

Universidade da Beira Interior

Departamento de Informática



Departamento de
Informática

UBI: XIUUU: Troca de Segredos Criptográficos Seguro

Elaborado por:

Ana Maria Delgado, a37668

David Pires, a37272

Gabriel Esteves, a38488

Inês Roque, a37174

Renato Lopes, a37408

Segurança Informática

Orientador:

Professor Doutor Pedro Inácio

Janeiro 2020

Conteúdo

Conteúdo	i
Glossário	iv
1 Introdução	1
1.1 Enquadramento	1
1.2 Motivação	1
1.3 Objetivos	1
1.4 Constituição do grupo	1
1.5 Organização do Documento	2
2 Estado da Arte	3
2.1 Introdução	3
2.2 Conceitos Relevantes	3
2.2.1 <i>Password Based Key Derivation Function 2(PBKDF2)</i> . .	3
2.2.2 Protocolo de acordo de chaves <i>Diffie-Hellman</i>	4
2.2.3 Assinatura Digital	5
2.2.4 Funções de <i>hash</i>	5
2.3 Tecnologias Utilizadas	6
2.3.1 <i>Java</i>	6
2.4 Conclusões	6
3 Engenharia de Software	7
3.1 Introdução	7
3.2 Casos de Uso	7
3.3 Diagramas	8
3.4 Conclusões	9
4 Implementação	10
4.1 Introdução	10

4.2	Geração de um segredo criptográfico a partir de palavras-passe inseridas pelo utilizador, nomeadamente através de algoritmos como o Password Based Key Derivation Function 2 (PBKDF2)	10
4.3	Troca de um segredo criptográfico usando o protocolo de acordo de chaves <i>DiffieHellman</i>	13
4.4	Troca de um segredo criptográfico usando <i>Puzzles</i> de <i>Merkle</i>	15
4.5	Troca de um segredo criptográfico usando o Rivest, Shamir e Adleman (RSA)	17
4.6	Distribuição de novas chaves de cifra a partir de chaves pré-distribuídas	18
4.7	Implementar forma de ter a certeza de que o segredo partilhado é o mesmo dos dois lados	20
4.8	Manual de utilização	21
4.9	Conclusões	25
5	Conclusões e Trabalho Futuro	27
5.1	Conclusões Principais	27
5.2	Trabalho Futuro	27

Acrónimos

AES *Advanced Encryption Standard*

CLI *Client Line Interface*

DES *Data Encryption Standard*

CLI *Client Line Interface*

DES *Data Encryption Standard*

ECB *Electronic Code Block*

GUI *Graphical User Interface*

IP *Internet Protocol*

PBKDF2 *Password Based Key Derivation Function 2*

RSA *Rivest Shamir Adleman*

SHA1 *Secure Hash Algorithm 1*

SHA224 *Secure Hash Algorithm 224*

SHA256 *Secure Hash Algorithm 256*

SHA384 *Secure Hash Algorithm 384*

SHA512 *Secure Hash Algorithm 512*

Glossário

Agente de Confiança Toma o papel central e sabe todas as chaves de sessão geradas entre quaisquer dois intervenientes, e não as partilha com mais ninguém.

Alice Agente emissor da mensagem.

Bits É a menor unidade de informação que pode ser armazenada e transmitida através de uma comunicação de dados, este pode ser apenas 0 ou 1.

Bytes É uma unidade de informação digital, que é equivalente a oito bits.

Bob Agente recetor da mensagem.

Chat Troca de mensagens entre dois utilizadores em tempo real.

Chave Privada Chave em que só o dono pode ter acesso.

Chave Pública Chave que pode ser conhecida por todos.

Chave Secreta Igual à chave privada.

Chave Simétrica Chave que é igual para os dois agentes da troca de mensagens.

Claire Agente que ataca as comunicações.

Criptograma É um texto cifrado que obedece a um código e a uma lógica pré-determinada para decifrar a mensagem.

Hash Valor numérico de comprimento fixo que identifica exclusivamente dados.

Help No contexto da aplicação, um *help* é uma ajuda de modo textual que resume o modo de funcionamento do programa para o utilizador.

Interface Ferramenta que permite que o utilizador comunique com as diferentes partes de um programa.

JAVA É uma linguagem de programação orientada a objetos, desenvolvida pela *Sun Microsystems* na década de 90. Hoje pertence à empresa Oracle.

Localhost Localização do sistema que está a ser usado.

NetBeans É uma plataforma de desenvolvimento de software gratuita. Esta destina-se principalmente ao desenvolvimento de código Java, mas também suporta outras linguagens.

Capítulo 1

Introdução

1.1 Enquadramento

Este projeto, denominado *XIUUU: Troca de Segredos Criptográficos Seguro*, foi realizado no contexto da unidade curricular de Segurança Informática, que se enquadra no terceiro ano de Licenciatura em Engenharia Informática da Universidade da Beira Interior, no ano letivo 2019/2020.

1.2 Motivação

O projeto foi proposto, e será avaliado, pelo docente Pedro Inácio, e tem em vista a implementação e aprofundamento dos conhecimentos adquiridos durante as aulas práticas e teóricas da unidade curricular em que se enquadra, ao longo do atual semestre.

1.3 Objetivos

Este projeto tem como base a implementação de um sistema que permita a troca de segredos entre duas entidades: cliente e servidor. Possui diversas funcionalidades implementadas através de conceitos das áreas de criptografia.

1.4 Constituição do grupo

A constituição do grupo de trabalho a que se deve a realização deste documento, e respetivo projeto, foi da responsabilidades dos próprios elementos, que se encontram aqui descritos:

- Ana Maria Delgado, a37668
- David Pires, a37272
- Gabriel Esteves, a38488
- Inês Roque, a37174
- Renato Lopes, a37408

1.5 Organização do Documento

De modo a refletir o trabalho que foi feito, este documento encontra-se estruturado da seguinte forma:

1. Introdução - nesta secção será feita uma descrição geral do projeto, identificando os objetivos, a sua finalidade e enquadramento.
2. Estado da Arte - apresentação de várias definições importantes para a compreensão do documento. São também descritas as tecnologias utilizadas para elaborar o projeto.
3. Engenharia de Software - apresentação de diagramas que ajudarão na compreensão do projeto.
4. Implementação - descrição mais aprofundada do programa desenvolvido e é feita uma listagem de um conjunto estruturado de testes.
5. Reflexão Crítica e Problemas encontrados - análise ao trabalho realizado e são comparados os objetivos iniciais com os atingidos. É também feita uma revisão da segurança do sistema implementado.
6. Conclusão - Revisão do documento e funcionalidades a implementar caso o projeto continuasse a ser desenvolvido fora do seu contexto atual.

Capítulo 2

Estado da Arte

2.1 Introdução

O projeto foi desenvolvido para computador, utilizando a linguagem de programação *JAVA*, compilado e testado com o IDE *NetBeans* em sistemas operativos *Windows* ou *Linux*. A aplicação corre em Graphical User Interface (GUI). O relatório foi desenvolvido utilizando a ferramenta *ShareL^AT_EX*. Nesta secção são abordados tópicos e definições relevantes para compreender alguns algoritmos/-protocolos utilizados neste projeto, nomeadamente: o algoritmo *Password Based Key Derivation Function 2* (PBKDF2), o protocolo de acordo de chaves *Diffie-Hellman*, *Puzzles de Merkle*, o *Rivest, Shamir e Adleman* (RSA),

No final do capítulo é ainda descrita a linguagem de programação utilizada e a razão da sua escolha para o desenvolvimento deste projeto.

2.2 Conceitos Relevantes

2.2.1 *Password Based Key Derivation Function 2*(PBKDF2)

Esta função vai cifrar um segredo com o auxílio de uma palavra-passe, onde o cliente pode escolher entre várias funções de *hash*.

Com a palavra-passe introduzida, o programa irá pegar na mesma, em *salt*, no número de iterações que terá de executar e no tamanho da chave. O *salt* em con-

junto com a palavra-passe utilizada irá gerar um novo vetor de *bytes*. Através disso e com a chave secreta que foi gerada com o auxílio da função de *hash*, vai calcular uma nova chave (com o algoritmo AES). Após isso, usa a chave gerada anteriormente e a mensagem, encriptando a mesma com "AES/CBC/PKCS5Padding" usando o vetor de inicialização e transforma este conjunto de *bytes* em *base64* para assim ser perceptível ao utilizador.

2.2.2 Protocolo de acordo de chaves *Diffie-Hellman*

Este algoritmo serve para partilhar um segredo sem que tenha existido uma troca de chaves prévia entre a Alice (cliente emissor) e o Bob (cliente recetor). Usando o protocolo de acordo de chaves Diffie-Hellman um adversário (ou Claire) pode escutar a comunicação, mas não pode manipulá-la.

Este acordo de chaves descreve-se da seguinte forma:

- A Alice e o Bob escolhem um número primo fixo suficiente grande (p). Este número pode ser tornado público. No nosso caso esse número tem 1024 *bits*.
- De seguida, é definido um gerador g entre 1 e p que pode ser tornado público.
- Quando estes dois valores (p e g) são combinados entre ambos, a Alice escolhe um número x entre 1 e p , que irá ser a sua chave secreta, e calcula $X = g^x \bmod p$ e envia X ao Bob. X pode ser público.
- O Bob recebe X e escolhe um número y , que irá ser a sua chave secreta, entre 1 e p e calcula $Y = g^y \bmod p$ e envia Y para a Alice, o valor Y pode ser tornado público.
- Quando a Alice tiver o Y calcula k da seguinte forma: $k = Y^x \bmod p$. k será a chave secreta que ambos vão ter. O Bob calcula k recorrendo a $k = X^y \bmod p$.

De seguida, é feita uma assinatura deste ficheiro com SHA256 e RSA que será confirmada pelo utilizador recetor do segredo e, caso esta assinatura seja verificada, o segredo será mostrado. Caso contrário o segredo não será exibido. Esta

assinatura que é efetuada pela Alice é feita com a sua chave privada, e para que seja verificada pelo Bob ele tem que pedir à Alice a chave pública.

2.2.3 Assinatura Digital

Este mecanismo criptográfico procura substituir as assinaturas físicas tradicionais com recurso às três ferramentas criptográficas seguintes: infraestrutura de chave pública, cifra de chave pública *Rivest-Shamir-Adleman (RSA)* e funções de *hash*.

O processo típico envolve a obtenção do valor de *hash* do ficheiro a assinar e a encriptação do mesmo usando a cifra *RSA*.

Note-se que a encriptação com chave secreta não oferece qualquer ato de cifra-gem, já que qualquer pessoa pode descriptar o criptograma com a chave pública da pessoa. Isto permite, contudo, que qualquer pessoa possa verificar a assinatura digital de alguém, ao contrário do que acontece com a assinatura tradicional.

Garantidas estão as seguintes propriedades:

1. Integridade do texto-limpo
2. Autenticidade da informação
3. Não repúdio
4. Dificuldade de falsificação

2.2.4 Funções de *hash*

Uma função de *hash* é uma função que recebe um grupo de caracteres e gera um número de tamanho arbitrariamente grande, mas finito, denominado de valor de *hash*.

Funções de *hash* têm diversas funcionalidades, como por exemplo, são usadas para aumentar a rapidez de extração de dado de uma base de dados, como também são usadas para encriptar ou descriptar assinaturas digitais.

Algumas das propriedades destas funções incluem:

1. Resistência à descoberta de texto original
2. Resistência à descoberta de um segundo texto
3. Resistência à colisão

Para o contexto deste trabalho, as funções de hash que foram implementadas são as seguintes:

1. *Message Digest 5* (MD5)
2. *Secure Hash Algorithm 256* (SHA256)

2.3 Tecnologias Utilizadas

2.3.1 *Java*

Java é uma linguagem de programação orientada a objetos que começou a ser criada em 1991, na Sun Microsystems. Teve início com o *Green Project*, no qual os mentores foram Patrick Naughton, Mike Sheridan, e James Gosling. Este projeto não tinha intenção de criar uma linguagem de programação, mais sim de antecipar a “próxima onda” que aconteceria na área da informática e programação. Os idealizadores do projeto acreditavam que em pouco tempo os aparelhos domésticos e os computadores teriam uma ligação.

2.4 Conclusões

Com esta secção ficam introduzidos os conceitos técnicos utilizados ao longo deste relatório, facilitando assim o acompanhamento do raciocínio descrito.

Capítulo 3

Engenharia de Software

3.1 Introdução

Cada capítulo intermédio deve começar com uma breve introdução onde é explicado com um pouco mais de detalhe qual é o tema deste capítulo, e como é que se encontra organizado (i.e., o que é que cada secção seguinte discute).

3.2 Casos de Uso

Ao iniciar a aplicação, o utilizador depara-se com um menu onde pode escolher o perfil que pretende utilizar – Cliente ou Servidor. Nesta secção vamos-nos focar nos casos de uso de cada um destes perfis.

- **Servidor:** A aplicação em modo servidor, tem a função de receber conexões de clientes e registá-los no sistema de forma a poder fornecer a lista dos clientes disponíveis.

- **Cliente:** A aplicação em modo cliente é capaz de enviar segredos a outros clientes, receber segredos de outros, gerar segredos e chaves, e ainda pedir ao servidor uma listagem de todos os clientes que estão registados na aplicação nesse momento. Ao escolher o perfil cliente, é pedido ao utilizador que introduza o

nome (que é apenas uma maneira mais amigável de representar o cliente), o IP e porta que vai ficar à escuta na sua máquina (para fazer conexão com outros utilizadores) e ainda o IP do servidor (para que, se houver servidor, todos os clientes estejam ligados a ele). Para enviar um segredo basta selecionar um cliente da lista escolher o algoritmo com que se pretende cifrar esse mesmo segredo e escrever uma mensagem. Se o outro cliente estiver conectado, vai receber o segredo enviado.

3.3 Diagramas

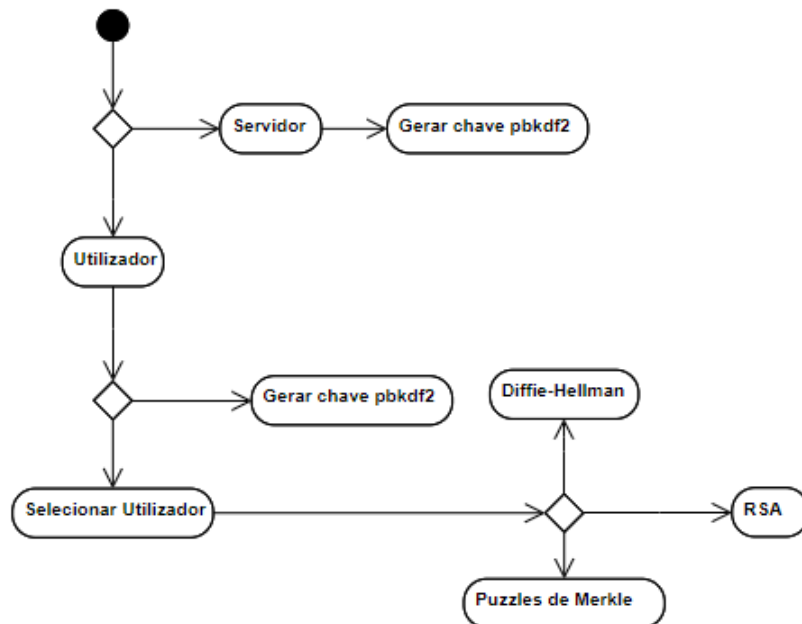


Figura 1.1

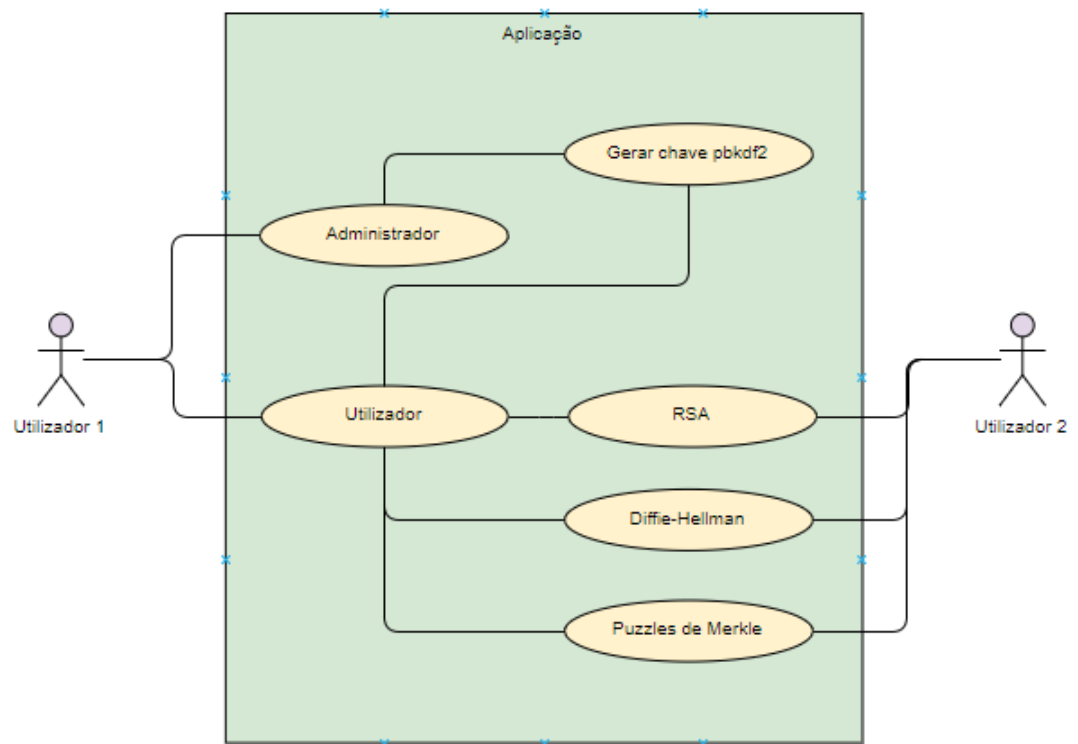


Figura 1.2

3.4 Conclusões

A implementação dos diagramas apresentados neste capítulo demonstram a solidez do projeto elaborado e fornecem uma descrição bastante detalhada do funcionamento do sistema desenvolvido.

Capítulo 4

Implementação

4.1 Introdução

Neste capítulo abordamos a implementação de cada um dos objectivos realizados no trabalho. Visto o foco deste trabalho ser a Segurança Informática, deixamos de parte outras implementações menos focadas no objetivo, como por exemplo, o *Design*.

Abordamos também neste capítulo, um Manual de utilização do programa.

Também, por não ser o nosso foco, decidimos não dedicar muito tempo ao tratamento de pequenos erros (como o *input* de *nicknames* iguais) que poderão dar problemas.

Neste capítulo cada secção intermédia irá representar um dos pontos dos objectivos do trabalho prático e discutirá a sua implementação .

4.2 **Geração de um segredo criptográfico a partir de palavras-passe inseridas pelo utilizador, nomeadamente através de algoritmos como o Password Based Key Derivation Function 2 (PBKDF2)**

O trecho de código seguinte corresponde a uma pedaço do ficheiro D_PBKDF2 correspondente à geração do criptograma:

4.2 Geração de um segredo criptográfico a partir de palavras-passe inseridas pelo utilizador, nomeadamente através de algoritmos como o Password Based Key Derivation Function 2 (PBKDF2)

11

```
String password = palavra.getText();
String secret = segredo.getText();
int option = box.getSelectedIndex() + 1;

PBKDF2 pbkdf2 = new PBKDF2(password, option);
String criptogram = pbkdf2.encrypt(secret);

JOptionPane.showMessageDialog(this,
    "Criptograma gerado: \n" + criptogram,
    "Informao",
    JOptionPane.INFORMATION_MESSAGE);
```

Excerto de Código 4.1: Trecho de código usado no projeto.

O trecho de código seguinte corresponde a um trecho da classe PBKDF2 usada para a realização deste algoritmo.

```
Cipher dcipher;

byte[] random = getRandom();
int iterationCount = 1024;
int keyLength = 128; //tamanho da chave
SecretKey key;
byte[] iv;

public PBKDF2(String passPhrase, int opcao_hash) throws Exception {
    SecretKeyFactory factory = null;
    switch (opcao_hash) {
        case 1: { //SHA1
            factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
            break;
        }
        case 2: { //SHA224
            factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA224");
            break;
        }
        case 3: { //SHA256
            factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
            break;
        }
        case 4: { //SHA384
            factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA384");
            break;
        }
        case 5: { //SHA512
            factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA512");
            break;
        }
    }
}
```


4.2 Geração de um segredo criptográfico a partir de palavras-passe inseridas pelo utilizador, nomeadamente através de algoritmos como o Password Based Key Derivation Function 2 (PBKDF2)

12

```
KeySpec spec = new PBEKeySpec(passPhrase.toCharArray(), random, iterationCount,
    keyLength);
SecretKey temp = factory.generateSecret(spec);
key = new SecretKeySpec(temp.getEncoded(), "AES");
dcipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
}

public String encrypt(String data) throws Exception {
    dcipher.init(Cipher.ENCRYPT_MODE, key);
    AlgorithmParameters params = dcipher.getParameters();
    iv = params.getParameterSpec(IvParameterSpec.class).getIV();
    byte[] utf8EncryptedData = dcipher.doFinal(data.getBytes());
    String base64EncryptedData = Base64.getEncoder().encodeToString(
        utf8EncryptedData);
    return base64EncryptedData;
}

//gera aleatorio
private static byte[] getRandom() throws NoSuchAlgorithmException {
    SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
    byte[] random = new byte[16];
    sr.nextBytes(random);
    return random;
}
```

Excerto de Código 4.2: Trecho de código usado no projeto.

Gerar segredo criptográfico através de chave gerada por palavra-passe (PBKDF2): Esta função vai cifrar um segredo com o auxílio de uma palavra-passe, onde o cliente pode escolher entre várias funções de *hash*.

Com a palavra-passe introduzida, o programa irá pegar na mesma, em *salt*, no número de iterações que terá de executar e no tamanho da chave. O *salt* em conjunto com a palavra-passe utilizada irá gerar um novo vetor de *bytes*. Através disso e com a chave secreta que foi gerada com o auxílio da função de *hash*, vai calcular uma nova chave (com o algoritmo AES). Após isso, usa a chave gerada anteriormente e pega na mensagem, encriptando a mesma com "AES/CBC/PKCS5Padding" usando o vetor de inicialização e transforma este conjunto de bytes em *base64* para assim ser perceptível ao utilizador.

4.3 Troca de um segredo criptográfico usando o protocolo de acordo de chaves *DiffieHellman*

O trecho de código seguinte mostra um pedaço da função `sendSecret()` correspondente à parte onde este o algoritmo para a execução deste ponto é invocado pela Alice.

```
SecureRandom rnd = new SecureRandom();
BigInteger P, G, x, X, Y, K;

boolean b;
do {
    P = BigInteger.probablePrime(1024, rnd);
    b = P.isProbablePrime(100);
} while(!b);
System.out.println("Generated P: " + P.toString());

G = DiffieHellman.nextRandomBigInteger(P);
System.out.println("Generated G: " + G.toString());

x = DiffieHellman.nextRandomBigInteger(P);
System.out.println("Generated x: " + x.toString());

X = G.modPow(x, P); //  $X = G^x \bmod P$ 

out.writeObject(P);
out.flush();
out.writeObject(G);
out.flush();
out.writeObject(X);
out.flush();

Y = (BigInteger) in.readObject();

K = Y.modPow(x, P);

String criptograma = AES.encrypt(message, K.toString());
out.writeObject(criptograma);
out.flush();
```

Excerto de Código 4.3: Trecho de código usado no projeto.

O seguinte trecho corresponde à parte do Bob.

```
P = (BigInteger) in.readObject();
G = (BigInteger) in.readObject();
X = (BigInteger) in.readObject();
```

4.3 Troca de um segredo criptográfico usando o protocolo de acordo de chaves *DiffieHellman*

14

```
y = DiffieHellman.nextRandomBigInteger(P);  
  
Y = G.modPow(y, P); //  $X = g^x \bmod p$   
  
out.writeObject(Y);  
out.flush();  
  
K = X.modPow(y, P);  
String cript = (String) in.readObject();  
message = AES.decrypt(cript, K.toString());
```

Excerto de Código 4.4: Trecho de código usado no projeto.

O protocolo de acordo de chaves *Diffie-Hellman* serve para partilhar um segredo sem que tenha existido uma troca de chaves prévia entre a Alice (cliente emissor) e o Bob (cliente recetor). Usando o protocolo de acordo de chaves *Diffie-Hellman* um adversário (ou Claire) pode escutar a comunicação, mas não pode manipulá-la.

Este acordo de chaves descreve-se da seguinte forma:

- A Alice e o Bob escolhem um número primo fixo suficiente grande (p). Este número pode ser tornado público. No nosso caso esse número tem 1024 *bits*.
 - Para realizar este passo utilizamos a função `probablePrime` que a biblioteca `java.math.BigInteger` nos disponibiliza, e assim obtemos um número que provavelmente é primo. O valor de retorno tem uma probabilidade de não ser primo de 2^{-100} . Este valor vai ainda ser confirmado dentro de um ciclo *do while* que fica a correr enquanto a função `isProbablePrime` confirma se o número p é mesmo número primo. Esta função é disponibilizada pela biblioteca referida acima.
- De seguida, é definido um gerador g entre 1 e p que pode ser tornado público.
 - Gera-se este valor fazendo uma chamada à função `nextRandomBigInteger` que se encontra na classe `DiffieHelman`.
- Quando estes dois valores (p e g) são combinados entre ambos, a Alice escolhe um número x entre 1 e p , que irá ser a sua chave secreta, e calcula $X = g^x \bmod p$ e envia X ao Bob. X pode ser público.

- O Bob recebe X e escolhe um número y , que irá ser a sua chave secreta, entre 1 e p e calcula $Y = g^y \bmod p$ e envia Y para a Alice, o valor Y pode ser tornado público.
- Quando a Alice tiver o Y calcula k da seguinte forma: $k = Y^x \bmod p$. k será a chave secreta que ambos vão ter. O Bob calcula k recorrendo a $k = X^y \bmod p$.

Depois usando o AES encripta-se a mensagem é enviada ao outro utilizador, que por sua vez já terá a chave k para poder descriptar e ler a mensagem.

4.4 Troca de um segredo criptográfico usando *Puzzles de Merkle*

O trecho de código seguinte mostra um pedaço da função `sendSecret()` correspondente à parte onde este o algoritmo para a execução deste ponto é invocado pela Alice.

```
MerklePuzzle mp = new MerklePuzzle();
int totalPuzzles = 10000;
int keyLen = 4;

ArrayList<byte[]> puzzles = new ArrayList<>();
ArrayList<String> keys = new ArrayList<>();
for (int i = 0; i < totalPuzzles; i++) {
    String aux = mp.random_string(16);
    keys.add(i, aux);
    byte[] cipherText = mp.encrypt(mp.random_key(keyLen), (aux + "PUZZLE" + i));
    puzzles.add(cipherText);
}

Collections.shuffle(puzzles);

out.writeObject(puzzles);
out.flush();
out.writeObject(MerkleCipher);
out.flush();
out.writeInt(keyLen);
out.flush();
```

```
String chosen = (String) in.readObject();

String keyChosen = keys.get(Integer.parseInt(chosen)); // chave simtrica
byte[] encryptedMessage = null;

if (MerkleCipher.equalsIgnoreCase("AES-ECB")) {

    byte[] encodedKey = keyChosen.getBytes();
    SecretKey sk = new SecretKeySpec(encodedKey, 0, encodedKey.length, "AES");

    encryptedMessage = AES.encrypt(message, sk);

} else {

    String keyZ = keyChosen + keyChosen;
    byte[] temp = keyZ.getBytes(Charset.forName("UTF-8"));
    SecretKey sk = new SecretKeySpec(temp, 0, 8, "DES");

    encryptedMessage = DES.encrypt(message, sk);

}

out.writeObject(encryptedMessage);
out.flush();
```

Excerto de Código 4.5: Trecho de código usado no projeto.

O seguinte trecho corresponde à parte do Bob.

```
MerklePuzzle mp = new MerklePuzzle();

ArrayList<byte[]> puzzles = (ArrayList) in.readObject();
cipherMode = (String) in.readObject();
int keyLen = in.readInt();

int chosen = new SecureRandom().nextInt(puzzles.size());
String tempKey = "";
boolean solved = false;
while(!solved) {
    tempKey = mp.decrypt(mp.random_key(keyLen), puzzles.get(chosen));

    if (tempKey != null && tempKey.contains("PUZZLE"))
        solved = true;
}

System.out.println("TotalPuzzles= " + puzzles.size());
System.out.println("Key= " + tempKey.substring(0, 16) + "\nPuzzle= " + tempKey.substring
(22));
```

4.5 Troca de um segredo criptográfico usando o Rivest, Shamir e Adleman (RSA)

17

```
String key = tempKey.substring(0, 16);

out.writeObject(tempKey.substring(22));
out.flush();

byte[] encodedKey = key.getBytes();

byte[] criptogram = (byte[]) in.readObject();

if (cipherMode.equalsIgnoreCase("AES-ECB")) {
    SecretKey sk = new SecretKeySpec(encodedKey, 0, encodedKey.length, "AES");
    message = AES.decrypt(criptogram, sk);
} else {
    String keyZ = key + key;
    byte[] temp = keyZ.getBytes(Charset.forName("UTF-8"));
    SecretKey sk = new SecretKeySpec(temp, 0, 8, "DES");

    message = DES.decrypt(criptogram, sk);
}
```

Excerto de Código 4.6: Trecho de código usado no projeto.

Neste algoritmo a Alice cria os puzzles cada um com chaves diferentes.(Excerto de Código 4.5) Cada *puzzle* é encriptado usando o algoritmo de cifra DES com uma chave aleatória diferente (Cada chave gerada é guardada numa lista que só Alice tem acesso). Depois dos *puzzles* serem gerados são baralhados e enviados para o Bob.(Excerto de código 4.6) Depois dos puzzles serem enviados ao Bob, ele escolhe um aleatoriamente e tenta decifrá-lo. Quando consegue decifrar fica a saber o número do *puzzle* e uma chave. O Bob envia então para a Alice o número do puzzle que decifrou e guarda a chave.A Alice recebe o número do *puzzle*, vai à sua lista que contem as chaves e vê qual a chave que esse *puzzle* continha. A partir desse momento ambos têm acesso a uma chave simétrica. A Alice cifra a mensagem com o algoritmo de cifra escolhido anteriormente (AES-ECB ou DES)(Excerto de Código 4.6) e envia o criptograma gerado para o Bob que usa a chave para o decifrar e ter acesso ao segredo que lhe foi enviado.

4.5 Troca de um segredo criptográfico usando o Rivest, Shamir e Adleman (RSA)

O trecho de código seguinte mostra um pedaço da função `sendSecret()` correspondente à parte onde este o algoritmo para a execução deste ponto é invocado

4.6 Distribuição de novas chaves de cifra a partir de chaves pré-distribuídas

pela Alice. Ou seja vai ser pedido ao Bob que envie a sua chave publica, para a Alice assim que esta receber a chave pública vai poder encriptar a sua mensagem, através da função `encrypt()`, que está na classe `RSA`, cujo seu resultado vai ser o criptograma pronto a enviar.

```
ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream((String) in.  
    readObject()));  
  
byte[] criptograma = RSA.encrypt(message, (PublicKey) inputStream.readObject());  
out.writeObject(criptograma);  
out.flush();
```

Excerto de Código 4.7: Trecho de código usado no projeto.

O seguinte trecho corresponde à parte do Bob.

```
RSA.generateKey();  
out.writeObject(RSA.PUBLIC_KEY_FILE);  
out.flush();  
  
criptograma = (byte[]) in.readObject();  
ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(RSA.  
    PRIVATE_KEY_FILE));  
PrivateKey privatekey = (PrivateKey) inputStream.readObject();  
message = RSA.decrypt(criptograma,privatekey);
```

Excerto de Código 4.8: Trecho de código usado no projeto.

Assim que o Bob receber um pedido da chave pública vai a enviar de modo a que esse utilizador consiga avançar no processo e encriptar a mensagem, depois o Bob vai receber o criptograma que é proveniente da Alice ou de outro qualquer utilizador.

4.6 Distribuição de novas chaves de cifra a partir de chaves pré-distribuídas

O trecho de código seguinte mostra um pedaço da função `sendSecret()` correspondente à parte da Alice.

```
// generate new key  
MerklePuzzle mp = new MerklePuzzle();  
String randomKey = mp.random_string(20);
```

4.6 Distribuição de novas chaves de cifra a partir de chaves pré-distribuídas

```
byte[] decodedKey = Client.ins.preDistributedKey.getBytes();
SecretKey originalKey = new SecretKeySpec(decodedKey, 0, 16, "AES");

byte[] encryptedKey = AES.encrypt(randomKey, originalKey);

decodedKey = randomKey.getBytes();
originalKey = new SecretKeySpec(decodedKey, 0, 16, "AES");
byte[] encryptedMessage = AES.encrypt(message, originalKey);

out.writeObject(encryptedKey);
out.flush();
out.writeObject(encryptedMessage);
out.flush();
```

Excerto de Código 4.9: Trecho de código usado no projeto.

O seguinte trecho corresponde à parte do Bob.

```
byte[] encryptedKey = (byte[]) in.readObject();
byte[] encryptedMessage = (byte[]) in.readObject();

byte[] keyByte = Client.ins.preDistributedKey.getBytes();
SecretKey originalKey = new SecretKeySpec(keyByte, 0, 16, "AES");

String aux = AES.decrypt(encryptedKey, originalKey);

keyByte = aux.getBytes();
originalKey = new SecretKeySpec(keyByte, 0, 16, "AES");

message = AES.decrypt(encryptedMessage, originalKey);
```

Excerto de Código 4.10: Trecho de código usado no projeto.

Para a utilização deste ponto, trocamos um segredo a partir de uma chave pré-distribuída em ambos os Clientes. Neste método é gerada uma chave aleatória e encriptada com AES através da chave pré-distribuída e a mensagem original é encriptada usando também AES com a chave aleatória gerada. O Bob vai decifrar a chave aleatória com a chave pré-distribuída e depois decifrar a mensagem original.

4.7 Implementar forma de ter a certeza de que o segredo partilhado é o mesmo dos dois lados

O trecho de código seguinte mostra um pedaço da função `sendSecret()` correspondente à parte onde está o algoritmo para a execução deste ponto é invocado pela Alice.

```
PublicKey pk = EncryptManager.getIns().getDigitalSignature().getMyPk();
out.writeUTF(username);
out.flush();
out.writeObject(pk);
out.flush();

byte[] signedMessage = EncryptManager.getIns().getDigitalSignature().getSignature(
    message);

out.writeObject(signedMessage);
out.flush();
```

Excerto de Código 4.11: Trecho de código usado no projeto.

O seguinte trecho de código corresponde à parte do Bob.

```
String receivedUsername = in.readUTF();
PublicKey pk = (PublicKey) in.readObject();

EncryptManager.getIns().getDigitalSignature().receivedIdentification(receivedUsername, pk
);
byte[] signature = (byte[]) in.readObject();

boolean validated = EncryptManager.getIns().getDigitalSignature().verifySignature(
    receivedUsername, message, signature);
if (!validated) {
    Client.ins.showError("Assinatura invlida. \nO segredo no corresponde ao enviado. \n
        nSegredo recebido: " + message);
    return;
}
```

Excerto de Código 4.12: Trecho de código usado no projeto.

Para verificarmos se o segredo recebido é igual dos dois lados, a Alice envia para o Bob a sua *PublicKey*, a sua Identificação e assinatura correspondente à mensagem original. O Bob irá receber isto e verificar a assinatura com a mensagem obtida depois de decodificada. Para calcular a Assinatura Digital usamos a codificação SHA256withRSA.

4.8 Manual de utilização

Logo após iniciar o programa, será apresentada uma janela perguntando ao utilizador se pretende prosseguir em Modo Servidor ou em Modo Cliente.(Figura 1.3)

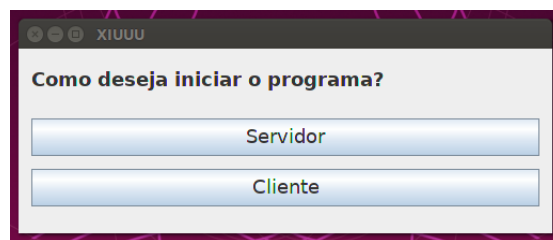


Figura 1.3

Se escolher iniciar como Servidor poderá inserir o número da porta manualmente ou manter o que se encontra por defeito (Figura 1.4).

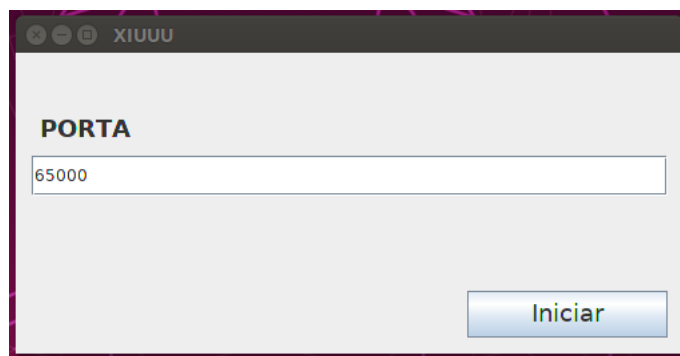
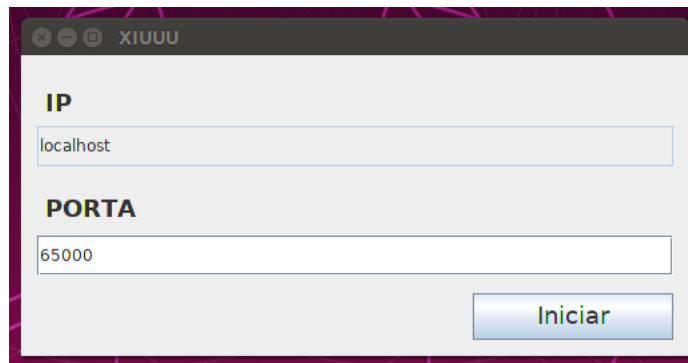


Figura 1.4

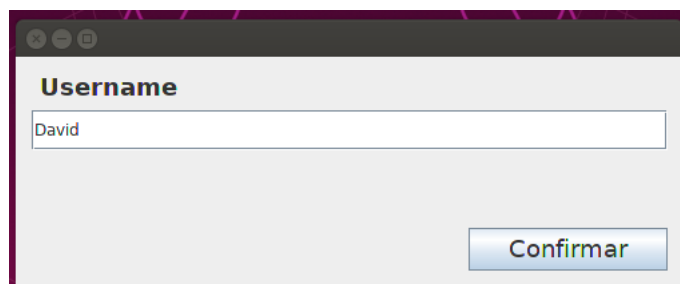
Caso pretenda optar pelo modo cliente o utilizador fará o mesmo processo como para o Servidor pois o IP está predefinido para *localhost* (Figura 1.5).



The screenshot shows a window titled 'XIUUU' with a light gray background. It contains two input fields: one labeled 'IP' with the text 'localhost' and another labeled 'PORTA' with the text '65000'. A blue button labeled 'Iniciar' is positioned at the bottom right of the form area.

Figura 1.5

De seguida, pode inserir o nome de utilizador que pretende (Figura 1.6)



The screenshot shows the same 'XIUUU' window, now with a 'Username' label above a text input field containing the name 'David'. A blue button labeled 'Confirmar' is located at the bottom right.

Figura 1.6

Depois de seleccionar o modo cliente poderá escolher o utilizador para o qual pretende enviar um segredo ou gerar um segredo a partir de PBKDF2 (Figura 1.7).



Figura 1.7

Após escolher gerar um segredo a partir de PBKDF2 o utilizador terá de inserir uma palavra passe, o segredo a enviar e escolher a função de *hash* a utilizar (Figura 1.8).

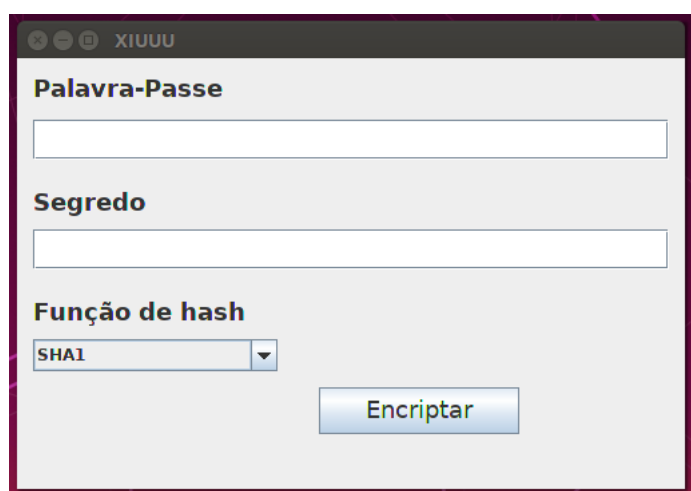


Figura 1.8

Ao clicar em cliente o utilizador poderá escolher a opção de encriptação. As opções disponíveis são: *RSA*, *DiffieHellman*, *MerkelPuzzle*, *PreDistributedKey*

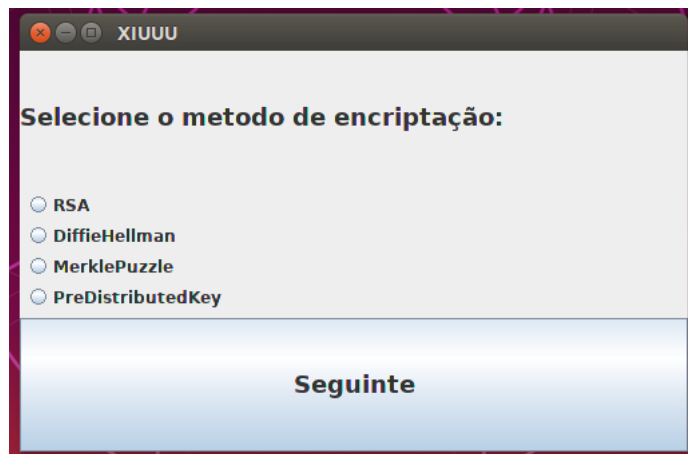


Figura 1.9

Se seleccionar a opção *RSA*, *DiffieHellman* ou *PreDistributedKey* (figura 1.9) será redireccionado para uma interface igual à da figura 1.10. Nesta o utilizador poderá inserir o segredo e enviá-lo.

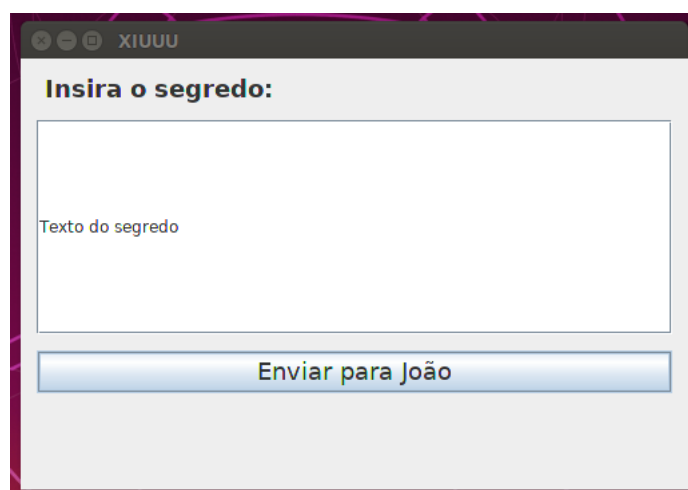


Figura 1.10

Caso o utilizador deseje utilizar o método de encriptação *MerkelPuzzle* terá de inserir o segredo que deseja enviar e o modo de cifra. (Figura 1.11)

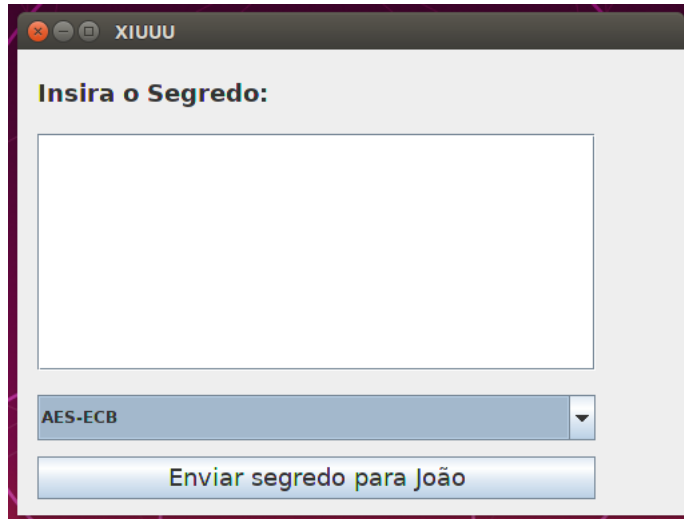


Figura 1.11

Depois de ser enviado o segredo criptográfico a partir do David, o João irá receber a informação numa nova janela com os tipo de encriptação usada, o segredo e o modo de cifra. (Figura 1.12)

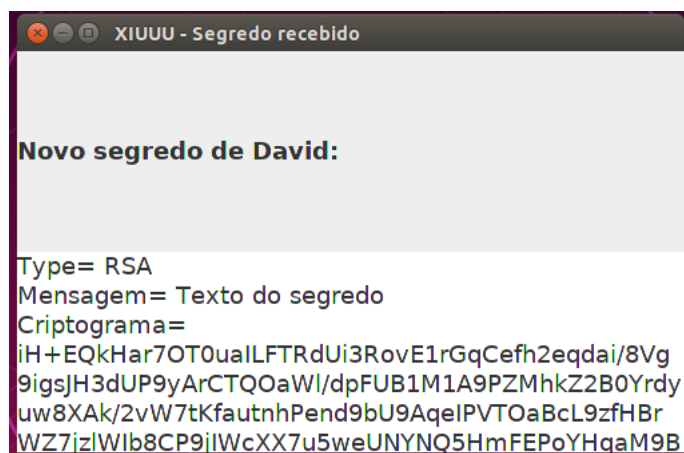


Figura 1.12

4.9 Conclusões

Neste capítulo, estão detalhados e explicados pormenores da implementação de código, bem como justificações de decisões tomadas. Há também um manual

de utilização completo, que tem o objectivo de clarificar ao máximo quem o lê, para que tenha uma melhor percepção de como utilizar a aplicação. No próximo capítulo, vão ser focados alguns problemas encontrados ao longo da construção do projecto e também reflexões sobre o estado final da aplicação.

Capítulo 5

Conclusões e Trabalho Futuro

5.1 Conclusões Principais

A realização desta aplicação foi uma experiência positiva, apesar de não termos conseguido alcançar todos os objetivos pretendidos no enunciado do trabalho. Ao fazer o projeto, adquirimos também mais conhecimento na área da Segurança Informática.

5.2 Trabalho Futuro

Achamos que, no futuro, implementar um sistema de *chat*(com 2 ou mais participantes), ao invés da troca singular de segredos.

Outro objetivo futuro seria fazer com que as comunicações não fossem feitas apenas no mesmo computador mas sim entre 2 ou mais computadores.

Como já foi citado, não conseguimos concluir alguns dos objetivos propostos pelo Professor e por isso outro objetivo seria termina-los. São eles: usar certificados digitais X.509 nas trocas de segredos que recorrem ao RSA, implementar uma infraestrutura de chave pública para o sistema e validar cadeias de certificados nas trocas de segredos que recorrem ao RSA, pensar numa forma correta de fornecer certificados digitais a utilizadores, implementar mecanismos de assinatura digital para verificação de integridade em trocas de chave efémeras usando o *Diffie-Hellman* e distribuição de novas chaves de cifra usando um agente de confiança(neste caso, a aplicação desenvolvida deve permitir que uma das instâncias

possa ser configurada como agente de confiança).

Em suma, demonstramos satisfação com o resultado alcançado e pensamos que, se o desenvolvimento do projeto fosse continuado este teria aplicação no dia à dia.