universidade de aveiro

Departamento de Electrónica, Telecomunicações e Informática

Segurança - Turma P1G7

Secure Instant Messaging System

Mestrado Integrado em Engenharia de Computadores e Telemática

Pedro Ferreira de Matos - **71902**
Tiago Alexandre Lucas de Bastos - **71770**

Docentes:

- Professor João Paulo Barraca
- Professor André Zúquete

Índice

1 Introdução

O trabalho proposto para o projeto da unidade curricular de Segurança é a criação de um sistema de troca de mensagens instantâneas entre utilizadores, em que o projeto final contenha clientes a trocarem mensagens através de canais seguros mediados por um servidor.

O objetivo deste sistema é garantir a máxima confidencialidade e integridade do serviço visto que a probabilidade de ocorrerem ataques é muito elevada. Estes ataques podem ser direcionados à obtenção de compensação monetária ou de provocar danos nos sistemas. Para isso são necessários vários processos como por exemplo, a cifragem de todo o material existente, derivação de chaves e autenticidade dos utilizadores com recurso ao cartão de cidadão.

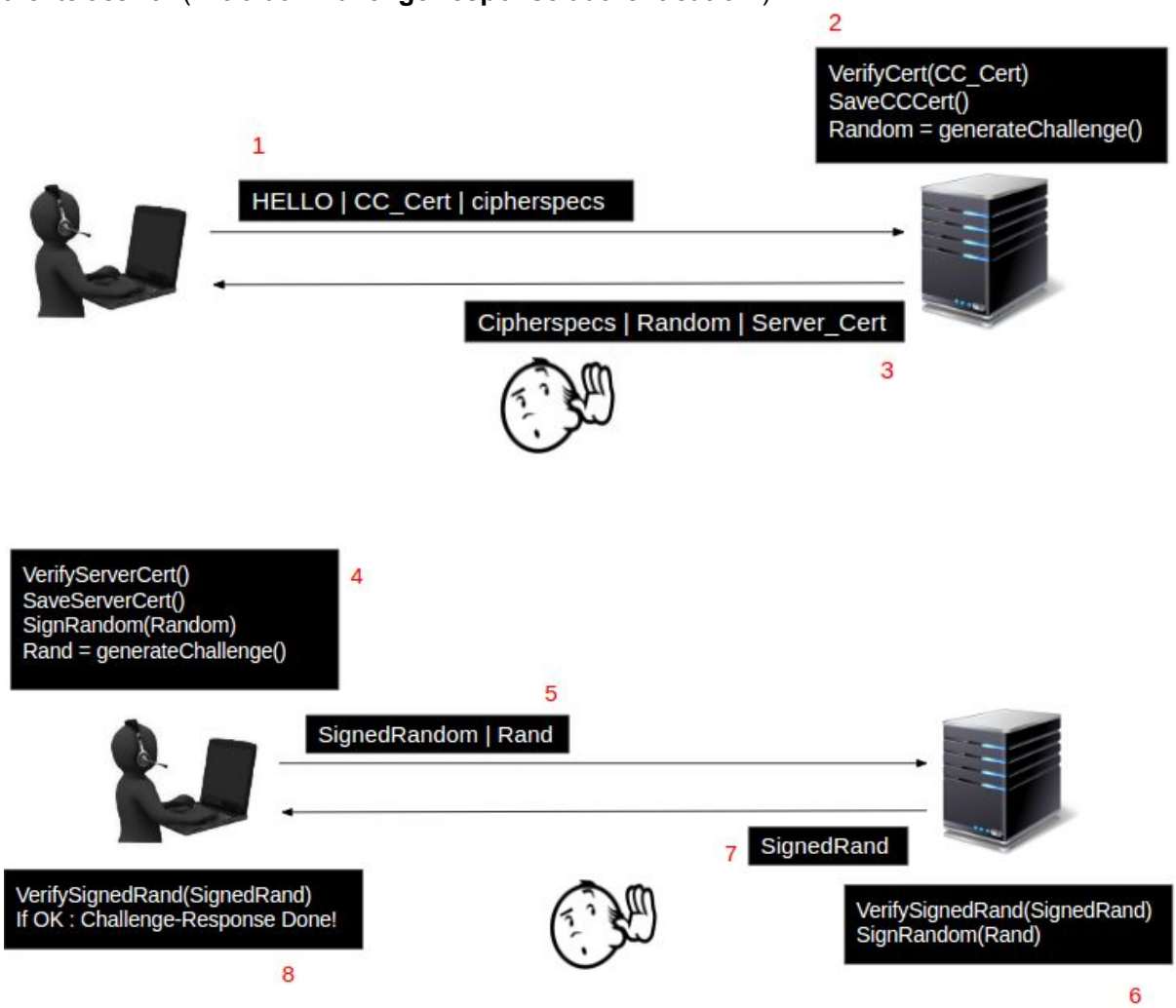
O relatório reflete todos os passos e decisões tomadas na criação do sistema, tecnologias utilizadas, descrição dos vários processos existentes e conclusão.

2 Alterações aos Processos da 1ª entrega

Para esta entrega tivemos de fazer pequenas alterações às fases de negociação porque é necessário introduzir o processo de **challenge-response authentication,** autenticação e leitura dos certificados do CC e do servidor. Também foi introduzida a assinatura de todas as mensagens e respectiva validação através de *Ack's* enviados tanto do lado dos Clientes como do Servidor.

2.1 Processo de HandShake entre Cliente-Servidor

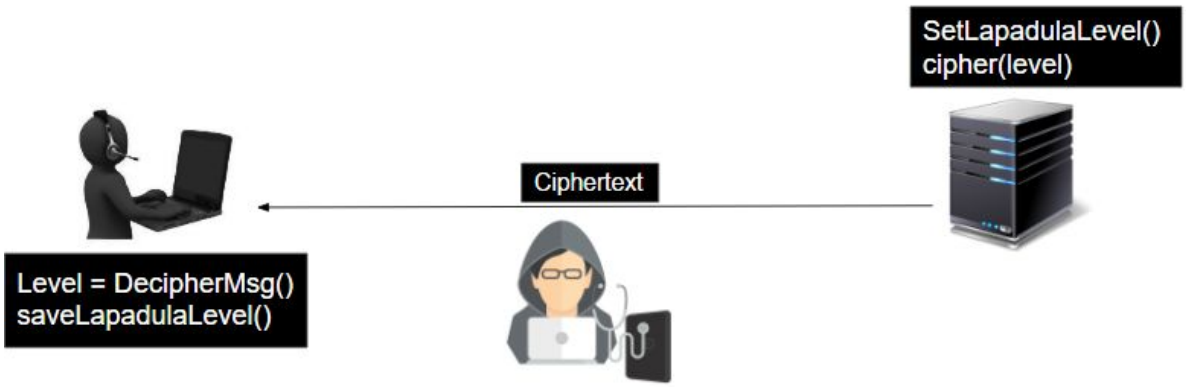
Para estabelecermos uma ligação com o servidor vamos ter de enviar o certificado do nosso cartão de cidadão, para que o servidor verifique se este é válido. Esta primeira mensagem funciona como um *"Client Hello"*, que é a primeira mensagem utilizada no modelo TLS. Caso o servidor confirme a validação do certificado, vai enviar o seu certificado, o qual o cliente também tem de verificar (há uma inversão de papéis), e envia ainda um random para o cliente assinar (início do **Challenge-response authentication**).



Após a verificação/validação dos certificados, procede-se a uma troca de randoms assinados(**Challenge-response authentication**). O servidor envia o primeiro random para o cliente, e o cliente tem de o assinar com a sua chave privada (presente no cartão de cidadão). O processo inverte-se e o servidor também tem de assinar o random que o cliente lhe envia, desta vez com a sua chave privada (gerada pelo xca). **Após esta fase ser concluída, todas as mensagens passam a ser assinadas com a chave privada de cada entidade**. Desta forma, o nosso processo de HandShake entre o Cliente-Servidor segue o modelo de TLS.

Para finalização, o cliente recebe o seu nível do modelo Bell-Lapadula para saber o seu nível de acesso, a implementação em si do Bell-Lapadula será explicada na parte das features de segurança.

O envio de acks assinados vai ser explicado mais à frente, tal como a implementação do Bell-Lapadula, num tópico próprio dos acks.



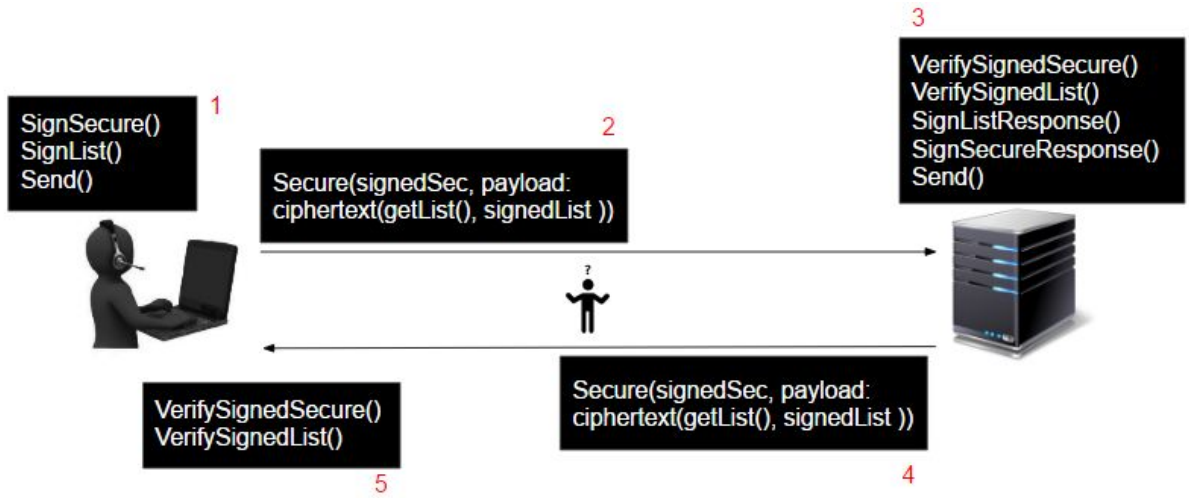
Após esta última mensagem em que o Cliente recebe o seu nível, o processo de conexão fica estabelecido. No entanto, quando ocorre uma regeneração de chaves por *time-out* ou por ter sido excedido o número de mensagens limite por conexão, o processo é efectuado da mesma maneira, com um novo Challenge-Response Authentication.

Estas foram as alterações que fizemos ao processo de Conexão entre Cliente e Servidor, a implementação de como é feito o acordo de chaves, e como era verificado se esse acordo de chaves era bem feito foi explicado na 1ª parte do relatório(DH elliptic curve).

2.2 Processo de Mensagens Secure

2.2.1 Mensagens List

Não ocorre nenhuma modificação estrutural neste tipo de mensagens relativamente à 1ª entrega. A ligação secure já está estabelecida entre o cliente e servidor, por isso, tanto o pedido list que o Cliente faz ao Servidor, como o secure em si, são **assinados** pelo cliente. O Servidor faz a verificação dessa assinatura, e responde com um secure e a mensagem do tipo list assinadas, sendo que o Cliente verifica estas assinaturas ao receber a mensagem.

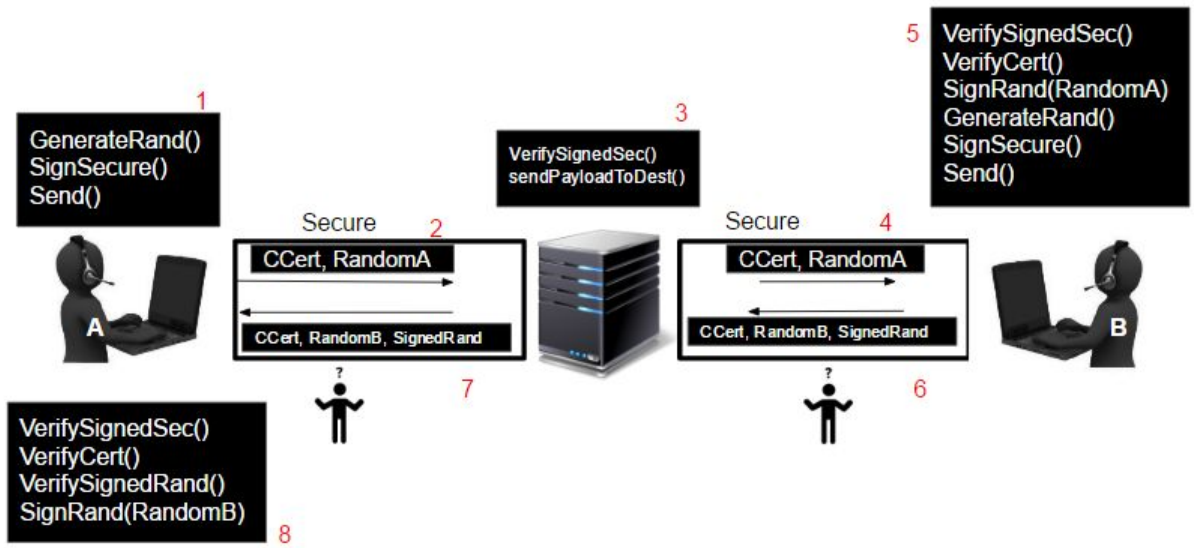


O Processo de Cifra/Decifra deste tipo de mensagens já tinha sido, explicado na 1ª parte do trabalho, estando só ilustrado na imagem como é feita a assinatura e verificação das mensagens Secure(List).

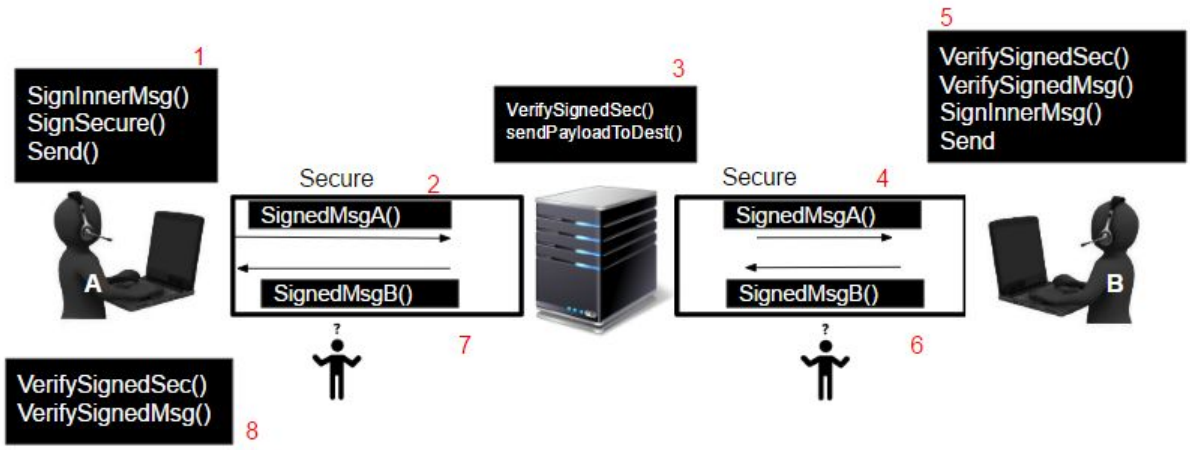
Tal como referenciamos na processo de ligação entre o cliente e servidor, o processo de recepção de acks deste tipo de mensagens vai ser explicado mais à frente.

2.2.2 Mensagens Client-Connect - Handshake Client-Client

Tal como o processo de ligação entre o cliente e servidor, as fases tiveram de ser ligeiramente modificadas. A 1º fase continua a ser o envio de uma mensagem secure para o servidor, em que dizemos qual o cipher spec que pretendemos usar com o cliente a qual nos queremos conectar. Após recebermos a resposta do servidor sobre a validade do cipher spec, enviamos o nosso certificado e um random para o cliente ao qual nos queremos conectar. O cliente ao receber esta mensagem, verifica a validade do certificado do cliente que se quer ligar a ele, e caso o certificado seja válido, assina o random e envia juntamente com o seu próprio certificado e um random para o outro cliente também assinar. O cliente que iniciou o handshake verifica o certificado que recebeu e o random que pediu para ser assinado, e assina também o random dado pelo outro cliente. O **Challenge-response authentication** também é realizado entre clientes.



Caso tudo esteja bem, a troca procede normalmente mas **todas as mensagens a partir deste momento vão assinadas**. Cada cliente tem de assinar o secure para o servidor e a mensagem para o outro cliente, para que o servidor saiba quem está a assinar, e outro cliente a mesma coisa(autenticidade). Ou seja, cada cliente vai assinar duas mensagens, cada uma para uma fonte diferente(servidor e cliente ao qual se quer ligar), e cada cliente que recebe uma mensagem tem de verificar se a assinatura do secure corresponde com a do servidor, e se a assinatura da mensagem privada provém do cliente certo.

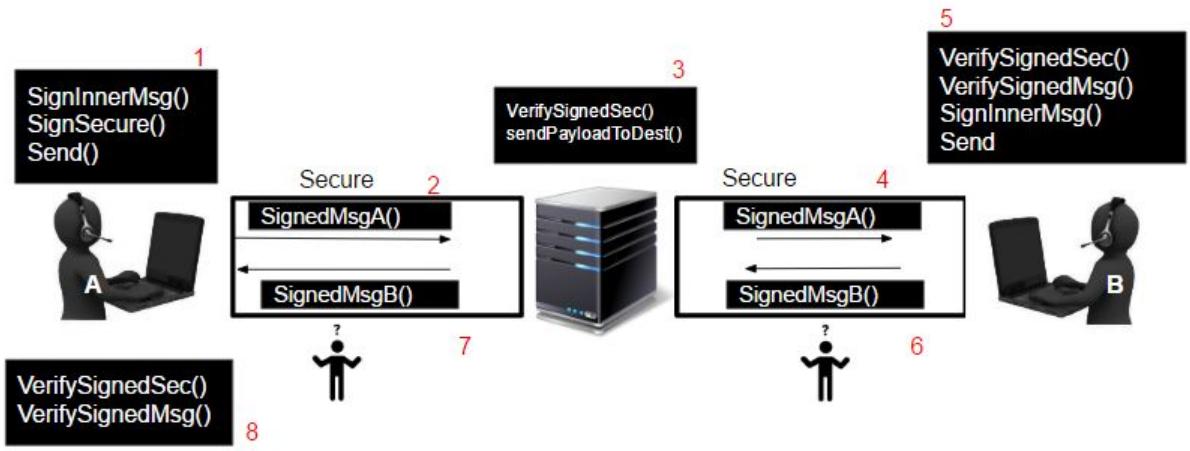


Desta forma, garantimos a integridade total das mensagens (lembrando que na primeira fase estas já eram cifradas com o segredo gerado do diffie-hellman elliptic curve) e a autenticidade das mesmas porque ao assinarmos com a chave privada do cartão de cidadão estamos a garantir que a mensagem veio mesmo do cliente X e não do cliente Y.

2.2.3 Mensagens Client-Com

As mensagens de texto entre dois clientes não sofreram nenhuma alteração estrutural nas fases. Cada mensagem é enviada dentro de um secure para o servidor, que depois encaminha a mensagem original para o outro cliente, dentro de um secure também.

A mensagem do tipo Client-Com vai **assinada** para o cliente destino, de forma a que este tenha a certeza de que esta foi mesmo originada pelo cliente que a manda. A mensagem secure que encapsula o cliente-com vai **assinada** para o servidor, pois este também tem de verificar se o secure provém mesmo do cliente que o envia.



Quando um cliente recebe um client-com **verifica a assinatura** desta mensagem para ter a certeza que ela foi **assinada** pelo cliente que a envia. Se este teste der positivo, tenta decifrar a mensagem enviada, e caso a decifra também seja bem sucedida, imprime a mensagem privada do cliente que enviou no client-com.

2.2.4 Mensagens Client-Disconnect

O cliente que se pretende desconectar envia uma mensagem do tipo client-disconnect que vai **assinada** para o cliente destino, e **assina** também a mensagem secure que o servidor recebe. O cliente destino ao receber uma mensagem deste tipo, verifica a assinatura do cliente que enviou a mensagem. Caso a assinatura esteja correta, procedem-se as fases de desconexão, em que cada mensagem também vai assinada.

2.2.5 Mensagens ACK

Nesta segunda iteração do projeto era essencial garantir a integridade total e autenticidade das mensagens. Isto só é possível ao termos a certeza que cada mensagem que recebemos vem, sem qualquer dúvida, da pessoa que diz que a envia.

Para tal podemos usar a assinatura da chave privada do cartão de cidadão na assinatura das mensagens. Desta forma conseguimos garantir que aquela mensagem foi realmente assinada com o certificado do cliente que se conectou com o mesmo certificado.

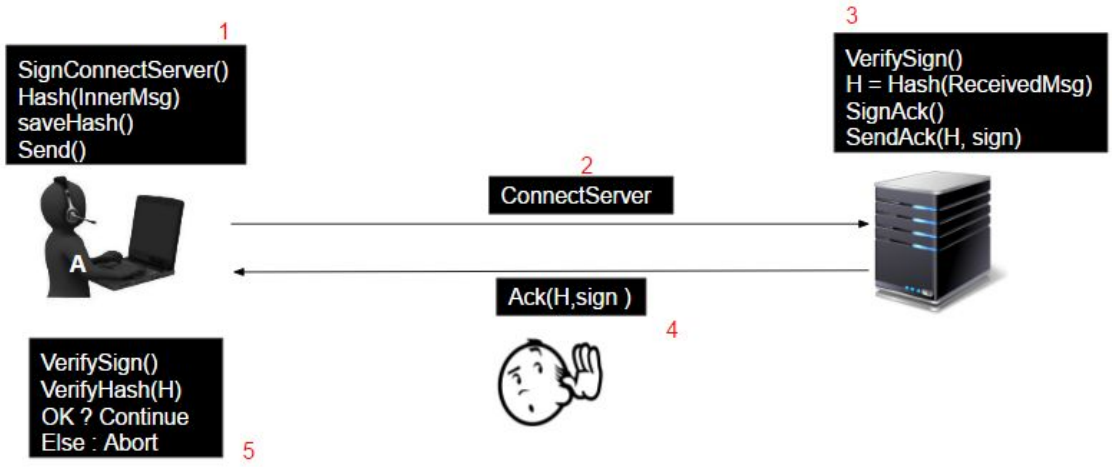
No entanto, ao enviarmos uma mensagem, nada nos garante que esta chegue realmente ao destino. É para garantir a entrega de cada mensagem que os ACKs são importantes. Dentro dos Ack's encontra-se sempre um **Hash (SHA-1)**, esse Hash corresponde ao **Hash da mensagem recebida**, o cliente de destino envia o Ack assinado com o Hash da mensagem que recebeu, e ao receber esse Ack, **o cliente de origem compara o Hash do Ack recebido com o Hash da mensagem que enviou originalmente**, para garantir a integridade da mensagem que originalmente enviou. Deste modo é possível garantir, que o cliente de destino recebeu mesmo a mensagem que foi enviada.

2.2.5.1 Mensagens acks entre Cliente e Servidor (fase Server HandShake)

O cliente e servidor não trocam mensagens do tipo **secure** enquanto não tiverem gerado o segredo para a sua troca de mensagens. Enquanto fazemos estas mensagens também temos de garantir o envio das mesmas através de acks, mas como ainda não há um segredo acordado entre ambos não podemos enviar os **acks** dentro de uma mensagem do tipo **secure**. Os acks são enviados apenas como tipo **ack** e sem qualquer encapsulamento.

No entanto, assim que o processo de **Challenge-response authentication** é concluído estes acks também são assinados para que a sua autenticidade seja garantida. Antes da conclusão do **Challenge-response authentication,** são enviados Ack's na mesma, mas sem estarem assinados.

O esquema ilustra o processo do envio de Ack's entre cliente e servidor após o **Challenge-response authentication**, a imagem apenas ilustra numa direção, mas por cada mensagem do servidor, o cliente **faz exatamente o mesmo processo que o servidor.**



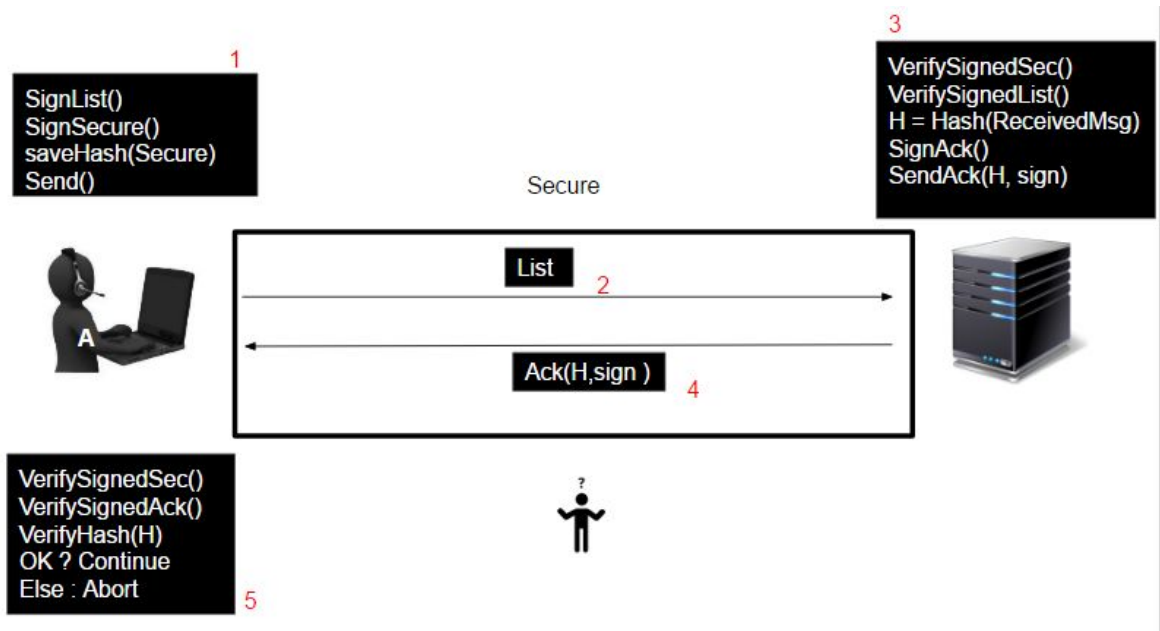
Como podemos ver no esquema, a entidade que originou a mensagem, após receber o Ack, compara o Hash recebido com o Hash que se encontra guardado da mensagem de origem, se a verificação do Hash der errada, quer dizer que o destino recebeu uma mensagem adulterada pelo caminho.

2.2.5.2 Mensagens acks-secure entre Cliente e Servidor

Entre cliente e servidor apenas é garantido a entrega da mensagem do tipo secure, por isso apenas há o envio de um ACK encapsulado dentro de uma mensagem secure.

Este ack é enviado após a recepção de cada mensagem do tipo **secure** para que o servidor/cliente tenha sempre a garantia que as mensagens que trocam entre si são sempre entregues, tal como foi explicado anteriormente, este Ack contém um Hash da mensagem anteriormente recebida, para o cliente que originou poder verificar se o destino recebeu a mensagem que realmente enviou.

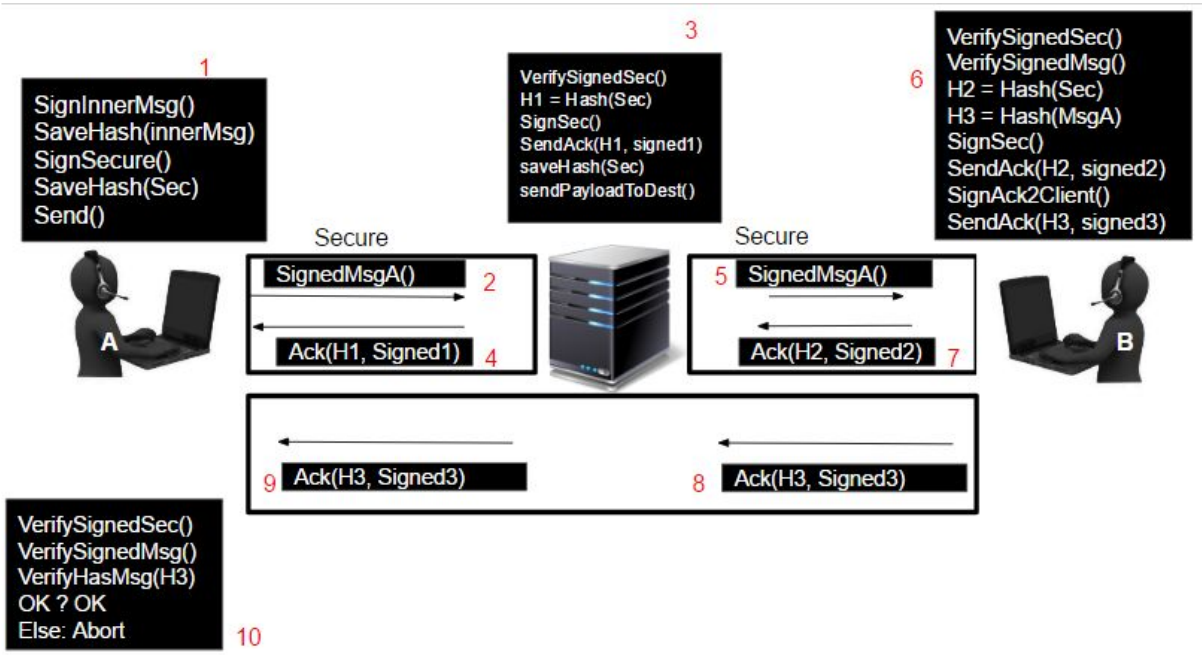
Um exemplo deste tipo de ACKs pode ser o processo de quando um cliente faz um pedido List ao Servidor, encontra-se ilustrado no esquema abaixo.



2.2.5.3 Mensagens acks-secure entre Cliente e Cliente

Ao haver troca de mensagens do tipo **client-connect**, **client-com** e **client-disconnect** temos de garantir que a mensagem encapsulada chega ao outro cliente, visto que os acks das mensagens **secure** já foram tratados. Para tal, ao recebermos cada mensagem de outro cliente temos de enviar um **ack** dentro de um secure para o cliente que nos enviou a mensagem. Para isso criamos um novo tipo de ack(**ack-connect-dst**) que é assinado e cifrado pelo cliente e vai encapsulado dentro de um **secure** até ao servidor, sendo que este ao ver o tipo de ack encaminha-o para o cliente de destino. Desta forma é possível os clientes saberem que as suas mensagens são entregue ao cliente com o qual estão a iniciar uma ligação e ao trocarem mensagens.

O esquema abaixo, pode ser complicado de perceber, mas o que se passa é simples: Quando um cliente (A, Origem) envia uma mensagem para um outro Cliente (B, destino), ele vai gerar um *Hash* do **Secure** e da Mensagem **interior** (com, connect, disconnect), quando o servidor recebe essa mensagem, verifica as assinaturas como sempre faz, e gera um hash da mensagem recebida, esse hash (H1), é enviado num Ack ao cliente de Origem (A). O Servidor posteriormente encaminha a mensagem para o cliente de destino (B), e esse cliente vai gerar 2 Hash's, um que vai no Ack só para o servidor (H2) e outro que vai num Ack que tem de chegar ao outro **cliente**, **passando pelo servidor(H3)**. O servidor verifica o Hash Recebido do Cliente de destino, e o Cliente de Origem verifica o Hash que o Cliente de destino lhe enviou no Ack(H3). Se todos os Hash's coincidirem com os que ficaram guardados garantimos que tanto os Secures como o seu payload (mensagem que tem de chegar ao outro cliente) nunca foram alterados.



2.3 Processo de Desconexão entre Cliente-Servidor

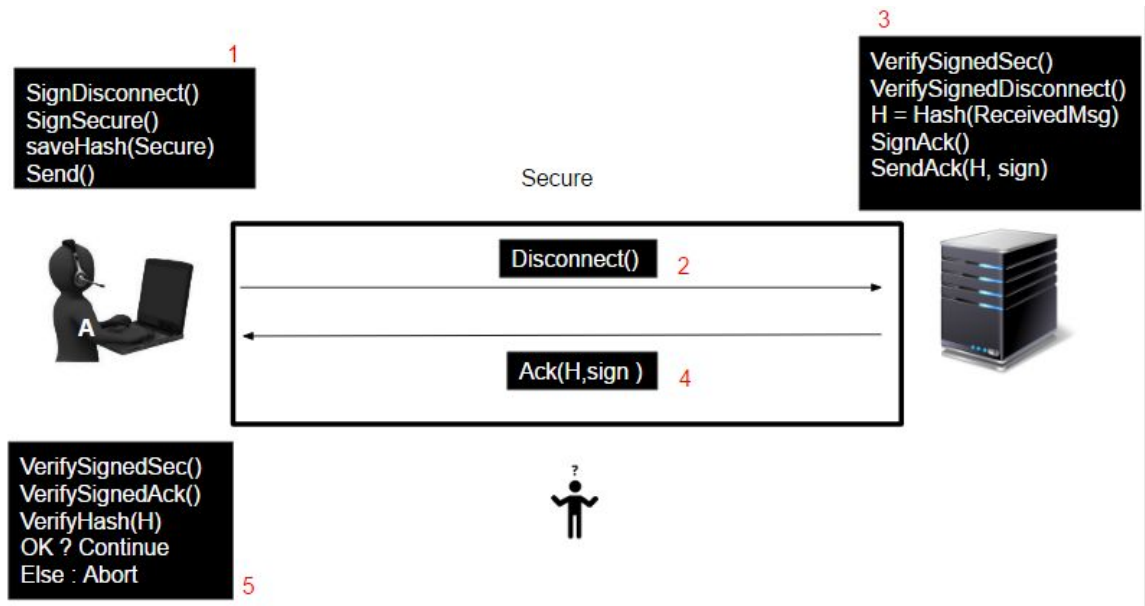
Quando há um pedido de desconexão do Cliente para o Servidor, o Cliente faz uma verificação prévia por clientes a que está conectado e desconecta-se deles, enviando uma mensagem para que estes apaguem a informação relativa a ele.

2.3.1 Mensagens ACK

Como o cliente também se vai desconectar de outros clientes, vão haver acks assinados e encapsulados entre o disconnect do cliente que inicia o processo e todos os outros clientes que têm uma conexão ativa com ele, e acks encapsulados para o processo de desconexão entre ele e o servidor. Todos estes acks são assinados com a chave privada do CC e vão dentro de mensagens secure. Os Acks também possuem uma Hash da mensagem recebida para garantir poder garantir a 100% a integridade da mensagem.

2.3.2 Mensagens assinadas

As mensagens **secure** trocadas com o servidor são sempre assinadas, e o mesmo acontece com as mensagens do tipo **client-disconnect.** Como já existe uma ligação segura entre o cliente e servidor, e entre o cliente e os restantes clientes a ele ligados, todas as mensagens têm de ir assinada. Após isto ser verificado, cada cliente pode se desconectar daquele que lhe enviou a mensagem.



O esquema acima ilustra as explicações dos ack's e das assinatura neste tipo de mensagens, os processos internos de desconexão foram explicados no relatório da 1ª fase do projeto.

2.4 Extra: Participant Consistency

Para a garantia desta funcionalidade, era pedido que um cliente conseguisse saber sempre que uma mensagem fosse originada de um cliente que estivesse numa máquina diferente da última em que enviou mensagens.

Na nossa implementação, impedimos que um utilizador se conecte ao servidor sem que faça logout, guardando o certificado de cada utilizador, e comparando estes com o de cada cliente que se tenta ligar ao servidor. Caso o certificado dele já esteja em uso, impedimos a ligação. Desta forma, é impossível que tenhamos dois clientes originários do mesmo cartão de cidadão. Isto é muito importante para o **participant consistency** porque desta forma apenas precisamos de verificar em que máquina o cliente se encontra no processo de **client-connect**, visto que todas as mensagens têm de vir dessa máquina, pois o mesmo cliente não pode estar conectado duas ou mais vezes ao mesmo tempo.

Em cada processo **client-connect**, ao fim do segredo estar gerado entre ambos, é enviado um hash com os specs imutáveis da máquina em que o cliente está(mac address, Sistema Operativo e informação do processador). O cliente verifica se o cliente com aquele certificado já se ligou alguma vez, e caso já tenha, verifica o hash que recebeu com o que já tinha guardado, para saber se o cliente mudou de máquina ou não. Caso tenha mudado, envia uma mensagem de aviso e imprime o novo hardware e informação da máquina em que o cliente se encontra agora. Essa informação também é guardada para que num próximo **client-connect** consiga saber se houve uma nova mudança de máquina por parte do cliente.

Referências para o extra:

- http://stackoverflow.com/questions/159137/getting-mac-address
- http://stackoverflow.com/questions/3103178/how-to-get-the-system-info-with-python
- https://pypi.python.org/pypi/py-cpuinfo

3 Features de Segurança

Na segunda fase do projeto foram implementadas as seguintes features de segurança. As restantes features já tinham sido explicadas no relatório da 1ª fase (**Multiple Cipher Support**, **Forward Secrecy** & **Backward Secrecy**, **Integridade do Ciphertext**), apenas ficando por implementar o último extra **(Conversation consistency)**.

3.1 Message Confidentiality, Integrity & Authentication

Agora é possível garantir a 100% a confidencialidade, integridade e autenticação de cada mensagem.

A **confidencialidade** está relacionada com a cifra da mensagem e do encapsulamento destas num Secure, deste modo asseguramos que alguém que consiga capturar uma mensagem de fora, não a consiga decifrar sem ter um segredo partilhado entre ambas as entidades, segredo esse que fica sempre guardado em cada cliente.

A **integridade** é garantida através dos Hash's gerados sobre cada mensagem na origem. O destino ao receber a mensagem, gera um hash da mensagem recebida e envia esse hash no Ack. Caso a Origem verifique que o hash guardado bate certo com o hash recebido no Ack, então podemos assegurar que a mensagem chegou íntegra ao destino tal como foi enviada. Caso seja verificado que alguma das mensagens foi alterada, ou seja, os hash's não batem certo, a ligação entre ambas as partes é abortada.

A **autenticação** é garantida através da assinatura das mensagens com as chaves privadas de cada entidade. A chave privada de cada cliente está guardada no cartão de cidadão, e o servidor verifica esta assinatura através da chave pública que veio dentro do certificado que o cliente enviou no início da conexão. A chave privada do servidor é uma chave privada RSA que foi gerada no programa **xca**, tal como o seu certificado. Como o cliente também recebe o certificado do cliente(que contém a chave pública), consegue validar a sua assinatura.

3.2 Destination Validation

Sempre que um cliente ou servidor envia uma mensagem, ele gera uma **hash** dessa mensagem para que possa comparar com o hash que vai receber de volta. Esse hash vem num **ack** que pode vir dentro de um secure após o cliente estar conectado com o servidor, ou num ack sem encapsulamento antes da ligação com o servidor estar estabelecida.

O cliente ou servidor ao receber um ack vai verificar se o hash que este traz coincide com o hack que enviou. Caso esta comparação não coincida não é possível garantir o destination validation e houve uma falha de segurança, o que nos leva a fechar a ligação.

3.3 Identity Preservation

Todos os utilizadores que queiram utilizar o nosso chat têm de possuir obrigatoriamente um cartão de cidadão válido, e possuir o pin de autenticação correcto, para poderem assinar devidamente as mensagens. A validade do certificado do cartão de cidadão é verificado no servidor através do **protocolo OCSP**, tal como todos os certificados utilizados para construir a X509 store do cartão de cidadão (todos os certificados necessário para validar o cartão são adicionados a uma X509 store), este protocolo serve para verificar o estado de revogação de um certificado digital X509 (função **loadCerts**() ficheiro **utilsAES**.py), e é utilizado como uma alternativa a CRL(certificate revokation lists).

Portanto, para cada certificado presente na store é feito um pedido OCSP a uma página, consoante o tipo de certificado o pedido é feito a um destes url's ("http://ocsp.omniroot.com/baltimoreroot/","http://ocsp.ecee.gov.pt/","http://ocsp.root.cartaodecidadao.pt/publico/ocsp","http://ocsp.auc.cartaodecidadao.pt/publico/ocsp"), que irá retornar o estado daquele certificado, sobre o certificado presente no Cartão de Cidadão, é feito o mesmo. A validade do certificado do servidor também é verificada.

O Certificado também é utilizado para garantir que o mesmo cartão de cidadão não pode estar ligado ao servidor em mais que uma instância ao mesmo tempo. Para tal, o certificado enviado na 1ª fase do Handshake entre Servidor e Cliente é guardado, e caso alguém se tente ligar ao Servidor com o mesmo cartão, sem fazer uma desconexão prévia, vai ser barrado, evitando assim logins com um cartão roubado.

Referências:

- http://www.pyopenssl.org/en/stable/api/crypto.html#x509-objects
- http://www.pyopenssl.org/en/stable/api/crypto.html#x509store-objects

3.4 Information Flow Control

Há um controlo de troca de mensagens consoante o nível associado a cada utilizador, conforme a política de controlo Bell-Lapadula. Um cliente que tenha o nível atribuído que seja superior ao do outro e lhe tente enviar uma mensagem, a mensagem é entregue. Caso o cliente tenha um nível inferior ao do cliente que quer comunicar, a mensagem é barrada no servidor e é apresentada uma mensagem de erro a indicar que o nível que tem não lhe permite enviar mensagens ao cliente que deseja.

3.5 Participant Consistency

A feature de Participant Consistency consiste na deteção do envio de mensagens de um computador diferente das anteriores, e foi explicado no ponto 2.4.

4 Modo de utilização

O Trabalho foi desenvolvido em Python 2.

4.1 Iniciar Cliente e Servidor

Entrar no VirtualEnv:

- $ source venv/bin/activate

De seguida, é necessário iniciar o Servidor:

- $ venv/bin/python Server.py

De seguida, podemos colocar a correr vários clientes com o comando:

- $ venv/bin/python python Client.py

Para ver as bibliotecas instaladas:

- $ venv/bin/pip freeze

Caso ocorra algum erro com o venv (ocorreu na maquina de Segurança):

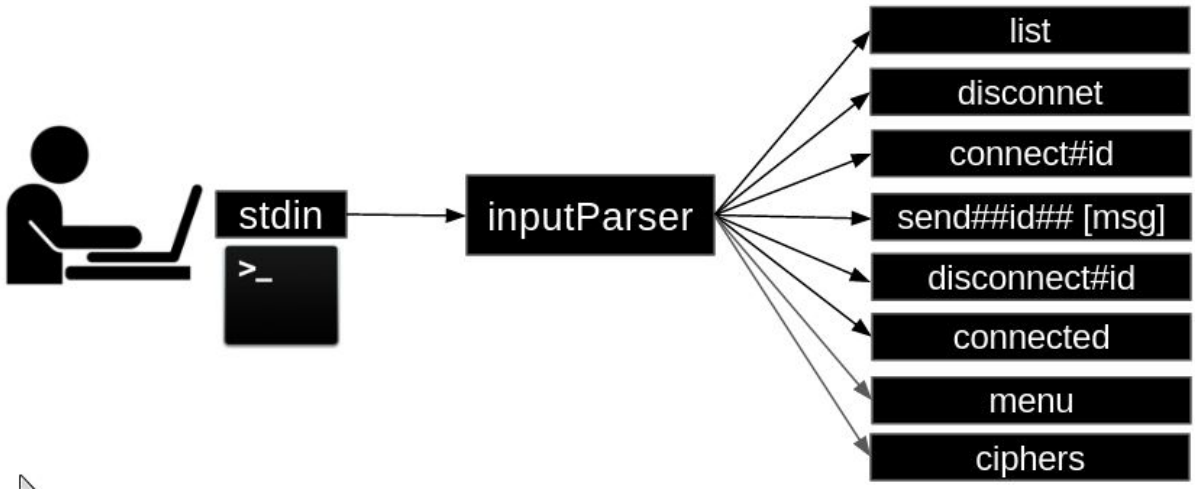
- $ sudo apt-get update
- $ sudo apt-get install virtualenv build-essential libssl-dev libffi-dev python-dev
- $ virtualenv venv
- $ source venv/bin/activate
- $ venv/bin/pip install -r requirements.txt

Após o cliente estar a “correr”, é necessário introduzir o slot (0 para o 1º, 1 para o 2º, etc…), escolher o cipher spec, e de seguida o Pin de autenticação do cartão de cidadão presente no slot.

4.3 Inserção de Comandos

Após o handshake, o utilizador (Cliente) pode interagir com o nosso Sistema através da inserção de comandos na sua linha de comandos.

Em primeiro lugar é mostrado ao utilizador no final de cada operação uma lista de comandos que o cliente pode processar, o utilizador após escrever o comando e carregar “Enter” vai fornecer o stdin ao parser, que vai chamar as funções necessárias para desempenhar as operações fornecidas.



São suportados vários comandos:

- **list:** Lista todos os clientes que estão conectados ao Servidor, naquele momento.
- **disconnect:** Termina a sessão com o servidor, limpando todos os dados associados a essa conexão.
- **connect#id:** O Cliente faz a operação de se ligar ao Cliente com o ID especificado, esta operação apenas é bem sucedida se o Cliente estiver conectado ao Servidor, e se forem acordadas chaves entre estes dois clientes.
- **send##id##msg:** Envio de mensagem para o Cliente com o ID especificado, para esta operação ser bem sucedida, ambos os clientes já têm de ter estabelecida uma conexão entre eles.
- **disconnect#id:** Envio de um pedido de desconexão ao Cliente com o ID especificado. Quando concluído, todos os dados de sessão entre os 2 clientes são apagados.
- **connected:** Listagem da informação dos Clientes ligados a “nós próprios”.
- **menu:** Listagem do menu de comandos
- **ciphers:** Listagem de cipher suites suportadas pelo cliente.