username,sc))

**return**;

sc.decryptFile(*passwords*);

FileWriter fw = **new** FileWriter(*passwords*,**true**);

fw.write(Stb.toString());

**new** File(username).mkdir(); //create user folder

fw.close();

sc.encrypthFile(*passwords*);

sc.createMAC(*passwords*);

} **catch** (Exception e1) {

e1.printStackTrace();

}

}

/\*\*

\* Funcao que indica se existe um utilizador

\* **@param** username - nome do utilizador que vamos avaliar

\* **@param** sc - security handler, trata de toda a seguranca aqui envolvida

\* **@return** true se o utilizador username existe; false caso contrario

\* **@throws** IOException

\*/

**public** **boolean** existsUser(String username,SecurityHandler sc){

BufferedReader reader;

**try** {

sc.decryptFile(*passwords*);

reader = **new** BufferedReader(**new** FileReader(*passwords*));

//temos de decriptar, talvez linha a linha

String line;

**while**((line = reader.readLine()) != **null**){

**if**(line.split(":")[0].equals(username)){

reader.close();

sc.encrypthFile(*passwords*);

**return** **true**;

}

}

reader.close();

sc.encrypthFile(*passwords*);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** **false**;

}

/\*\*

\* Funcao que trata do login do utilizador

\* **@param** username - nome do utilizador

\* **@param** passwordGiven - password do utilizador

\* **@param** sc - security handler, trata de toda a seguranca aqui envolvida

\* **@return** true se o login foi feito com sucesso; false caso contrario

\*/

**public** **boolean** login(String username,**byte**[] passwordGiven,SecurityHandler sc){

BufferedReader reader;

**try** {

sc.decryptFile(*passwords*);

reader = **new** BufferedReader(**new** FileReader((*passwords*)));

String line;

**while**((line = reader.readLine()) != **null**){

**if**(line.split(":")[0].equals(username)){

**byte**[] gotPass = sc.hash(line.split(":")[1]);

**if**(Arrays.*equals*(gotPass, passwordGiven)){

reader.close();

sc.encrypthFile(*passwords*);

**return** **true**;

}

reader.close();

sc.encrypthFile(*passwords*);

**return** **false**;

}

}

reader.close();

sc.encrypthFile(*passwords*);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** **false**;

}

}

myGitServer:

/\*\*Grupo sc005

\* Francisco JoÃ£o GuimarÃ£es Coimbra de Almeida AraÃºjo nÂº45701

\* Joana Correia MagalhÃ£es Sousa nÂº47084

\* JoÃ£o Marques de Barros Mendes Leal nÂº46394

\*/

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**import** java.security.KeyStore;

**import** java.security.PrivateKey;

**import** java.security.PublicKey;

**import** java.util.Scanner;

**import** javax.net.ServerSocketFactory;

**import** javax.net.ssl.SSLServerSocketFactory;

**import** java.security.cert.Certificate;

//Servidor myServer

**public** **class** myGitServer{

**private** repCatalog reps = **new** repCatalog("users.txt");

**private** userCatalog users = **new** userCatalog();

**private** **static** SecurityHandler *sc\_hd*;

**private** **final** **static** **int** ***ADD\_USER*** = 10;

**private** **final** **static** **int** ***PUSH\_REP*** = 20;

**private** **final** **static** **int** ***PUSH\_FILE*** = 30;

**private** **static** **final** **int** ***PULL*** = 40;

**private** **final** **static** **int** ***SHARE*** = 50;

**private** **final** **static** **int** ***REMOVE*** = 60;

**private** **static** String *pwd\_in*;

**private** **static** PublicKey *pubK*;

**private** **static** PrivateKey *privKey*;

**public** **static** **void** main(String[] args) {

System.***out***.println("servidor: main");

myGitServer server = **new** myGitServer();

**if**(args.length < 1 || args.length > 1){

System.***out***.println("Erro: Criacao do servidor so recebe o porto");

**return**;

}

Scanner sc = **new** Scanner(System.***in***);

//AO INICIAR O SERVIDOR, EH PRECISO PEDIR UMA PASSWORD AO UTILIZADOR

System.***out***.println("Porfavor de me a password");

*pwd\_in* = sc.nextLine();

sc.close();

*sc\_hd* = **new** SecurityHandler(*pwd\_in*);

File file = **new** File("passwords.txt");

**try** {

file.createNewFile();

} **catch** (IOException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

**if**(!**new** File("passwords.mac").exists()){

**try** {

System.***out***.println("Vai ser criado um MAC para as passwords");

*sc\_hd*.createMAC(file);

} **catch** (Exception e) {

e.printStackTrace();

}

}**else**{

**try** {

**if**(*sc\_hd*.checkMAC(file) == 0){

System.***out***.println("MAC ERRADO!!!! ABORT! ABORT!");

System.*exit*(-1);

}

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

FileInputStream kfile;

**try** {

kfile = **new** FileInputStream("serverkeystore.dd");

KeyStore kstore = KeyStore.*getInstance*("JKS");

kstore.load(kfile,"qwerty".toCharArray());

**if**(kstore.containsAlias("server"))

System.***out***.println("Nao contem");

Certificate cert = kstore.getCertificate("server");

*pubK* = cert.getPublicKey();

*privKey* = (PrivateKey) kstore.getKey("server",

"qwerty".toCharArray());

**if**(*privKey* == **null**)

System.***out***.println("Esta null");

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**int** serverPort = Integer.*parseInt*(args[0]);

server.startServer(serverPort);

}

/\*\*

\* Funcao que inicia o servidor

\* **@param** serverSocket - socket pelo qual e feito a ligacao com o servidor

\*/

**public** **void** startServer (**int** serverSocket){

ServerSocket sSoc = **null**;

**try** {

System.*setProperty*("javax.net.ssl.keyStore", "serverkeystore.dd");

System.*setProperty*("javax.net.ssl.keyStorePassword", "qwerty");

ServerSocketFactory ssf = SSLServerSocketFactory.*getDefault*( );

sSoc = ssf.createServerSocket(serverSocket);

} **catch** (IOException e) {

System.***err***.println(e.getMessage());

System.*exit*(-1);

}

**while**(**true**) {

**try** {

Socket inSoc = sSoc.accept();

ServerThread newServerThread = **new** ServerThread(inSoc);

newServerThread.start();

}

**catch** (IOException e) {

e.printStackTrace();

}

}

//sSoc.close();

}

**class** ServerThread **extends** Thread {

**private** Socket socket = **null**;

ServerThread(Socket inSoc) {

socket = inSoc;

}

**public** **void** run(){

**try** {

ObjectOutputStream outStream = **new** ObjectOutputStream(socket.getOutputStream());

ObjectInputStream inStream = **new** ObjectInputStream(socket.getInputStream());

String user = (String)inStream.readObject();

ServerClientHandler sch = **new** ServerClientHandler(user);

**int** resp;

**if**((resp = sch.AddUser(users, outStream, inStream, *sc\_hd*)) == 1 ){

**switch**(inStream.readInt()){

**case** ***ADD\_USER***:

**break**;

**case** ***PUSH\_REP***:

sch.push\_rep(inStream,outStream, reps,*sc\_hd*,*pubK*);

**break**;

**case** ***PUSH\_FILE***:

System.***out***.println("Entrou no push file");

sch.push\_file(reps, inStream, outStream,*sc\_hd*,*pubK*);

**break**;

**case** ***PULL***:

sch.pull(reps, outStream, inStream,*sc\_hd*,*privKey*);

**break**;

**case** ***SHARE***:

sch.share(outStream, inStream,reps,users, *sc\_hd*);

**break**;

**case** ***REMOVE***:

sch.remove(outStream, inStream, reps, users,*sc\_hd*);

**break**;

**default**:

System.***out***.println("Comando nao reconhecido");

}

}

**else** **if**(resp == 0){

outStream.close();

inStream.close();

socket.close();

System.*exit*(-1);

}

outStream.close();

inStream.close();

socket.close();

}**catch** (IOException e) {

e.printStackTrace();

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

}

}

}

}

ServerClientHandler:

/\*\* Grupo sc005

\* Francisco JoÃ£o GuimarÃ£es Coimbra de Almeida AraÃºjo nÂº45701

\* Joana Correia MagalhÃ£es Sousa nÂº47084

\* JoÃ£o Marques de Barros Mendes Leal nÂº46394

\*/

**import** java.io.File;

**import** java.io.FileFilter;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.security.PrivateKey;

**import** java.security.PublicKey;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** javax.crypto.Cipher;

**import** javax.crypto.SecretKey;

**public** **class** ServerClientHandler {

**private** String username;

**private** Messager msg;

**public** ServerClientHandler(String username){

**this**.username = username;

msg = **new** Messager();

}

/\*\*

\* Adiciona um utilizador a uma lista de utilizadores que se encontra no servidor

\* **@param** users - lista de utilizadores onde se vai adicionar o novo

\* **@param** outStream - objeto por onde escreve ao servidor

\* **@param** inStream - objeto por onde le ao servidor

\* **@param** sc - security handler, trata de toda a seguranca aqui envolvida

\*/

**public** **int** AddUser(userCatalog users,ObjectOutputStream outStream,

ObjectInputStream inStream, SecurityHandler sc){

**try** {

//assim verificamos sempre, mesmo quando nao vamos trabalhar no pass

**if**(sc.checkMAC(**new** File("passwords.txt")) == 0)

**throw** **new** SecurityException();

**if**(!users.existsUser(username,sc)){

System.***out***.println("O user nao existe");

msg.confirm(outStream);

**if**(inStream.readShort() == 1){ //LER, confirmado pelo util

String passwd = (String) inStream.readObject();

users.addUser(username, passwd,sc);

msg.confirm(outStream);

**return** 1;

}**else**{

**return** -1;

}

//falta me confirmar

}**else**{

msg.reject(outStream);

//enviar o nonce

String nonce = sc.generateNonce();

System.***out***.println("nonce = " +nonce);

outStream.writeObject(nonce); //escrevos uma string

outStream.flush();

//receber a password hashada

**int** size = inStream.readInt();

**byte**[] hashedPasswd = **new** **byte**[size]; //lemos um inteiro

inStream.read(hashedPasswd); //lemos um array de bytes

size = inStream.readInt(); //receboms um inteiro

**byte**[] hashedNonce = **new** **byte**[size];

inStream.read(hashedNonce); //recemos um array de bytes

**if**(!Arrays.*equals*(hashedNonce, sc.hash(nonce))){

System.***out***.println("Nonce diferente!");

}

//temos de comparar os hash das duas

**if**(!users.login(username, hashedPasswd,sc)){//nao consegui fazer login

msg.reject(outStream);

**return** -1;

}

System.***out***.println("Fez o login bem");

msg.confirm(outStream);

**return** 1;

}

}**catch** (SecurityException e){

System.***err***.println("MAC ERRADO!!!!");

msg.securityError(outStream);

**return** 0;

}

**catch** (Exception e) {

e.printStackTrace();

}

**return** -1;

//ou dizer que deu erro

}

/\*\*

\* Funcao que apos receber o pedido de push de um repositorio, faz o push de todos os

\* ficheiros que se encontram do repositorio local para o repositorio do servidor com

\* o mesmo nome

\* **@param** inStream - objeto por onde le ao servidor

\* **@param** outStream - objeto por onde escreve ao servidor

\* **@param** reps - lista dos repositorios que se encontram no servidor

\* **@param** sc - security handler, trata de toda a seguranca aqui envolvida

\* **@param** pubKey - chave publica utilizada no security handler

\*/

**public** **void** push\_rep(ObjectInputStream inStream,ObjectOutputStream outStream,

repCatalog reps,SecurityHandler sc,PublicKey pubKey){

**try** {

String repname = fullNameRep((String) inStream.readObject());

System.***out***.println("repname: " + repname);

**if**(!**new** File(repname).exists() && isCreator(repname)){

reps.addRep(repname, username,sc);

msg.confirm(outStream);

}

**else** **if**(!**new** File(repname).exists() && !isCreator(repname)){

outStream.writeShort(0);

**return**;

}

**else**

msg.reject(outStream); //ja exisita

**if**(!hasAccess(reps, repname, username, outStream,sc))

**return**;

**int** size = inStream.readInt(); //receber o num dos ficheiros

**final** ArrayList<File> allAddedFiles = **new** ArrayList<File>();

**for**(**int** i = 0; i < size; i++){

outStream.flush();

String filename = (String) inStream.readObject();

**long** date = inStream.readLong();

String totalName = repname + "/" +filename.split("/")[1];

File file = **new** File(totalName);

allAddedFiles.add(file);

**long** id;

**if**((id = file.lastModified()) < date){

msg.confirm(outStream);

**byte**[] sign = **new** **byte**[inStream.readInt()];

inStream.read(sign);

setSignature(sign,totalName + ".sig"); //nao sei se esta a dar bem

SecretKey key = (SecretKey) inStream.readObject();

addHist(totalName);

file = msg.receiveFile(totalName,inStream);

file.setLastModified(date);

setKey(key,pubKey,totalName);

**if**(id == 0 && **new** File(totalName + ".1").exists())

outStream.writeShort(1);

**else** **if**(id == 0)

outStream.writeShort(0);

**else**

outStream.writeShort(-1);

}

**else**

outStream.writeShort(-1);

}

File[] files = **new** File(repname).listFiles( **new** FileFilter(){

@Override

**public** **boolean** accept(File pathname) {

**char** lastChar = pathname.getName().charAt(

(**int**) (pathname.getName().length() - 1));

**return** !allAddedFiles.contains(pathname) &&

!Character.*isDigit*(lastChar)&& !pathname.getName().contains(".sig")

&& !pathname.getName().contains(".key.server");

}

});

outStream.writeShort(files.length);

String totalName;

**for**(File fl:files){

outStream.writeObject(fl.getName());

totalName = repname +"/"+fl.getName();

addHist(totalName); //tenho que testar

fl.delete();

File sig = **new** File(fl.getPath() + ".sig");

File keyServer = **new** File(fl.getPath() + ".key.server");

**if**(sig.exists())

sig.delete();

**if**(keyServer.exists())

keyServer.delete();

}

}**catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Funcao que trata de fazer o push de um ficheiro para um repositorio especifico

\* no servidor

\* **@param** reps - lista de repositorios

\* **@param** inStream - objeto por onde escreve ao servidor

\* **@param** outStream - objeto por onde le ao servidor

\* **@param** sc - security handler, trata de toda a seguranca aqui envolvida

\* **@param** pubKey - chave publica, eh utilizada no security hanlder

\*/

**public** **void** push\_file(repCatalog reps,ObjectInputStream inStream,

ObjectOutputStream outStream,SecurityHandler sc,PublicKey pubKey){

String endOfKeyServ = ".key.server";

**try** {

String filename = (String) inStream.readObject();

String fullName = fullNameFile(filename);

File file = **new** File(fullName);

**if**(!hasAccess(reps, fullName, username, outStream,sc))

**return**;

**if**(!**new** File(fullName.split("/")[0] + "/" +

fullName.split("/")[1]).exists()){

msg.reject(outStream);

**return**;

}

**else**

msg.confirm(outStream);

**if**(inStream.readShort() == -1)

**return**;

**long** date = inStream.readLong();

**if**(file.lastModified() < date){

msg.confirm(outStream);

//acho que tenho de enviar passos a passos

**byte**[] sign = **new** **byte**[inStream.readInt()];

inStream.read(sign);

setSignature(sign,fullName + ".sig"); //nao sei se esta a dar bem

SecretKey key = (SecretKey) inStream.readObject();

addHist(fullName);

file = msg.receiveFile(fullName,inStream);

file.setLastModified(date);

setKey(key,pubKey,fullName);

}

**else**

msg.reject(outStream);

}**catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Funcao que coloca a assinatura num dado ficheiro com um nome especifico

\* **@param** sign - a assinatura num array de bytes

\* **@param** filename - nome do ficheiro ao qual vai ser colocado a assinatura

\* **@throws** IOException

\*/

**private** **void** setSignature(**byte**[] sign,String filename) **throws** IOException{

**byte**[] fileBytes = **new** **byte**[1024];

File file = **new** File(filename);

**if**(file.exists())

file.delete();

FileOutputStream fos = **new** FileOutputStream(filename);

**int** fileSize = sign.length;

fos.write(fileBytes, 0, fileSize);

fos.close();

}

/\*\*

\* Funcao que cifra a chave publica do servidor

\* **@param** key - chave que eh utilizada para cifrar a chave publica

\* **@param** pubKey - chave publica que vai ser cifrada

\* **@param** filename

\* **@throws** Exception

\*/

**private** **void** setKey(SecretKey key,PublicKey pubKey,String filename) **throws** Exception{

String end = ".key.server";

Cipher cipher = Cipher.*getInstance*("RSA");

cipher.init(Cipher.***ENCRYPT\_MODE***, pubKey);

**byte**[] cipherData = cipher.doFinal(key.getEncoded());

File fl = **new** File(filename + end);

**if**(fl.exists())

fl.delete();

FileOutputStream fos = **new** FileOutputStream(filename + end);

fos.write(cipherData, 0, cipherData.length);

fos.close();

}

/\*\*

\* Funcao de trata de fazer pull de um ficheiro que se encontra num repositorio no servidor

\* para o repositorio local com o mesmo nome

\* **@param** rep - nome do repositorio de onde se vai fazer o pull dos seu ficheiros

\* **@param** outStream - objeto por onde escreve ao servidor

\* **@param** inStream - objeto por onde le ao servidor

\*/

**public** **void** pull(repCatalog rep,ObjectOutputStream outStream,

ObjectInputStream inStream,SecurityHandler sc,PrivateKey privKey){

String filename;

File file;

**try** {

filename = (String) inStream.readObject();//nome

String totalName;

**if**(filename.split("/").length == 1)

totalName = username + "/" + filename;

**else** **if**(filename.split("/").length == 2){

File test= **new** File(username +"/"+filename);

**if**(test.exists() && test.isFile())

totalName = username + "/" + filename;

**else**

totalName = filename;

}

**else**

totalName = filename;

file = **new** File(totalName);

//ver se esta em formato folder

**if**(file.isDirectory()){ //ver se eh diretoria

System.***out***.println("EH DIRETORIA");

outStream.writeShort(1);//dir

pull\_rep(file, totalName, rep, outStream, inStream,sc,privKey);

}

//ver se esta em formato file

**else** **if**(file.isFile()){

outStream.writeShort(0); //confirmar

outStream.flush();

**if**(inStream.readShort() == -1)

**return**;

**if**(!hasAccess(rep, totalName, username, outStream,sc)){

**return**;

}

**if**(msg.notModified(file,outStream, inStream)){

File keyServer = **new** File(file.getPath() + ".key.server");

**byte**[] decKey = getKey(privKey, keyServer);

msg.sendFile(file, outStream);

outStream.writeInt(decKey.length);

outStream.write(decKey);

//msg.sendFile(new File(file.getPath() + ".sig"), outStream);

}

}

**else**

outStream.writeShort(-1);

}**catch** (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* Funcao que obtem a chave que foi inicialmente cifrada

\* **@param** priKey - chave secreta utilizada para decifrar a chave que se

\* pretende obter

\* **@param** file

\* **@return** a chave decifrada num array de bytes

\* **@throws** Exception

\*/

**private** **byte**[] getKey(PrivateKey priKey,File file) **throws** Exception{

Cipher decrypt = Cipher.*getInstance*("RSA");

decrypt.init(Cipher.***DECRYPT\_MODE***, priKey);

**byte**[] encryptedArray = **new** **byte**[(**int**) file.length()];

FileInputStream fis = **new** FileInputStream(file);

fis.read(encryptedArray); //read file into bytes[]

fis.close();

**return** decrypt.doFinal(encryptedArray);

}

/\*\*

\*

\* Funcao de trata de fazer pull de todos os ficheiros que se encontram num repositorio

\* especifico no servidor para o repositorio local com o mesmo nome

\* **@param** file - o repositorio onde vamos buscar os ficheiros

\* **@param** totalName - nome do diretorio do repositorio

\* **@param** rep - lista de repositorios que se encontram no servidor

\* **@param** outStream - objeto por onde escreve ao servidor

\* **@param** inStream - objeto por onde le ao servidor

\* **@param** sc - security handler, trata de toda a seguranca aqui envolvida

\*/

**public** **void** pull\_rep(File file,String totalName,repCatalog rep,

ObjectOutputStream outStream,ObjectInputStream inStream,SecurityHandler sc,

PrivateKey privKey){

**if**(!hasAccess(rep, totalName, username, outStream,sc)){

**return**;

}

File[] files = file.listFiles( **new** FileFilter(){

@Override

**public** **boolean** accept(File pathname) {

**char** lastChar = pathname.getName().charAt(

pathname.getName().length() - 1);

**return** !Character.*isDigit*(lastChar) && !pathname.getName().contains(".sig")

&& !pathname.getName().contains(".key.server");

}

}); //nao ha subdir

**try** {

**if**(totalName.split("/")[0].equals(username))

outStream.writeBoolean(**true**); //eh o nosso util a fazer o push?

**else**

outStream.writeBoolean(**false**);

outStream.writeInt(files.length);

**final** ArrayList<String> sendFiles = **new** ArrayList<>();

**for**(File fl:files){

outStream.writeObject(fl.getName()); //enviar o nome

**if**(msg.notModified(fl,outStream, inStream)){

File keyServer = **new** File(fl.getPath() + ".key.server");

**byte**[] decKey = getKey(privKey, keyServer);

msg.sendFile(fl, outStream);

outStream.writeInt(decKey.length);

outStream.write(decKey);

//msg.sendFile(new File(file.getPath() + ".sig"), outStream);

}

sendFiles.add(fl.getName()); //nao sei se aqui ou dentro do notmod

}

//Envio o nome do primeiro historico dos que nao foram enviados

File[] histFiles = file.listFiles( **new** FileFilter(){

@Override

**public** **boolean** accept(File pathname) {

System.***out***.println(pathname);

**if**(pathname.getName().contains(".sig") || pathname.getName().contains(".key.server"))

**return** **false**;

**char** lastChar = pathname.getName().charAt(

(**int**) (pathname.getName().length() - 1));

String[] allDots = pathname.getName().split("\\.");

System.***out***.println(pathname.getName());

String fileActualName;

**if**(allDots.length < 2)

fileActualName = allDots[0];

**else**

fileActualName = allDots[0] + "." + allDots[1];

**return** lastChar == '1' && !sendFiles.contains(fileActualName);

}

});

outStream.writeInt(histFiles.length);

**for**(File fl : histFiles){

String[] allDots = fl.getName().split("\\.");

String fileActualName = allDots[0] + "." + allDots[1];

outStream.writeObject(fileActualName);

}

}**catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Funcao que trata da organizacao do acesso de um utilizador a um repositorio

\* **@param** outStream - objeto por onde escreve ao servidor

\* **@param** inStream - objeto por onde le ao servidor

\* **@param** reps - lista de repositorios que se encontram no servidor

\* **@param** users - lista de utilizadores que se econtram no servidor

\* **@param** sc - seciruty handler, trata de toda a seguranca aqui envolvida

\*/

**public** **void** share(ObjectOutputStream outStream,

ObjectInputStream inStream,repCatalog reps,userCatalog users,

SecurityHandler sc){

System.***out***.println("Vamos fazer o share");

**try** {

String myRep = (String) inStream.readObject();

String userTo = (String) inStream.readObject();

**if**(userTo.equals(username)){

msg.reject(outStream);

**return**;

}

**else**

msg.confirm(outStream);

**if**(users.existsUser(userTo,sc)){

outStream.writeInt(1); //first confirm

**if**(**new** File(username + "/" + myRep).exists() &&

reps.isCreator(username, myRep)){

**int** resp;

**if**((resp=reps.addUser(username, myRep, userTo, sc)) == 1)

outStream.writeInt(0);

**else** **if**(resp == -1)

outStream.writeInt(1);

**else**{

System.***out***.println("Mac do cliente errado");

outStream.writeInt(-10);

}

}

**else**

outStream.writeInt(-1);

}**else**

outStream.writeInt(-1);

} **catch** (SecurityException e) {

**throw** **new** SecurityException();

} **catch** (ClassNotFoundException | IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Funcao que trata de remover a permissao de acesso de um utilizador a um repositorio

\* especifico

\* **@param** outStream - objeto por onde escreve ao servidor

\* **@param** inStream - objeto por onde le ao servidor

\* **@param** reps - lista de repositorios que se encontram no servidor

\* **@param** users - lista de utilizadores que se encontram no servidor

\* **@param** sc - security handler, trata de toda a seguranca aqui envolvida

\*/

**public** **void** remove(ObjectOutputStream outStream,

ObjectInputStream inStream,repCatalog reps,userCatalog users,

SecurityHandler sc){

System.***out***.println("Vamos fazer remove");

**try** {

String myRep = (String) inStream.readObject();

String userTo = (String) inStream.readObject();

**if**(users.existsUser(userTo,sc)){

outStream.writeInt(1);

**if**(**new** File(username + "/" + myRep).exists() &&

reps.isCreator(username, myRep)){

**int** resp;

**if**((resp = reps.removeUser(username, userTo, myRep,sc)) == 1)

outStream.writeInt(1);

**else** **if**(resp == -1)

outStream.writeInt(0);

**else**{

outStream.writeInt(-10);

System.***err***.println("Mac errado!");

}

}

**else**

outStream.writeInt(-1);

}**else**

outStream.writeInt(-1);

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Funcao que trata do historico de um ficheiro

\* **@param** filename - nome do ficheiro onde vamos tratar do historico

\*/

**public** **void** addHist(String filename){

**if**(**new** File(filename).exists()){

**boolean** found = **false**;

**for**(**int** i = 1 ;!found;i++ ){

File file = **new** File(filename + "." + i);

**if**(!file.exists()){

**new** File(filename).renameTo(file);

found = **true**;

}

}

}

}

/\*\*

\* Funcao que devolve a diretoria de um ficheiro

\* **@param** filename - nome do ficheiro onde vamos descobrir o historico

\* **@return** o diretorio do ficheiro com nome filename

\*/

**private** String fullNameFile(String filename){

String[] allFiles = filename.split("/");

**if**(allFiles.length == 1)

**return** filename;

**else** **if**(allFiles.length == 2){

**return** username + "/" +filename;

}

**else**

**return** filename;

}

//Se o ficheiro for um folder

/\*\*

\* Funcao que devolve o diretorio de um repositorio

\* **@param** filename - nome do repositorio

\* **@return** o diretorio do repositorio

\*/

**private** String fullNameRep(String filename){

String[] allFiles = filename.split("/");

System.***out***.println("InFullNameRepFunc arg0 = " + filename);

**if**( allFiles.length == 1 ){

**return** username + "/" + filename;

}

**else** **if**(allFiles.length == 2){

**return** filename;

}

**else**

**return** filename;

}

/\*\*

\* Funcao que indica se um utilizador tem acesso a um repositorio especifico

\* **@param** reps - lista de repositorios que se encontram no servidor

\* **@param** fullName - diretorio do repositorio

\* **@param** username - nome do utilizador que vamos avaliar

\* **@param** outStream - objeto por onde escreve ao servidor

\* **@return** true se o utilizador username tem acesso ao repositorio com o diretorio fullName;

\* false caso contrario

\*/

**private** **boolean** hasAccess(repCatalog reps,String fullName,

String username,ObjectOutputStream outStream,SecurityHandler sc){

String[] folderNames = fullName.split("/");

**try** {

**if**(fullName.split("/")[1].equals("users.txt") ||

fullName.split("/")[1].equals("..")){

msg.reject(outStream);

**return** **false**;

}

**if**(isCreator(fullName)){

msg.confirm(outStream);

**return** **true**;

}

**if**(!reps.hasAccess(folderNames[0] +"/"+ folderNames[1], username,sc)){

msg.reject(outStream);

**return** **false**;

}

msg.confirm(outStream);

**return** **true**;

} **catch** (IOException e) {

e.printStackTrace();

}

**return** **false**;

}

/\*\*

\* Funcao que indica se um utilizador eh criador de um repositorio

\* **@param** fullNameFile - diretorio do repositorio

\* **@return** true se o utilizador username eh criador do repositorio com o diretorio

\* fullNameFile; false caso contrario

\*/

**private** **boolean** isCreator(String fullNameFile){

**return** username.equals(fullNameFile.split("/")[0]);

}

}