universidade de aveiro

Departamento de Eletrônica, Telecomunicações e Informática

Segurança
1º Semestre 2016/2017

# Secure Instant Messaging System

Relatório - Projeto Final

Grupo: P3G6

Nuno Silva [72708]
Ricardo Filipe [72727]

Docente: André Zúquete

# Conteúdos


**Introdução e Objetivos** **3**

**Componentes do Sistema** **3**

Servidor 4

Cliente 4

**Tipos de mensagens** **4**

Connect 4

Secure 4

List 5

Client-Connect 5

Client-Com 5

Client-Disconnect 5

**Implementação** **6**

Socket Connection 6

Diffie-Hellman 6

Geração de novo par de chaves 7

Processo de cifragem 7

Mensagens entre Cliente-Servidor e vice-versa 7

Mensagens entre Clientes 7

Processo de decifragem 8

Mensagens entre Cliente-Servidor e vice-versa 8

Mensagens entre Clientes 8

Comandos 8

Funcionalidades do Cartão de Cidadão 9

Entrar no Sistema 9

Ligação entre peers 9

Envio de mensagens 10

Bell-LaPadula 10

**Conclusão** **10**

# 1. Introdução e Objetivos

O trabalho proposto para o projeto da unidade curricular de Segurança foi um sistema de troca de mensagens instantâneas seguro. A implementação deste sistema consiste num server central que interliga os vários clientes que trocam mensagens instantâneas entre si.

Este relatório apresenta todas as fases necessárias ao desenvolvimento deste sistema, mostrando passo-a-passo as diferentes etapas da implementação, de desenvolvimento e os diferentes elementos que compõem o servidor e os clientes .

Os objetivos deste trabalho são acreditar ao sistema um conjunto de componentes que o tornam seguro. A garantia de segurança é dada através de técnicas de criptografia e troca de chaves que impedem que as mensagens entre clientes sejam “escutadas”, introduzidas por terceiros, modificadas ou que sejam sujeitas a qualquer tipo de ataque.

# 2. Componentes do Sistema

O sistema de troca de mensagens instantâneas pode ser visto como uma rede que tem uma topologia centralizada num **servidor** que interliga os **clientes** que queiram estabelecer comunicações entre si.

## 2.1. Servidor

O servidor no sistema é visto como o ponto fulcral onde os clientes de se conectam e todas as mensagens de clientes são processadas e encaminhadas para um cliente.

O servidor contém uma lista dos clientes conectados a si, assim como a respectivo conteúdo criptográfico necessário para interagir com cada um dos seus *peers*.

## 2.2. Cliente

O cliente são nós na rede que primeiramente têm de estabelecer uma conexão segura com o servidor antes de estabelecer qualquer outro tipo de ligação.

Após estarem ligados ao servidor os clientes podem enviar pedidos como obter uma lista de peers ligados ao servidor ou iniciar uma comunicação com outro cliente.

# 3. Tipos de mensagens

Todas as mensagens entre peers e servidor e respostas do servidor aos peers são através de mensagens JSON.

## 3.1. Connect

***Connect*** é a primeira mensagem que o cliente envia ao servidor para estabelecer a sua ligação.

Consiste numa mensagens onde o cliente envia o seu *id*, *nome*, *certificado*, a lista de *cipher-specs* que suporta e ainda um *challenge* que o servidor tem de completar.

Sem uma mensagem ***connect*** inicial é impossível enviar qualquer outro tipo de mensagem seja ela qual seja.

## 3.2. Secure

***Secure*** é usado para encapsular mensagens, com exceção do *connect*, de maneira segura.

A mensagem encapsulada vai encriptada no campo *payload* da mensagem ***secure***.

Todas as mensagens secure têm os campos *hash* e *signature* que permitem que seja possível verificar a integridade e autenticidade, respetivamente, da mensagem.

## 3.3. List

***List*** é um tipo de mensagem encapsulado que o cliente usa para obter uma lista de clientes conectados ao server. É útil para obter o *id* dos clientes para proceder à sua conexão. A lista retornada apenas contém os clientes com os quais o emissor pode iniciar enviar mensagens ***client-com***.

## 3.4. Client-Connect

***Client-Connect*** é um tipo de mensagem encapsulado para estabelecer uma ligação segura entre dois clientes. Consiste numa mensagem que contém o *emissor*, *destinatário*, a lista de *cipher-specs* que o cliente suporta, a sua *chave RSA pública*, *certificado* e ainda um *challenge* que o *peer* tem de completar.

Mais campos são enviados, mas que não têm influência no processo de estabelecimento de uma conexão segura.

## 3.5. Client-Com

***Client-com*** é um tipo de mensagem encapsulado para um cliente enviar a mensagem propriamente dita para outro cliente. Consiste numa mensagem que contém o *emissor*, *destinatário*, o *texto cifrado*, a *chave simétrica* que foi utilizada para cifrar o texto, que por sua vez é cifrada com a chave RSA pública do destinatário, o *iv* e ainda o *unique device id* que permite ao destinatário saber se a mensagem foi enviada que um dispositivo diferente.

Esta mensagem contém também os campos *hash* e *signature* que permitem que seja possível verificar a integridade e autenticidade, respetivamente, da mensagem ao nivel do cliente.

## 3.6. Client-Disconnect

***Client-Disconnect*** é um tipo de mensagem encapsulado usado para terminar a ligação entre dois clientes. Os campos desta mensagem incluem uma o remetente e o destinatário. Esta mensagem contém também os campos *hash* e *signature* que permitem que seja possível verificar a integridade e autenticidade, respetivamente, da mensagem ao nivel do cliente.

## 3.7. ACK

***ACK*** é um tipo de mensagem especial pois pode ser encapsulado ou não.

Este tipo de mensagens é utilizado para avisar o cliente que a mensagem que o cliente enviou foi recebida com sucesso pelo servidor ou pelo *peer.* Após se estabelecer uma ligação segura com o servidor, os ack são encapsulados em mensagens ***secure***. Todas as mensagem ***ack*** contêm o campo hash que

identifica inequivocamente a mensagem à qual é referido. Os ***ack*** que são destinados a outro cliente contém também os campos *src* e *dst* para se permitir identificar o remetente e o destinatário da mensagem. Depois de se estabelecer uma ligação segura entre dois clientes, os ***ack*** passam a conter também um *hash* e *signature* para permitir que o destinatário averigue a integridade e autenticidade do ***ack***.

# 4. Implementação

Durante o desenvolvimento do sistema foram usadas libraries como cryptography, socket, json, base64, openssl, pykcs11, entre outras. Para simplificar algumas operações como cifragem e decifragem de conteúdos, bem com processamento de informação guardada no cartão de cidadão criámos um conjunto de módulos num ficheiro *utils.py* que se baseou nas primitivas das bibliotecas cryptography, openssl, pykcs11, entre outros.

Segue-se como foi implementado o sistema e como funcionam alguns processos fulcrais:

## 4.1. Socket Connection

Quando o servidor inicia, estabelece uma sessão TCP/IP através de sockets por onde possam ser transportadas mensagens.

## 4.2. Diffie-Hellman

Quando um cliente pretende conectar-se ao servidor envia uma mensagem tipo ***connect***. O cliente do seu lado possui um par de chaves pública e privada onde transmite ao servidor o seu valor público. O servidor recebe o valor público do cliente e com a sua chave privada gera um *shared secret*. Ao enviar o seu valor público ao cliente o cliente pode gerar o *shared secret* com a sua chave privada.

Durante este “handshake”, existe um acordo entre algoritmos de cifragem. O servidor dispõe de algoritmos de cifragem que pretende usar com uma ordem de preferência. Quando o cliente pretende conectar-se ao servidor propõe ao servidor os algoritmos que pretende usar.

Se houver uma discordância entre o servidor e cliente, isto é, se os algoritmos que o cliente propôs não se encontram na lista de algoritmos que o servidor pretende usar, então a ligação é interrompida.

## 4.3. Geração de novo par de chaves

Um dos problemas que tínhamos no primeiro *milestone* é que não era permitido um cliente, ou servidor, enviar duas mensagens secure sem receber uma entretanto pois criava uma dessincronização das chaves.

Para resolvermos este problema, ao enviar uma nova mensagem, cria-se um novo par de chaves temporário, não alterando os acordados no connect, e gera-se um novo segredo:

$$Sh = Priv_{tmp} + Pub_{ConnectionB}$$

A mensagem é cifrada com este novo segredo e é enviada para o servidor, ou cliente, com a chave pública temporária.

## 4.4. Processo de cifragem

Os processos de cifragem ocorrem em dois instantes: cifragem de mensagens entre cliente-servidor e cifragem do campo ***text*** entre clientes:

### 4.4.1. Mensagens entre Cliente-Servidor e vice-versa

Depois de se estabelecer uma conexão segura entre o cliente e servidor através do processo Diffie-Hellman, cliente e servidor podem trocar mensagens encriptadas usando o método escolhido no processo de conexão.

A cada nova mensagem a ser enviada, a entidade que está a enviar a mensagem gera um novo par de chaves temporário e utiliza a chave pública que conhece do parceiro, acordada no processo de conexão, ***connect*** para gerar um novo segredo.

$$Sh = Priv_{tmp} + Pub_{ConnectionB}$$

A mensagem é então cifrada com o novo segredo e é enviada para o recetor juntamente com o chave pública temporária gerada.

### 4.4.2. Mensagens entre Clientes

Para as mensagens entre clientes estas são cifradas com uma chave gerada no processo de conexão. Após ser acordado o algoritmo de cifragem a usar, quem inicia a conexão gera uma chave simétrica, e encripta-a com a chave

pública RSA do cliente com o qual quer fazer a conexão. Este cliente ao receber a chave encriptada decripta-a com a sua chave privada. Assim, ambos os clientes obtêm o mesmo segredo sem que este possa ser descoberto pelo meio.

A cada nova mensagem client-connect, mensagem que requer encriptação do texto, é gerada uma nova chave simétrica que é encriptada com a chave pública RSA do destinatário.

## 4.5. Processo de decifragem

Os processos de decifragem ocorrem em dois instantes: decifragem de mensagens entre cliente-servidor e decifragem de mensagens entre clientes:

### 4.5.1. Mensagens entre Cliente-Servidor e vice-versa

Ao ser recebida uma mensagem ***secure***, quem a vai decifrar (cliente ou servidor) necessita primeiro de saber com que chave é que ela foi cifrada.

Adjacente à mensagem vem uma chave pública de quem a enviou, que foi usada para gerar o texto encriptado. O recetor da mensagem aplica então a função para obter o segredo comum.

$$Sh = Priv_{ConnectionB} + Pub_{tmpA}$$

Ao obter a chave com que foi cifrada a mensagem, basta decriptá-la usando o algoritmo acordado na fase de conexão.

### 4.5.2. Mensagens entre Clientes

O processo de decifragem nas mensagens entre clientes segue o mesmo princípio que as mensagens entre cliente e servidor.

Juntamente com a mensagem, é enviada a chave que foi utilizada para encriptar a mensagem, que a destinatário tem de decriptar com a sua chave privada RSA para posteriormente poder decriptar o texto.

## 4.6. Comandos

Para facilitar a utilização do programa segue em lista os comandos suportados e uma breve descrição dos mesmos:

- ***client-connect <id>*** - Cria uma ligação segura com o cliente com <id>
- ***list*** - Recebe uma listagem de clientes conectados ao servidor

- ***client-com <id> <message>*** - Envia uma mensagem para o cliente com <id> . Apenas é permitida depois de haver uma conexão segura com o cliente.
- ***client-disconnect <id>*** - Termina uma ligação segura com o cliente <id>
- ***peerlist*** - Mostra todos os clientes com os quais tenho uma conexão segura.
- ***messages*** - Mostra a lista de mensagens que foram enviadas e que ainda não foi recebido um ***ack***.

## 4.4 Funcionalidades do Cartão de Cidadão

O sistema está preparado para ser utilizado pelo cartão de cidadão. O CC é útil ao nível da autenticação e da validação das mensagens pois é possível utilizar os seus certificados e chaves e identifica inequivocamente o utilizador.

### 4.4.1. Entrar no Sistema

Quando se inicia um *client* este lê o Cartão de Cidadão a partir de um leitor de cartões ligado ao computador, acedendo à informação disponível no chip do cartão. É pedido ao utilizador que introduza o seu PIN de autenticação a partir do middleware do Cartão de Cidadão.

Após introduzir corretamente o PIN o client está ligado ao sistema (mas não conectado ao servidor). O **id** atribuído ao client é igual ao número do seu CC assegurando preservação de identidade. Desta forma existe uma associação direta entre um utilizador e um cidadão português usando o Cartão de Cidadão.

### 4.4.2. Certificados

O sistema utiliza os certificados presentes no cartão de cidadão para verificar integridade do utilizador. Tanto o servidor como o cliente possuem uma *store* para validar os certificados que recebem. Esta store contém os certificados responsáveis pela emissão dos certificados existentes nos cartões de cidadão, bem como o certificado emissor do servidor. Nos processos de estabelecimento de ligações seguras, os certificados são testados a ver se são confiáveis. Esta validação não é feita apenas com a store mas também acedendo à entidade emissora para averiguar se o certificado não foi revogado. Como a qualquer momento o certificado pode ser revogado, o servidor testa a validade de uma mensagem recebida de um cliente a cada dois minutos.

### 4.4.3. Ligação entre peers

Quando um client pretende conectar-se a outro peer (servidor ou outro client) começa por enviar um *challenge* e o seu certificado (do Cartão de Cidadão). Do outro lado, o receptor valida o certificado e assina o *challenge*.

Após isto, ele cria e envia um *challenge* e o seu certificado (também do Cartão de Cidadão). O client verifica se o seu *challenge* foi respondido corretamente, assina o *challenge* e valida o certificado que recebeu. Do outro lado é verificado se o *challenge* foi respondido de forma correta.

Se  o processo falha na conexão o cliente termina imediatamente, pois o sistema foi comprometido.

### 4.4.4. Envio de mensagens

No envio de mensagens, quando uma mensagem é enviada esta gera uma *hash* que verifica a sua integridade que posteriormente é assinada pela chave privada do Cartão de Cidadão.

## 4.5 Bell-LaPadula

Foi implementado controlo sobre o fluxo de informação, atribuindo um nível de segurança a cada utilizador.

O conceito defende que um utilizador pode receber mensagens de outro utilizador num nível igual ou inferior ao dele, e pode enviar mensagens a outro utilizador num nível igual ou superior ao dele. O servidor contém uma base de dados, para os efeito de demonstração é utilizado um dicionário, que tem uma correspondência direta entre cada utilizador e o seu nível de acesso.

## 4.6 Device Unique ID

Uma das *features* do sistema é a possibilidade de um cliente verificar se a mensagem que o cliente lhe envia origina numa máquina diferente. Isto é possível, gerando um device unique id a quando a aplicação é iniciada. Este *uuid* é gerado fazendo um *hash* de várias informações do sistema como por exemplo: mac address, processor name, entre outros parâmetros.

Este *uuid* é enviado em todas as mensagens ***client-com***.

# 5. Conclusão

A implementação do projeto foi desafiadora mas contudo ofereceu uma oportunidade de aprofundar o conhecimento de maneira mais concisa sobre a matéria lecionada nas aulas teóricas e práticas.

O tema em si é interessante pois aborda tecnologias que usamos todos os dias sem o utilizados comum se aperceber. Tecnologias essas que envolvem o desenvolvimento de sistemas de trocas de mensagens e de sistemas seguros.

Ao longo do desenvolvimento do projeto foram encontrados obstáculos sobre temas que já nos eram conhecidos e outros temas que desconheciamos. A aprendizagem sobre bases de certos temas que nos eram pouco familiares ajudou ao nosso crescimento como futuro engenheiros.

# Referências


- Criptography python library - *https://cryptography.io/*
- Segurança em Redes Informáticas, André Zúquete, 4º Ed. Aumentada, FCA - Editora de informática, Lda.