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(a ser realizado na 2 fase do projeto).

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

2 Processos da 1ª entrega

2.1 Processo de HandShake entre Cliente-Servidor

Para que um cliente possa ser aceite pelo servidor de forma a trocar mensagens, é preciso seja estabelecido um acordo sobre as cifras a serem utilizadas para estabelecer uma ligação segura e cifrar as mensagens. Para haver uma troca de mensagens entre clientes é necessário garantir a confidencialidade das mesmas. Esta é atingida através do uso de mensagens cifradas que apenas são decifradas pelo destino.

Diffie-Hellman é uma forma de gerar um segredo entre 2 entidades, mas sem que este possa ser obtido por uma 3 entidade que esteja a observar a comunicação(eve). Não há troca de informação privada durante a troca de chaves, mas sim a criação de uma chave em conjunto.

No entanto, após uma pesquisa exaustiva chegamos à conclusão que a melhor abordagem a este problema é através da combinação do método de Diffie-Hellman com o algoritmo de acordo de chaves Elliptic Curve. Assim, o nosso método usado é o **Elliptic curve Diffie–Hellman** (**ECDH**).

2.1.1 **Na Prática:**

Considere que existe uma entidade chamada **Alice** que quer estabelecer uma ligação segura com uma entidade chamada **Bob**. No entanto, esta ligação está a ser observada por uma 3 entidade, a **Eve**. Caso haja a troca do segredo entre a **Alice** e o **Bob**, a **Eve** irá tomar conhecimento deste e pode usá-lo para cifrar e decifrar as mensagens. Por isso, o segredo nunca é tornado público.

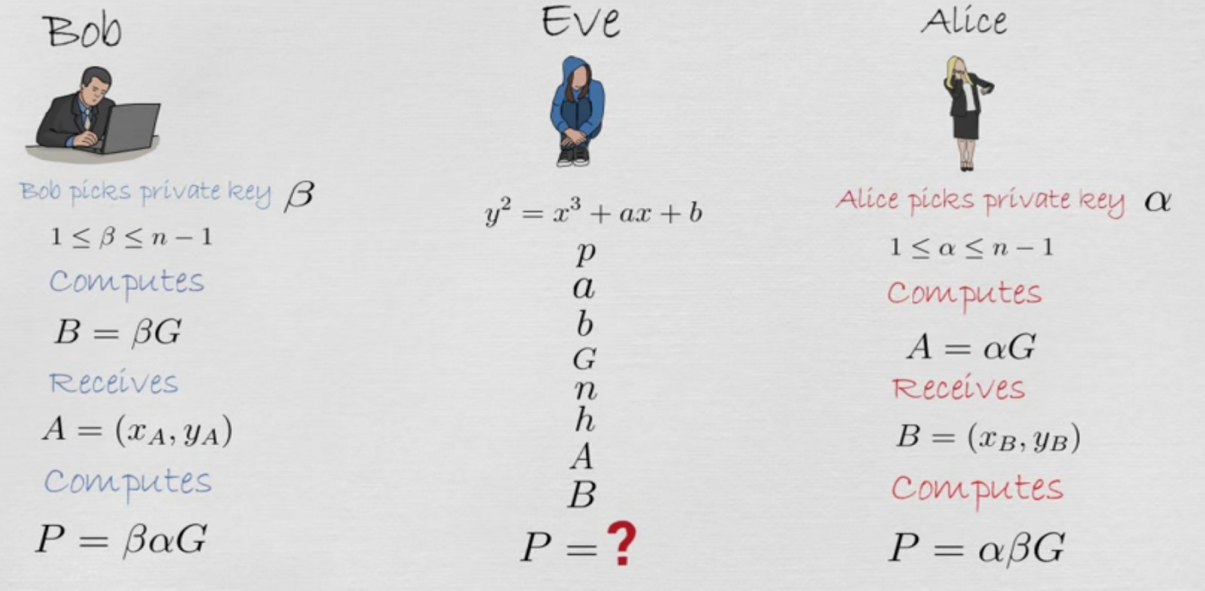
O que acontece é que tanto a **Alice**, como o **Bob** vão gerar uma chave privada, α e β, respectivamente. Estas chaves privadas não são partilhadas com ninguém, mas é através delas que são criadas as suas chaves públicas, **A** e **B**, respectivamente.

As chaves públicas são trocadas entre os dois, e mesmo que a **Eve** as intercepte, não há problema, porque a **Eve** nunca teve acesso às chaves privadas de ambos. E é devido às chaves privadas nunca terem sido trocadas que é possível gerar um segredo em conjunto. A **Alice** vai usar a sua chave privada e a chave pública do **Bob** para gerar o segredo, e o **Bob** a sua chave privada e a chave pública da **Alice**. Como a **Eve** nunca teve conhecimento das chaves privadas, ela vê-se incapaz de reproduzir o segredo.

Tal como foi explicado anteriormente, o α e o β são as **chaves privadas** da **Alice** e do **Bob**, que correspondem a um conjunto de pontos na recta de uma curva elíptica. Esse valor é multiplicado por um **factor de Geração** G, o que vai gerar uma **chave pública** B e A. O valor de B, A e mesmo G é passado pelo canal inseguro, o que permite à Eve ver estes valores. Até esta altura, poderíamos pensar que seria fácil para a Eve descobrir o valor das chaves privadas, visto que possui os valores de G, B e A, no entanto, isto torna-se praticamente impossível devido à geração da chave pública ser baseada em multiplicação escalar de curvas elípticas, que é considerada uma ***One Way Function***, isto é, uma função em que se calcula facilmente o resultado, mas é muito difícil de fazer a operação inversa.

Matematicamente explicando: Se tivermos uma curva elíptica, em que os pontos Q e P pertencem à curva, em que Q é múltiplo de P, o problema logarítmico da curva elíptica discreta diz que encontrar o k dado que $Q = kP$ é **muito difícil** (computacionalmente quase impossível).

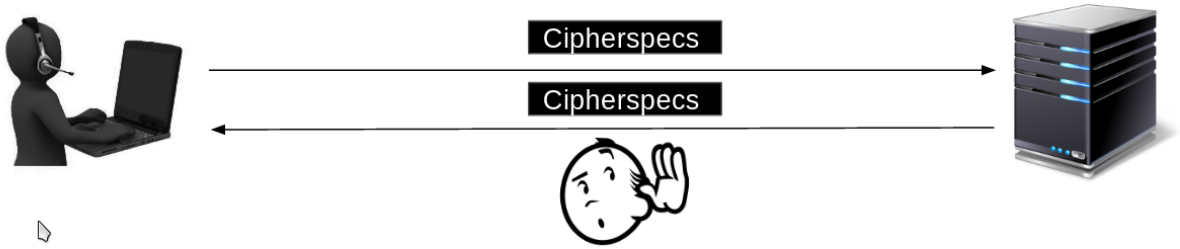
O P na imagem é o segredo partilhado pelo **Bob** e pela **Alice**, o qual é calculado através de uma multiplicação chave privada de cada um e a chave pública do outro. Ou seja, o **Bob** descobre P através da multiplicação da sua chave privada(β) com a chave pública da **Alice**(A): $P = \beta A$



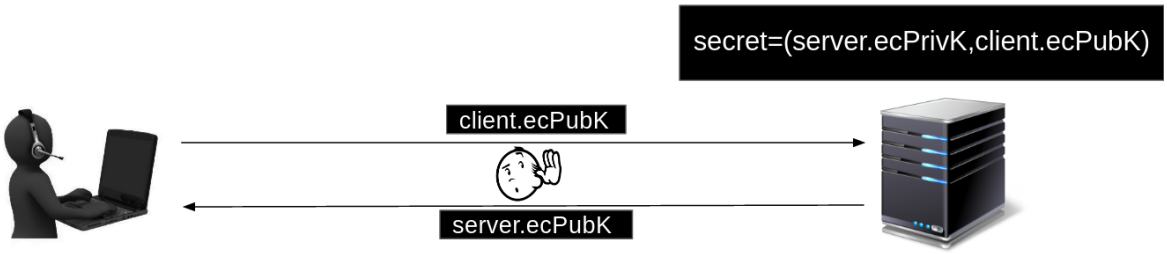
Como acréscimo de segurança, o nosso método de ***Elliptic Curve Diffie-Hellman*** é **efémero** (ECDHE), que é diferente do Diffie-Hellman normal, pela seguinte razão: vamos proceder sempre à derivação de duas novas chaves a cada troca de mensagens. Uma para gerar o HMAC e outra para cifrar a mensagem. Desta maneira nunca colocamos em perigo comunicações anteriores, ou seja, cada conjunto de chaves para cada sessão é temporária, o que assegura ***Forward Secrecy*** e **Backward Secrecy**, visto que com a chave derivada atual "x", não conseguimos chegar à chave "x-1" nem à chave "x+1".

2.1.2 Modelo TLS:

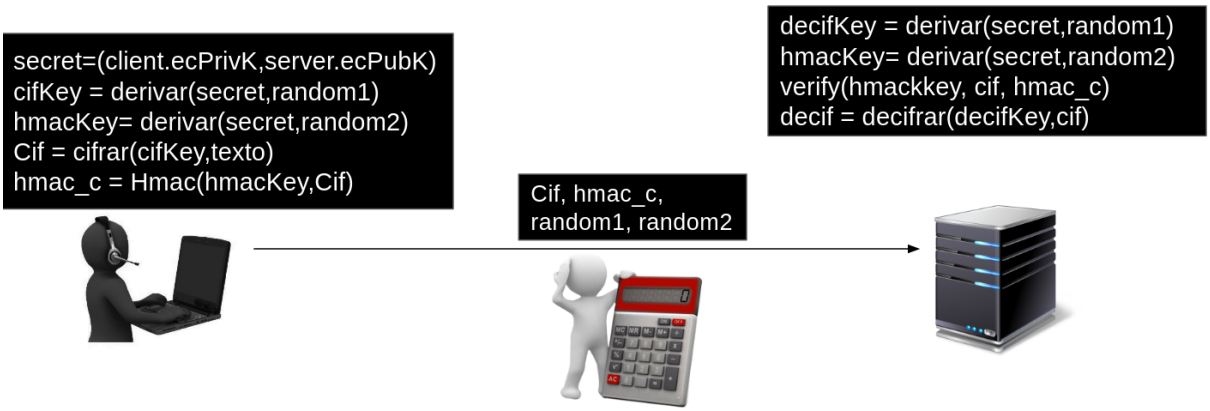
O Modelo em que nos baseamos para fazer o handshake completo entre o Cliente e Servidor foi o modelo de handshake do TLS.



- O **cliente** envia uma mensagem para o servidor com os cipherspecs que suporta.
- O **servidor** verifica se suporta os cipherspecs que o cliente lhe enviou e envia de volta os que coincidem.

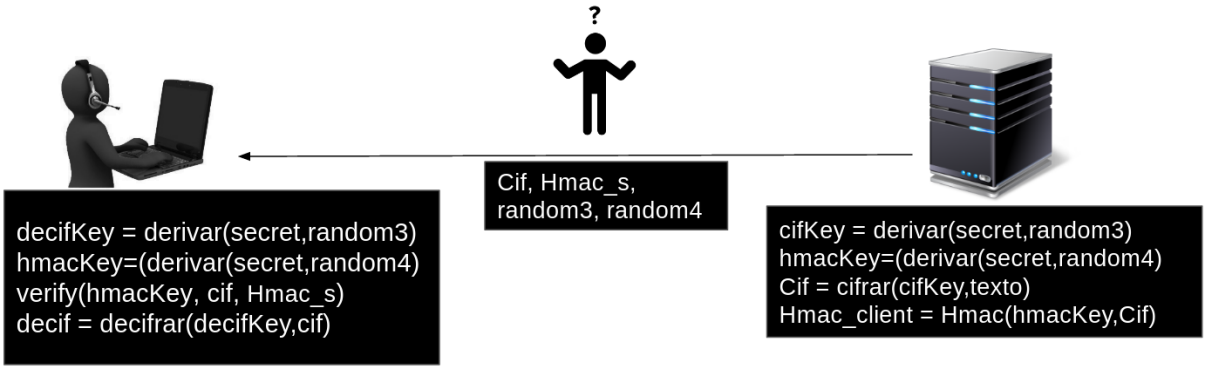


- O **cliente** envia o cipherspec que deseja utilizar(dentro dos que coincidem) e uma mensagem contendo a sua chave pública (ECDH).
- O **servidor**, guarda o cipherspec, gera o segredo (através da **sua** chave privada e da chave pública **recebida**) e envia ao cliente a sua chave pública gerada pelo ECDH.



- O **cliente**, recebe esse valor e gera o segredo da mesma maneira que o servidor, através da **sua** chave privada e a chave pública **recebida**. Desta forma, ambos têm o segredo formado, e que irá ser utilizado nesta sessão para **derivar** as chaves necessárias para os processos criptográficos.
  O **cliente** deriva uma chave a partir do segredo comum com um número pseudo-aleatório(salt), e **cifra uma mensagem**. Volta a derivar outra chave a partir do segredo com um novo número pseudo-aleatório(salt), e vai criar um **HMAC** sobre a mensagem cifrada. A mensagem cifrada enviada é a mensagem original cifrada mais o HMAC da sua cifra (**encrypt-then-MAC**).
  Os números pseudo-aleatórios usados para gerar as duas chaves, também são enviados para o servidor.

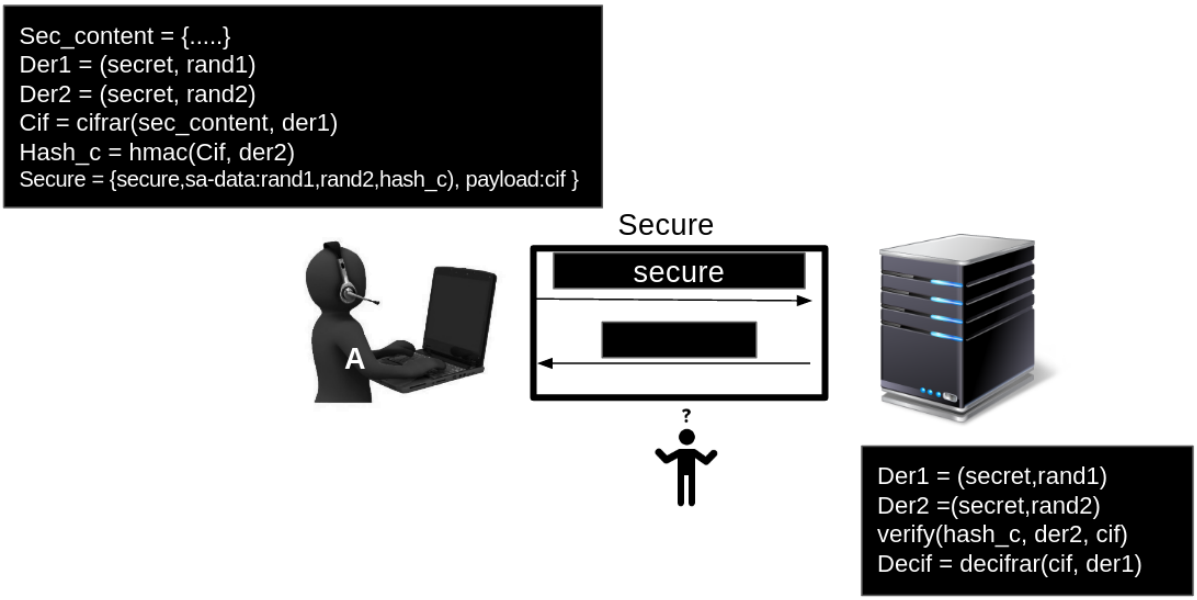
- O **servidor**, deriva duas chaves, baseadas no segredo e nos 2 números pseudo-aleatórios recebidos, com um deles decifra a mensagem e com o outro gera um HMAC sobre a mensagem cifrada. Apenas se os HMAC coincidirem é que podemos garantir a **integridade** da mensagem, ou seja, que não foi adulterada por ninguém e que é a mensagem original(**encrypt-then-MAC**). O servidor faz o mesmo processo do cliente, deriva duas chaves, cifra uma mensagem e gera um HMAC sobre a mensagem cifrada, e envia o HMAC, a mensagem cifrada e os números pseudo-aleatórios para o Cliente.



- O **cliente** gera mais duas chaves, com base no segredo e os números pseudo-aleatórios recebidos, compara os HMAC(para garantir a integridade da mensagem) e decifra a mensagem enviada pelo Servidor , para saber se o handshake pode ser terminado,
- Caso o **cliente** verifique que o HMAC enviado pelo servidor, e o HMAC que ele gera não são idênticos, volta a tentar estabelecer uma nova ligação com o servidor desde a primeira fase.
- Caso os HMAC's sejam idênticos, o processo de Handshake fica concluído e o Cliente pode ser adicionado aos clientes que o servidor tem conectados a ele. Foi garantido que o canal é seguro para a troca de mensagens, e garantido que quem está "no meio" não foi capaz de adulterar o processo de conexão.

2.3 Processo de Mensagens Secure

As mensagens do tipo ***Secure***, são mensagens que são transportadas entre o canal seguro estabelecido entre o Cliente e Servidor, e que transportam uma mensagem no seu interior. Sempre que o Cliente envia uma mensagem do tipo ***Secure*** ao servidor, o seu conteúdo, ou seja, a mensagem que vai encapsulada, é cifrada com uma nova derivação do segredo entre ambos, e é gerado um HMAC sobre a mensagem cifrada a partir de outra derivação para o servidor confirmar quando receber a mensagem se o conteúdo do Secure não foi forjado/alterado pelo meio. O fato de gerarmos outra chave derivada serve para garantir **Forward Secrecy & Backward Secrecy**.



De seguida, consoante a mensagem que vem cifrada, é feito o processamento da mensagem Segura, que pode ser do tipo List, Client-Connect, Client-Disconnect, Client-Com ou ACK.

2.3.1 Mensagens List

Após o Servidor ver que o tipo da mensagem encapsulada é do tipo ***list***, o servidor responde ao cliente uma mensagem do mesmo tipo, contendo uma lista de todos os clientes conectados a ele.

Essa mensagem é cifrada com o mesmo processo das mensagens secure descrito em cima(nova chave derivada para cifrar mensagem e gerar HMAC).

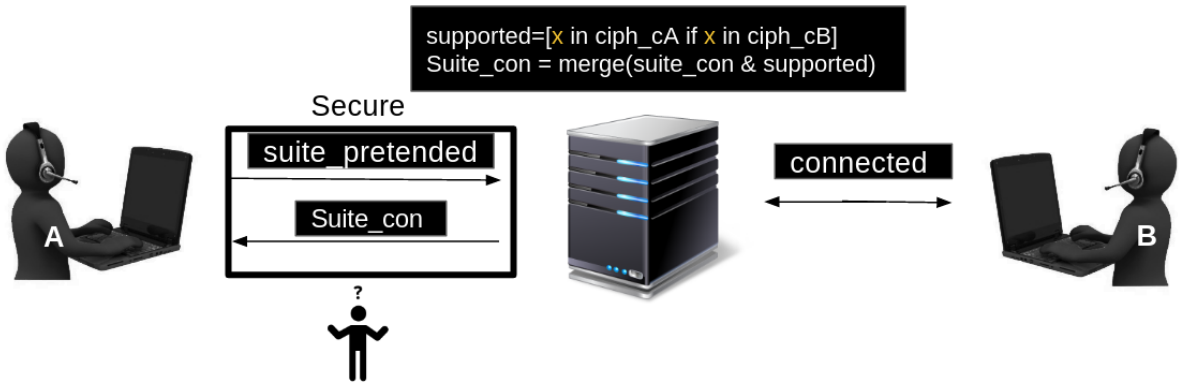
A mensagem Secure(encryptedList) é enviada pelo canal seguro, contendo também o ***sa-data*** com os dados necessários para o cliente conseguir decifrar a mensagem e garantir a sua integridade.

De seguida o Cliente processa o que recebeu dentro do Secure, e imprime para o utilizador os clientes que estão conectados ao servidor, ou seja, os clientes a que ele se pode conectar.

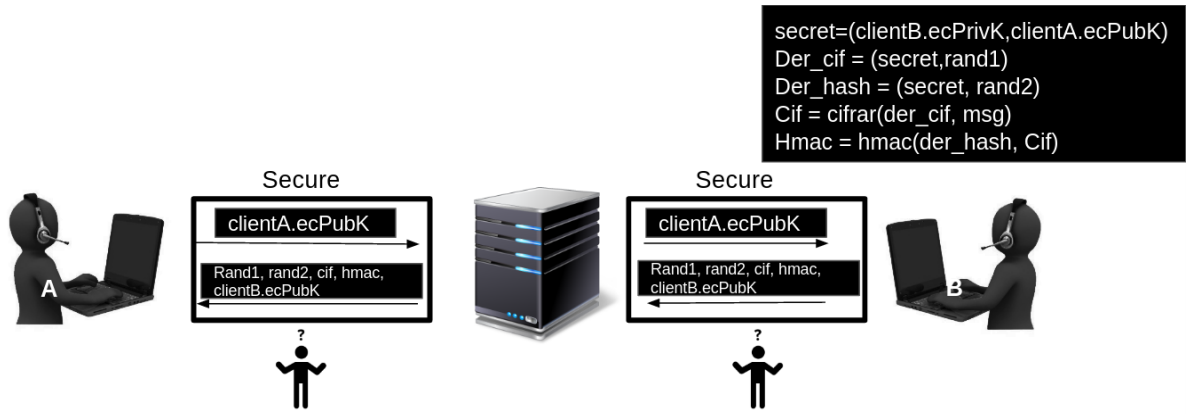
2.3.2 Mensagens Client-Connect - Handshake Client-Client

O processo de Handshake entre 2 clientes baseia-se no mesmo processo que foi implementado para o Handshake entre o Cliente e Servidor.

- Se tivermos um Cliente A que se quer conectar ao Cliente B, vai enviar uma mensagem do tipo *client-connect* encapsulada numa mensagem **Secure** com o seu ID de Origem e o ID de destino(id do Cliente B), e o/os cipher suite que ele quer utilizar na conexão ao outro cliente.
- O Servidor encarrega-se de fazer um Merge dos cipher suites que ambos os clientes suportam. Caso o Cliente A indique apenas um cipherspec e este também exista no Cliente B, o servidor devolve esse cipherspec para confirmar a utilização dos mesmo na conexão entre os clientes. Se o Cliente A indicar vários cipherspecs, o Servidor verifica aquele que tem valor criptográfico mais elevado e indica esse para utilização na conexão entre os clientes.



- O Cliente A de seguida gera o seu par de chaves para poder ser estabelecido um canal seguro entre os 2 clientes, sobre o canal seguro entre [Cliente A - Servidor, Servidor - Cliente B], e envia a sua chave pública para o Cliente B, através de um ***client-connect*** encapsulado dentro de um ***secure***.
- O Cliente B recebe os pontos públicos, gera o segredo que vai ser partilhado entre ambos, cifra uma mensagem do tipo ***client-connect*** com uma chave derivada do segredo e gera um HMAC sobre a mensagem cifrada com uma nova derivação do segredo. De seguida, envia a mensagem cifrada, o HMAC, a sua chave pública, o IV e o salts usados para cifrar e derivar chaves para o Cliente A. Esta mensagem é cifrada com uma derivação do segredo acordado entre o cliente e servidor, para garantir que a mensagem vai por um canal seguro, e ninguém a não ser o servidor consegue aceder à mensagem que vai ser encaminhada para o outro cliente.



- O Cliente A, recebe a chave pública do Cliente B e gera o segredo. Para verificar a integridade da mensagem, deriva uma nova chave e cria um HMAC a partir da mensagem cifrada. Caso o HMAC não corresponda ao HMAC enviado pelo outro Cliente, a mensagem foi adulterada. Se a correspondência estiver correta, decifra a mensagem enviada. Agora, o Cliente A faz o processo inverso, cifrando uma mensagem, com duas **novas** derivações do segredo, tanto para cifrar como para gerar o HMAC, e envia estes parâmetros para o Cliente B.

- O Cliente B, faz a mesma verificação do Cliente A, e se os HMAC forem corretos, podemos assumir que o canal entre ambos os clientes é seguro. Ou seja, ambos têm um segredo comum para poderem fazer derivações de chaves novas entre cada mensagem.
- Depois de ambos verificarem que conseguem cifrar e decifrar mensagens correctamente a partir de derivações do seu segredo, a conexão entre Clientes fica estabelecida através do canal seguro com o servidor. Quando é excedido o tempo limite de uma sessão entre 2 clientes, ou um número de mensagens definido como limite máximo para validade de uma sessão, este processo é repetido. Com isto conseguimos garantir outra camada de ***Forward Secrecy*** e **Backward Secrecy**.

2.3.3 Mensagens Client-Com

Do ponto de vista do servidor, após a receção de um ***Client-Com***, o servidor apenas decifra o que vem do cliente origem e cifra o ***Client-Com*** para este poder "passar" no canal seguro que foi estabelecido previamente com o cliente destino, e gera o HMAC relativo à sua troca de mensagem com o cliente destino. Ou seja, apenas encaminha as mensagens entre os clientes, através do canal seguro que tem com ambos. O Cliente de destino recebe a mensagem, verifica se esta não foi adulterada e decifra-a(com geração de nova chave derivada). Após isto responde com um Ack, dentro de um Secure, para o Cliente que lhe enviou a mensagem, para sinalizar que de facto a mensagem enviada pelo cliente não foi extraviada.

Do ponto de vista do Cliente, o Cliente produz o conteúdo da mensagem de texto que quer enviar ao cliente de destino, cifra o texto da mensagem com uma derivação do segredo partilhado com o cliente de destino e gera um HMAC do texto cifrado com outra derivação do segredo partilhado. Envia isto dentro do canal seguro que tem com o servidor. O outro Cliente ao receber a mensagem, verifica a integridade da mesma, decifra e imprime a mensagem ao utilizador. Mal recebe a mensagem, envia um ACK ao cliente que originou a mensagem para este saber que a sua mensagem foi entregue.

2.3.4 Mensagens Client-Disconnect

Após o Servidor receber uma mensagem do tipo Client-Disconnect, apenas vai encaminhar para o destino, mas desta vez com a mensagem Client-Disconnect cifrada com os dados de segurança acordada com o cliente que corresponde ao destino.

Quando cada um dos clientes recebe a mensagem em si, tem de decifrar com os parâmetros *sa-data* que o servidor envia.

Visto de maneira faseada, o Cliente A que faz o 1º pedido, envia uma mensagem com o tipo client-disconnect, através do Secure, essa mensagem no campo data contém um parâmetro 'cif' que leva uma frase cifrada com os parâmetros entre clientes e que indica ao outro cliente que de facto eu me quero desconectar, e também com um parâmetro 'phase' a 1, quando o cliente B recebe a mensagem, verifica se o texto que vem cifrado corresponde realmente a uma mensagem de desconexão, incrementa a fase que veio no campo data, e após a mensagem ser enviada (bufout ser esvaziado) elimina toda a informação que tem do Cliente A e remove da sua lista de clientes ligados. O Cliente A, que pediu a desconexão, recebe a mensagem com a fase incrementada, o que serve como uma espécie de *ack* para desconexões, e elimina toda a informação associada ao cliente B.

2.3.5 Mensagens ACK

As mensagens ACK, são sempre enviadas pelo cliente de destino após a receção de uma mensagem do tipo Client-Com. As mensagens deste tipo vão cifradas do destino para o servidor, o servidor decifra, e encaminha o Ack para o Cliente que originou o Client-Com, e assim, podemos garantir ao utilizador que a mensagem dele de facto chegou ao destino.

2.4 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 ao cliente que vai se desconectar.

Quando faz a desconexão dos clientes a que está conectados, vai enviar uma mensagem para o servidor, novamente, cifrada com os dados entre ambos e também um hmac para verificação. O servidor ao receber essa mensagem, responde ao cliente com uma mensagem cifrada no campo data que serve como forma de indicar ao cliente que ele se pode desconectar sem problemas, mal essa mensagem seja enviada pelo bufout, ele elimina o cliente e todos os seus dados associados, o cliente ao receber essa mensagem termina o programa, visto que já não tem conexão com o servidor.

3 Features de Segurança

Para esta fase do trabalho, foram implementadas as seguintes *features* de segurança:

- **Integridade do Ciphertext:** Através do método *Encrypt-then-MAC* (o ciphertext corresponde ao append do texto encriptado e do Hmac gerado sobre o texto encriptado) aplicado neste trabalho é possível assegurar a integridade do Ciphertext. Assumindo que o segredo entre duas entidades não foi comprometido podemos deduzir isso. No entanto, a outra parte envolvida, quando faz a decifragem da mensagem verifica o HMAC da mensagem cifrada recebida, e se não bater certo, podemos quebrar a ligação ou forçar a uma nova geração de acordo de segredo (handshake).
    - Resolvemos utilizar o método Encrypt-then-MAC, porque é mais seguro do que *MAC-then-encrypt ou Encrypt-and-MAC*. Isto porque, com o Encrypt-then-MAC, o destino ao receber a mensagem tenta validar o hash, e basta ele não bater certo para nem sequer proceder ao processo de decifrar a mensagem recebida, prevenindo assim ataques de negação de serviço.

    - O Método Encrypt-then-MAC assume a integridade do ciphertext ao contrário dos outros, no Encrypt-And-Mac o hmac é feito sobre o texto por decifrar logo a partir daí nunca vamos garantir integridade do ciphertext. No Mac-then-Encrypt o ciphertext é gerado a partir do hmac do texto cifrado e da cifragem desse hmac.

- **Integridade das mensagens:** O Método Encrypt-Then-Mac oferece integridade sobre as mensagens, visto que, uma vez que depois de verificado o hmac do texto cifrado recebido com o hmac recebido no ciphertext, podemos assumir que a decifra da mensagem cifrada também vai ser igual, visto que a derivação de chave utilizada para decifrar vai ser originada com o salt que originou a chave que cifrou a mensagem originalmente.

- **Validação do destino:** Na primeira fase do trabalho, a validação do destino ainda não pode ser implementada a 100%, apesar de ser enviado um ACK após a receção de cada mensagem, esse ACK ainda não é assinado por quem o envia e verificado no destino, portanto, será uma feature que estará a 100% na segunda fase do trabalho.

- **Forward Secrecy & Backward Secrecy:** O conceito do Forward Secrecy corresponde é uma propriedade que nos diz que uma chave de um momento X não vai comprometer chaves futuras (x+1), e o de Backward Secrecy o contrário, uma chave de um momento X não pode comprometer chaves anteriores (x-1). Na implementação que efectuámos, a cada mensagem que é cifrada e enviada, é utilizada uma nova chave derivada a partir do segredo, logo as chaves para cada mensagem são diferentes, o que nos garante que uma mensagem cifrada com

uma chave X não vai poder comprometer a segurança da mensagem anteriores, nem das seguintes, visto que a chave utilizada para cifras e hmac vai ser novamente gerada. De forma a incrementar mais o nível de segurança, após um período temporal de 5 minutos são gerados novos segredos entre Cliente-Servidor ou Cliente-Cliente, ou então após o envio de X mensagens. Deste modo não só mudamos as chaves utilizadas em cada mensagem, como após um período temporal ou um limite de mensagens, alteramos o segredo que serve para derivar chaves.

- **Multiple Cipher Support:** A implementação da 1ª parte do trabalho está a suportar 2 cipher suites:
    - **ECDH-AES128-SHA256:** Acordos de conexão para estabelecer um segredo gerados por um *Elliptic-Curve Diffie-Hellman*, Mensagens cifradas e decifradas com AES128, e HMACs gerados com base num hash SHA256.
    - **ECDH-AES256-SHA256:** Acordos de conexão para estabelecer um segredo gerados por um *Elliptic-Curve Diffie-Hellman*, Mensagens cifradas e decifradas com AES256, e HMACs gerados com base num hash SHA256.

4 Modo de utilização

O Trabalho foi desenvolvido em Python 2.

4.1 Instalação de dependências externas

Todos os módulos do trabalho estão dentro do mesmo directório, eles são o Server.py e o Client.py. Antes de meter o Servidor e o Cliente a correr, é necessário instalar as dependências de bibliotecas externas, para isso é fornecido um ficheiro de *requirements* para o pip, que pode ser corrido com o seguinte comando:

- (Correr na VM fornecida, dentro do directório onde se encontram os módulos)
- $sudo apt-get update
- $sudo apt-get install virtualenv build-essential libssl-dev libffi-dev python-dev # sem isso nao consigo instalar a versão do cryptography necessária **dentro da máquina**
- $virtualenv venv # dentro do directório m1
- $source venv/bin/activate
- $sudo venv/bin/pip install -r requirements.txt

4.2 Iniciar Cliente e Servidor

De seguida, é necessário iniciar o Servidor:

- $(dentro do virtualenv) python Server.py

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

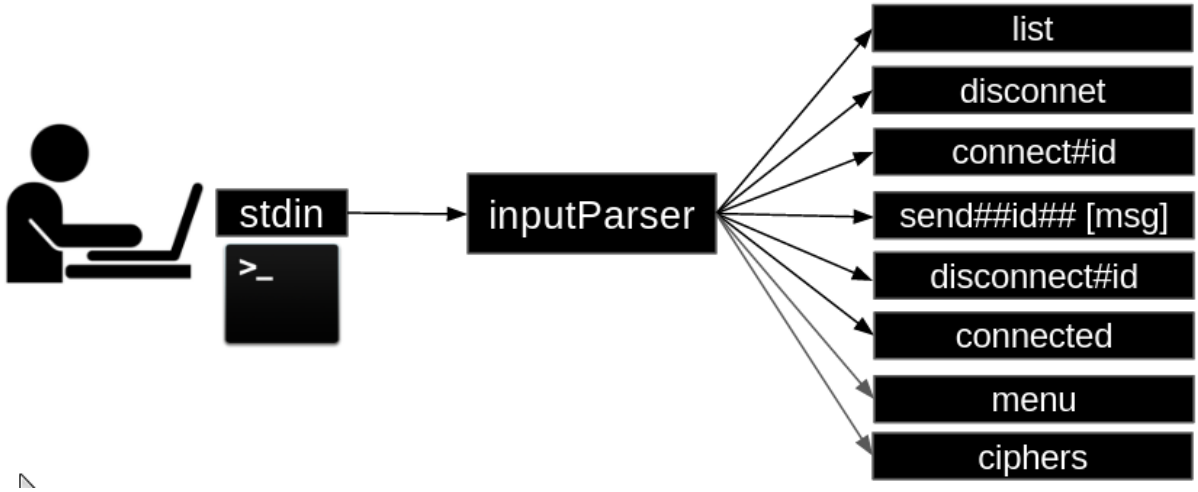
- $(dentro do virtualenv) python Client.py

Após o cliente está a "correr", é necessário introduzir o nome, e a partir daí o Cliente e o Servidor iniciam a fase de *handshake*, se for bem sucedida, o utilizador pode começar o serviço de IM seguro.

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.