

Secure Messaging Repository System

MIECT - Segurança 2017-2018 - p2g3

Realizado por:

André Rodrigues 73152
Cristiano Vagos 65169

Os Professores:

André Zúquete e João Paulo Barraca

Aveiro, 9 December 2017

Índice

Project Overview	3
INTRODUÇÃO	3
OBJECTIVOS	3
REQUISITOS	4
CONFIGURAÇÕES	4
Project Implementation	5
FUNÇÕES DISPONÍVEIS	5
REGISTO E CONEXÃO	5
LISTAGEM DOS UTILIZADORES REGISTADOS	5
LEITURA, ENVIO E ESTADO DAS MENSAGENS (MESSAGEBOX)	5
EMISSÃO DE UM RECIBO	5
IMPLEMENTAÇÃO	6
REGISTO	6
CONEXÃO (CHALLENGE E SESSÃO)	7
LISTAGEM DOS UTILIZADORES REGISTADOS	8
LEITURA, ENVIO E ESTADO DAS MENSAGENS (MESSAGE BOX)	9
EMISSÃO DE UM RECIBO	11
MENSAGEM CLIENTE-CLIENTE	11
ORGANIZAÇÃO DAS MESSAGE BOXES	12
GERENCIAMENTO DAS CHAVES ASSIMÉTRICAS E REFRESCAMENTO DAS CHAVES	12
CERTIFICADOS PÚBLICOS (VALIDAÇÃO DA CADEIA DE CONFIANÇA E DA ASSINATURA)	13
CONTROLO DO FLUXO MENSAGENS	13
INTERCEPÇÃO E CAPTURA DE PACOTES	14
CONCLUSÃO	15
REFERÊNCIAS & CODE SNIPPETS	16

PROJECT OVERVIEW

INTRODUÇÃO

O objetivo deste projeto é desenvolver um sistema que permita aos utilizadores a troca de mensagens de forma assíncrona. As mensagens são enviadas e recebidas através de um repositório central não confiável, que guarda as mensagens dos utilizadores para futuros usos, como por exemplo ler conteúdo das mensagens.

O sistema é composto por um *Rendezvous Point* (Servidor) e por vários clientes que poderão utilizar o serviço após o seu registo.

OBJECTIVOS

O sistema deve ser projectado para suportar requisitos de segurança como confidencialidade, integridade, autenticação, preservação de identidade e informação de entrega de informação.

Neste projecto é necessário estabelecer uma *session key* entre o cliente e o servidor, garantir autenticação entre estes pares e controlo de integridade na troca de mensagens, garantir que a resposta correcta corresponde ao pedido correcto, registar dados seguros relevantes no processo de criação de um cliente, utilização do cartão de cidadão no processo de registo, cifrar mensagens enviadas, assinar e validar as mensagens enviadas e recebidas, *receipts* deverão ser cifrados, o envio de um *receipt* deverá ser seguro, verificar um *receipt* correctamente, verificação dos certificados das chaves públicas, um utilizador apenas poderá ler uma mensagem da sua *Message Box* e não é permitido que um utilizador envie um *receipt* de uma mensagem não lida.

REQUISITOS

Para executar o projecto é **necessário**:

- Python2.7
- pyopenssl
- PyKCS11
- pycrypto

CONFIGURAÇÕES

Chaves:

- Chaves Assimétricas, **RSA** (2048 bits)
- Chaves Simétricas (256 bits)
- Chave Derivada da Chave de Sessão, **PBKDF2** (256 bits)

Modos de Cifra:

- AES (Simétrica, modo CBC)
- RSA (Assimétrica, PKCS1_v1_5)
- RSA sob AES (Híbrida)

Funções de Hash:

- SHA256
- SHA512 (HMAC)

PROJECT IMPLEMENTATION

FUNÇÕES DISPONÍVEIS

O sistema implementado possui as funções básicas de um repositório de mensagens com features adicionais de modo a garantir os objectivos mencionados para o projecto.

Nesta secção será feita uma breve descrição das funcionalidades do sistema, no ponto de vista de um utilizador.

Registo e Conexão

A função de registo permite que um novo utilizador seja capaz de criar um cliente no sistema, para tal é necessário fornecer dados tais como *username*, *password* e dados obtidos através do smartcard pertencente ao usuário (neste caso o Cartão de Cidadão) tais como *nome*, *serialnumber* e *certificados*.

O sistema possui várias restrições de modo a garantir consistência dos dados, por exemplo, um utilizador possui um *username* único, desta forma não existe mais do que um utilizador com o mesmo *username*.

Um utilizador pode registar mais do que um cliente, para tal é necessário que o seu *smartcard* seja válido e reconhecido pelo sistema.

Após o registo é possível efectuar uma nova conexão com o servidor, o utilizador deve introduzir as suas credenciais de modo a garantir a sua identidade.

Listagem dos Utilizadores Registados

É possível verificar quais os clientes registados no sistema e obter informações sobre estes tais como o *username*, *nome* (registado no smartcard) e *estado de ligação atual* (Online/Offline).

Leitura, Envio e Estado das Mensagens (MessageBox)

Na zona MessageBox, um cliente pode verificar o estado da sua caixa de correio, poderá ler mensagens na sua caixa de entrada, mensagens recebidas, pode verificar a sua caixa de saída, consultar o estado das mensagens enviadas e o seu respectivo conteúdo e poderá enviar uma mensagem para outro cliente indicando o *username* deste.

Emissão de um Recibo

Após a leitura de uma mensagem, o cliente poderá emitir um recibo da mensagem para o remetente, para tal é necessário o uso do seu smartcard. Após a emissão do recibo, o remetente poderá verificar o estado da mensagem, garantido que a mensagem foi lida correctamente e pela pessoa certa.

IMPLEMENTAÇÃO

Nesta secção iremos ao interior de cada função e analisá-la no ponto de vista de um programador.

Registo

Numa primeira abordagem, um utilizador executa o cliente e irá registar a sua primeira conta, para tal, o cliente executa o comando **/create**, é necessário fornecer um **username** não utilizado, uma **password** que irá ser utilizada para complementar a autenticidade e manter o seu par de chaves assimétricas guardado de forma cifrada (com a utilização da *password* como uma *passphrase*), a sua **chave pública**, RSA (2048 bits) gerada pelo cliente, e os **certificados** das chaves públicas. Ao receber os dados necessários para o registo, o servidor procede à validação das credenciais (existência do *username*) e registo dos dados.

A *password* é enviada e guardada como um *hash* (*sha256*).

É feita uma validação dos certificados e a respectiva verificação da cadeia total de confiança de modo a garantir que os certificados não foram revogados, e a validade da assinatura (com a data de registo da conta) são verificados no cliente.

No primeiro registo é guardado no cliente (directório *t1_t2/key.bin*) o seu primeiro par de chaves assimétricas cifrado com a *passphrase*, sendo que futuramente, nas próximas conexões ao sistema, o cliente terá que carregá-las decifrando-as com a *passphrase* correcta. O *t1* será o timestamp do ‘nascimento’ da chave e o *t2* será o timestamp de ‘morte’ da chave que inicialmente é vazio (por exemplo o path **15555555555531**_**15555555555530**/*key.bin*, possui a chave mais atual, enquanto que o path **15555555555529**_**15555555555530**/*key.bin*, possui uma chave correspondente ao espaço temporal **15555555555529** e **15555555555530**.

```
Python — python client.py — 70x15
~/Documents/5ano/SEG/Server/Python — python client.py
-----
Register
-----
Username: novo_utilizador
Password:
You may connect now! (/connect)

Python — python server.py — 113x6
~/Documents/5ano/SEG/Server/Python — python server.py
CREATEINFO:root: server_actions.py:168: processCreate: Creating New Account
ERROR:root: server_actions.py:168: processCreate: Certificate Not Revoked
ERROR:root: server_actions.py:168: processCreate: Valid Signature. Current Date: Thu Dec 28 18:53:42 2017
DEBUG:root:server_registry.py:101: addUser: add user description mboxes/2/description
INFO:root: server_actions.py:168: processCreate: User Added Successfully
```

Coneção (Challenge e Sessão)

Quando um utilizador pretende conectar-se ao servidor, inicialmente terá que fornecer as suas credenciais, **username**, **password** e o **pin** (assinando).

Numa primeira fase é enviada do cliente para o servidor a *hash* da *password* de modo a fazer a validação das credenciais, caso não se verifique, o processo de autenticação falha e pára. Validadas as credenciais, é novamente enviado a mesma *hash* para o servidor só que desta vez assinada de modo a garantir a autenticidade do cliente.

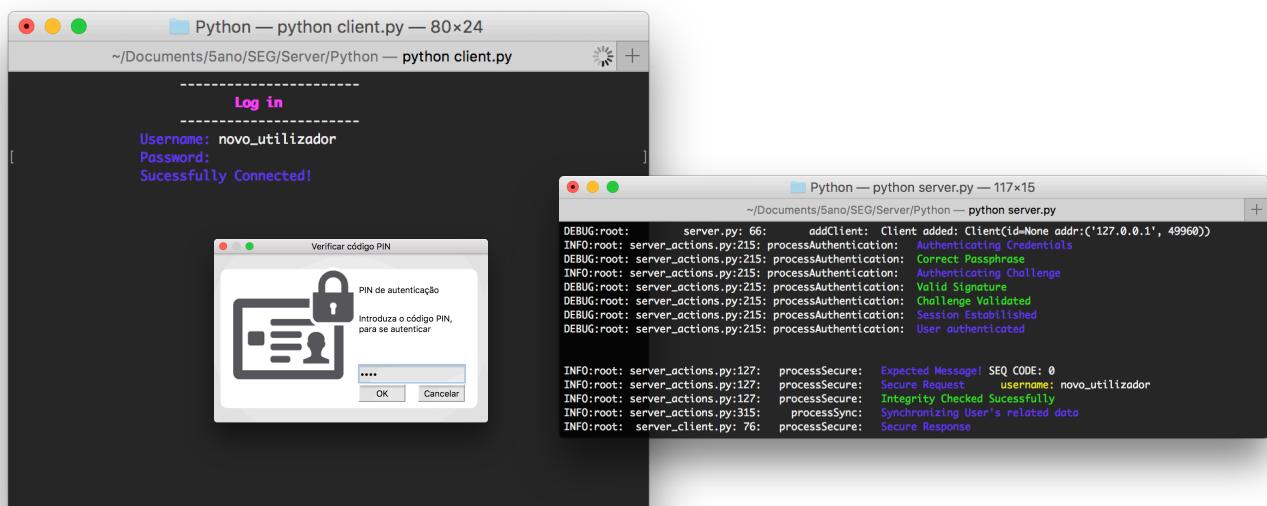
A autenticação é dividida em duas fases, validação das credenciais e posteriormente a validação da sua autenticidade (assinatura), uma vez que no caso em que um utilizador erra a sua *password*, não terá que a assinar, só terá que a assinar quando for a correcta (passar a 1^a fase, validação das credenciais).

No processo descrito, o *challenge* é um método de autenticação onde o cliente irá fornecer uma *hash* da sua *password* assinada, que terá que ser validado pelo servidor.

Na segunda fase, caso seja validada a assinatura da *hash*, é simultaneamente feito o estabelecimento de uma nova sessão, o par cliente-servidor procede com o algoritmo de Diffie-Hellman, são acordados os valores necessários e em apenas num par de mensagens (cliente-servidor e servidor-cliente), ou seja, na resposta da segunda fase, é estabelecido uma chave de sessão. Após este momento é criado um novo canal de comunicação seguro onde o conteúdo é cifrado recorrendo a uma chave simétrica derivada da chave de sessão, facilitando a “compressão” dos dados graças ao uso da cifragem simétrica, de notar que para cada pedido/mensagem trocada, é obtida uma nova chave derivada da chave de sessão indicando em cada novo pedido o *salt* necessário para gerar a chave derivada através da chave de sessão.

Estabelecida a sessão, todos os pedidos serão encapsulados e enviados neste novo canal de comunicação, para garantir a integridade é utilizado o HMAC resultante da mensagem encapsulada e da chave derivada utilizada em cada mensagem.

Após o válido ‘carregamento’ do par de chaves assimétricas e conectado com o sistema, o cliente irá fazer uma sincronização com o servidor obtendo todos os dados relativos aos demais clientes (certificados, chave publica) permitindo também a utilização de mecanismos de segurança com outros clientes (cifra, decifra, atualizando as chaves mais recentes dos outros utilizadores uma vez que não faria sentido utilizar uma chave antiga de outro utilizador para o envio de uma mensagem).



Listagem dos Utilizadores Registados

De modo a efectuar a listagem dos utilizadores registados no sistema, o cliente terá que executar o comando **/list**, uma lista de utilizadores será disponibilizada indicando o *username* e o respectivo *nome*, os restantes dados relativos aos mecanismos de segurança de cada utilizador já foram previamente distribuídos na sincronização com o sistema, feito no login.

The image displays three terminal windows illustrating the interaction between a client and a server in a secure messaging system.

- Top Terminal:** Shows the client's main menu with commands: `(/list)`, `(/all)`, and `(/send <user> <text>)`. It is connected as `novo_utilizador`.
- Middle Terminal:** Shows the available users list. The output is:

```
Available Users
Username: a    Nome: ANDRÉ FILIPE NASCIMENTO RODRIGUES
Username: b    Nome: ANDRÉ FILIPE NASCIMENTO RODRIGUES
Username: c    Nome: ANDRÉ FILIPE NASCIMENTO RODRIGUES
Username: utilidor    Nome: ANDRÉ FILIPE NASCIMENTO RODRIGUES
```
- Bottom Terminal:** Shows the server's log output. The output is:

```
INFO:root: server_actions.py:127: processSecure: Expected Message! SEQ CODE: 1
INFO:root: server_actions.py:127: processSecure: Secure Request   username: novo_utilizador
INFO:root: server_actions.py:127: processSecure: Integrity Checked Successfully
INFO:root: server_actions.py:299: processList: Listing Users
DEBUG:root: server_actions.py:299: processList: Looking for all connected users
INFO:root: server_client.py: 76: processSecure: Secure Response
```

Leitura, Envio e Estado das Mensagens (Message Box)

É possível aceder a caixa de correio, *Message Box*, através do comando /all.

A *Message Box*, é constituída por dois dicionários com key auto-incrementado, o primeiro é respetivo à caixa de entrada e o segundo à caixa de saída. Na caixa de entrada, as mensagens são ordenadas por ordem de chegada, dando prioridade às mensagens não lidas sendo estas sinalizadas no topo, já na caixa de saída estas são apenas ordenadas pela ordem de envio.

The image shows two terminal windows. The left window is titled "Python — python client.py — 84x23" and the right window is titled "Python — python server.py — 117x11". Both windows are running on the same machine, likely a Mac OS X system, as indicated by the window icons.

The client window displays the following text:

```
0 Received Messages:  
You didn't receive any message yet.  
  
0 Sent Messages:  
You didn't send any message yet.  
  
Commands:  
(/send <user> <text>) Send a Message  
(/recv <msg_number>) Read message  
(/status <msg_number>) Check Receipt Status  
(<>) Main Menu
```

The server window displays the following log output:

```
INFO:root: server_actions.py:127: processSecure: Expected Message! SEQ CODE: 2  
INFO:root: server_actions.py:127: processSecure: Secure Request username: novo_utilizador  
INFO:root: server_actions.py:127: processSecure: Integrity Checked Successfully  
INFO:root: server_actions.py:344: processAll: Correct Message Box Owner!  
DEBUG:root:server_registry.py:238: userMessages: Look for files at aboxes/2 with pattern _? [0-9]+_[0-9]+  
DEBUG:root:server_registry.py:238: userMessages: Found file description  
DEBUG:root:server_registry.py:238: userMessages: Look for files at receipts/2 with pattern [0-9]+_[0-9]+  
INFO:root: server_client.py: 76: processSecure: Secure Response
```

É possível efectuar o envio de uma mensagem através do comando **/send <username> <mensagem>**, onde será feita uma verificação da existência do *username* indicado o conteúdo da mensagem que será cifrado com a utilização de cifras híbridas (utilização da chave publica do cliente destino, garantindo a *confidencialidade*) e a copia da mensagem, para uma futura verificação do estado da mensagem que será cifrada também com o uso de cifras híbridas (utilização da chave publica do próprio cliente, garantido a *confidencialidade*). Juntamente com a mensagem será enviado a assinatura (mensagem cifrada assinada com a chave de assinatura) desta de modo a garantir a *autenticidade* da mesma, a integridade é conseguida uma vez que a mensagem é encapsulada no pedido, sendo que este pedido é novamente encapsulado e enviado juntamente com o HMAC.

Durante o processo de envio é feita uma verificação da cadeia de confiança e a validade da assinatura para que seja possível efectuar uma assinatura válida.

The image shows three terminal windows. The left window is titled "Python — python client.py — 84x23" and the right window is titled "Python — python server.py — 108x8". A small modal dialog box titled "Verificar código PIN" is overlaid on the client window.

The client window displays the following text:

```
0 Received Messages:  
You didn't receive any message yet.  
  
0 Sent Messages:  
You didn't send any message yet.  
  
Commands:  
(/send <user> <text>) Send a Message  
(/recv <msg_number>) Read message  
(/status <msg_number>) Check Receipt Status  
(<>) Main Menu
```

The modal dialog box contains:

Verificar código PIN

Introduza o código PIN, para se autenticar

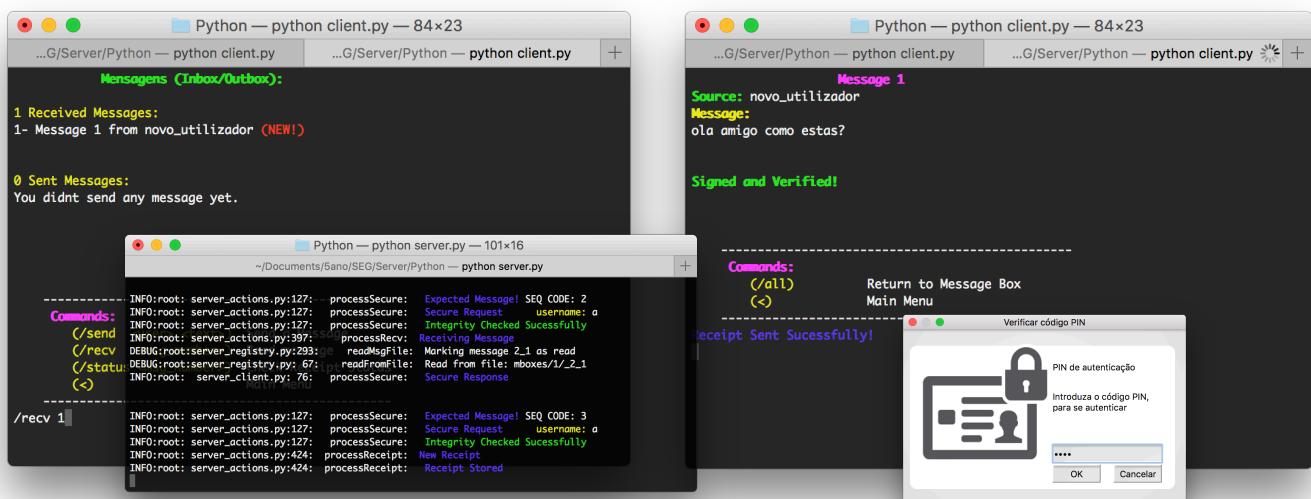
... OK Cancelar

The right window displays the following log output:

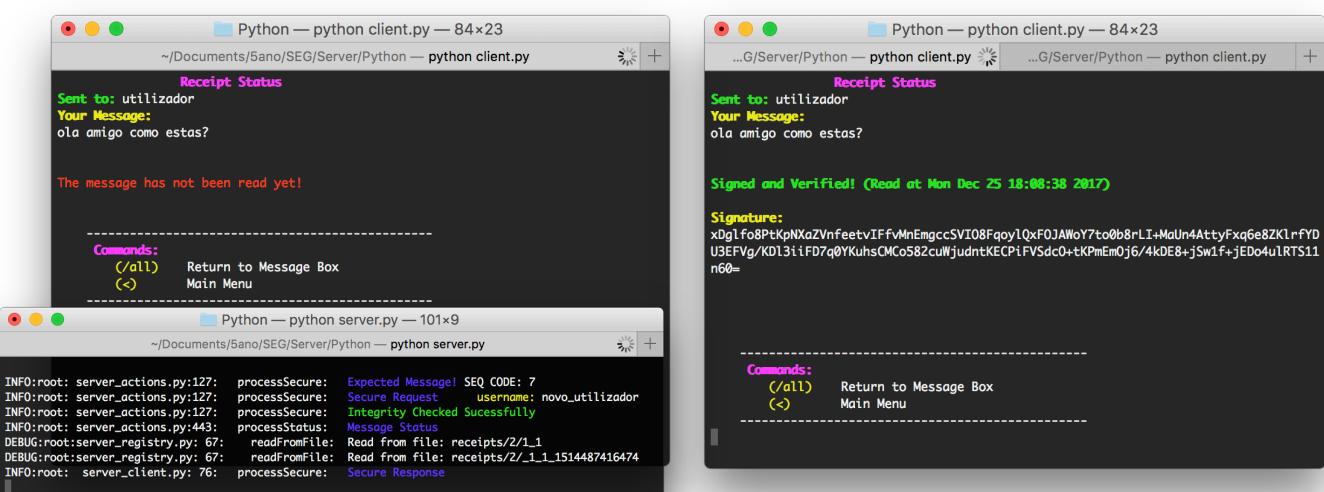
```
INFO:root: server_actions.py:127: processSecure: Expected Message! SEQ CODE: 5  
INFO:root: server_actions.py:127: processSecure: Secure Request username: novo_utilizador  
INFO:root: server_actions.py:127: processSecure: Integrity Checked Successfully  
INFO:root: server_actions.py:366: processSend: Sending Message  
INFO:root: server_actions.py:366: processSend: Message Sent Successfully  
INFO:root: server_client.py: 76: processSecure: Secure Response
```

A leitura da mensagem é feita recorrendo ao comando **/recv <id>**, onde o **id** seleciona a mensagem do dicionário correspondente à caixa de entrada. Para uma correcta leitura, é necessário que o cliente possua a sua chave privada carregada, ou seja, será usada a *password* como *passphrase* para fazer o seu “carregamento” e este terá que ser válido, assim, apenas quem a possui é capaz de a ler, garantindo *confidencialidade*. De modo a escolher correctamente a chave privada associada à mensagem é usado o *timestamp* de escrita da mensagem.

No processo de leitura, são verificadas as cadeias de confiança dos certificados, a validade da chave de assinatura e a assinatura em si, do cliente remetente. É obrigatório que o cliente assine o *receipt*, na primeira leitura senão não será capaz de obter o conteúdo da mensagem.



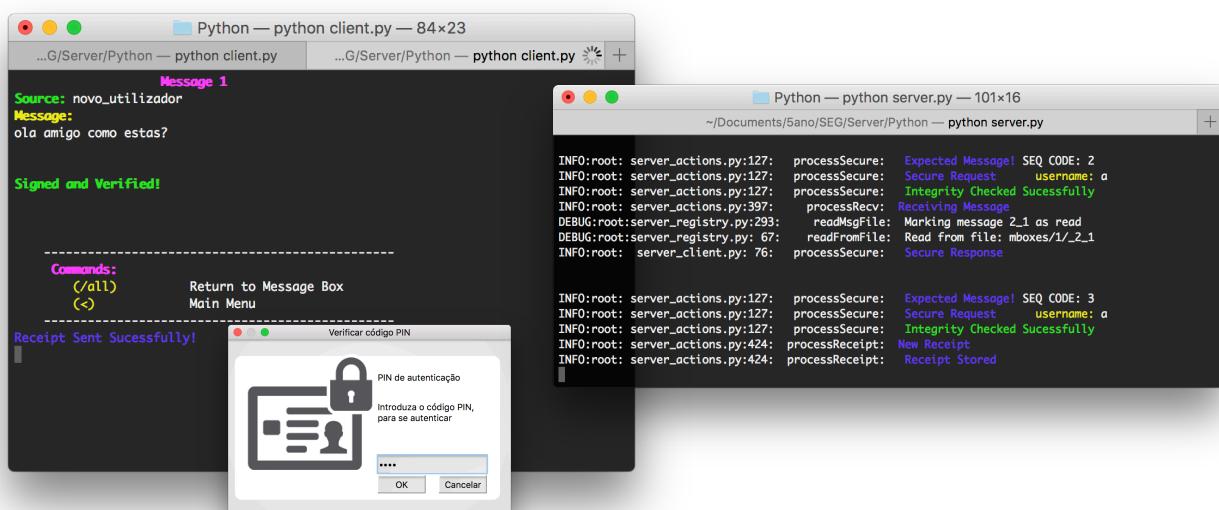
Ao verificar o estado de uma mensagem enviada, é possível saber se a mensagem foi entregue e lida com sucesso, obtendo detalhes tais como o *timestamp* da leitura da mensagem e a assinatura que comprova que a mensagem foi lida pelo utilizador correcto e não por outrem (validada no cliente), do mesmo modo, para uma correcta leitura é necessário recolher o *timestamp* do *receipt* de modo a escolher a chave privada correcta para decifrar o conteúdo.



Emissão de um Recibo

Durante a primeira leitura, o cliente terá que obrigatoriamente enviar um *receipt* da leitura da mensagem informando quando é que a mensagem foi lida, *timestamp*, e que foi lida por si, para tal terá que o assinar com a sua chave de assinatura, relembrando que o seu conteúdo foi previamente cifrado recorrendo a uma cifra híbrida utilizando a chave publica do cliente que enviou a mensagem, ou seja, o estado apenas poderá ser verificado pela utilizador que enviou a mensagem tal como se sucede quando uma mensagem é enviada, o destinatário será o único capaz de a decifrar e obter o seu conteúdo.

Para um correcto envio e uma válida assinatura, é novamente feita a validação da cadeia de confiança e da assinatura com o *timestamp* atual de envio do *receipt*.



Mensagem Cliente-Cliente

Como já foi referido anteriormente, todos os pedidos ao servidor serão encapsulados garantindo segurança, autenticação e integridade, estes pedidos poderão ser de vários tipos.

No caso de envio de uma mensagem nova para outro cliente o tipo será *send*, o campo **msg** irá conter um dicionário com o texto cifrado com uma chave simétrica e a chave simétrica cifrada com a chave publica do cliente destinatário, garantindo confidencialidade para o destinatário. No campo **copy**, necessário para futuramente consultar o estado de uma mensagem, é igualmente um dicionário mas neste caso a chave simétrica é cifrada com a chave publica do cliente remetente, garantindo confidencialidade para o remetente.

Juntamente com a mensagem será enviado a assinatura do conteúdo de modo a garantir a autenticidade do cliente remetente no cliente destinatário e o *timestamp* aquando a sua escrita.

Organização das Message Boxes

O servidor realiza uma transparência em relação à atribuição e tradução das Message Boxes aquando a sua criação / registo no sistema.

Um utilizador é distinguido univocamente pelo seu uuid (*username*) que estará associado a um id (auto-incrementado) que representa a sua Message Box.

São feitas verificações de modo a garantir a consistência dos dados.

Gerenciamento das Chaves Assimétricas e Refrescamento das Chaves

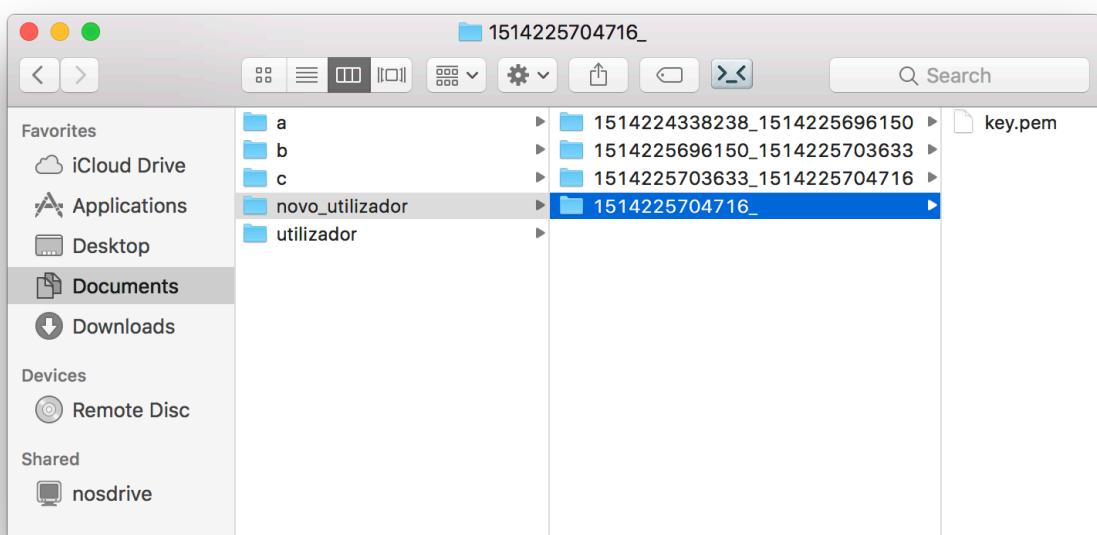
De modo a cumprir uma política de “Back-Forward-Policy”, as chaves têm o seu período de vida garantindo que se a chave privada de um cliente for descoberta num determinado espaço de tempo, as mensagens decifradas antes e depois desse espaço temporal não serão decifradas.

O primeiro par de chaves assimétricas é criado no registo, exportado e guardado num directório no cliente, este par está cifrado com uma *passphrase*, futuramente, após um tempo aleatório pré-definido será efectuado o refrescamento das chaves simétricas e chave de sessão e este novo par de chaves assimétrico é novamente exportado e guardado no cliente.

Em qualquer momento, um cliente pode decifrar e ler qualquer mensagem na sua Message Box, para tal, é feita uma verificação e seleção da chave correcta de acordo com o espaço temporal onde a mensagem foi criada, ou seja, para cada espaço temporal temos uma chave e para decifrar uma mensagem enviada no tempo *t* será necessário importar e usar uma chave que represente e/ou inclua esse espaço temporal.

Complementando, de modo a garantir a segurança do canal ao longo do tempo, no final de um número aleatório pré-acordado, o cliente e o servidor vão acordar uma nova chave de sessão e também gerar um novo par de chaves assimétricas, conseguindo assim o *refrescamento das chaves*.

Novamente, no directório *clientes/username/* existirá uma lista de directórios com o esquema *N_M*, onde *N* é o *timestamp* quando o par de chaves foi criado e o *M* é o *timestamp* quando o par foi substituído, correspondendo assim ao tempo de vida de uma chave *[N,M]*.



Certificados Públicos (Validação da Cadeia de Confiança e da Assinatura)

A validação da cadeia de confiança, por parte do servidor é feita no registo, as demais verificações, assinatura e data da assinatura são verificadas no registo e no processo de autenticação.

No cliente é verificada a cadeia de confiança e a validade da assinatura e correspondente data no envio de uma nova mensagem, o mesmo para a leitura de uma nova mensagem, no envio de um *receipt* e no processo de autenticação.

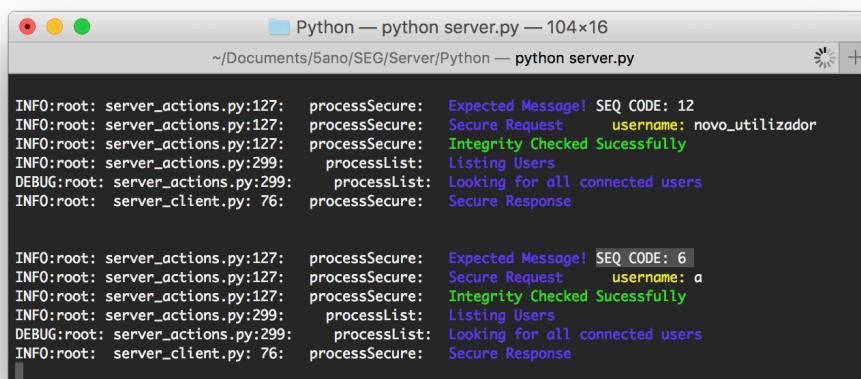
Na verificação de um *receipt* apenas é feita a validação da assinatura e da data da assinatura quando foi emitido o *receipt*.

Para obter e validar a cadeia de confiança foi utilizado o OpenSSL, obtendo os *issuers* de cada certificado e montando as diferentes cadeias de confiança (autenticação / assinatura), obtendo os *bases* e os *deltas*, e finalmente validando as cadeias numa x509 *Store Context* com a flag *CRL_CHECK_ALL*.

Para a validação da assinatura é usada a função *verify* (OpenSSL.crypto), para a validação da data da assinatura é verificado se o *timestamp* recolhido / de emissão da assinatura se encaixa no período definido pelo *NOT_BEFORE* e *NOT_AFTER* que se encontram no certificado da chave pública do cliente.

Controlo do Fluxo Mensagens

Tanto como no Cliente como no Servidor, foi feito um registo de modo a especular que futuras mensagens devem aparecer e serem tratadas, evitando ataques de injeção de pacotes, foi introduzido no pacote de mais alto nível que encapsula os pedidos, um *msgCode* que serve como um índice de uma sequência que confirma o fluxo da mensagem, assim é possível distinguir a resposta correcta para diferentes pedidos feitos simultaneamente.



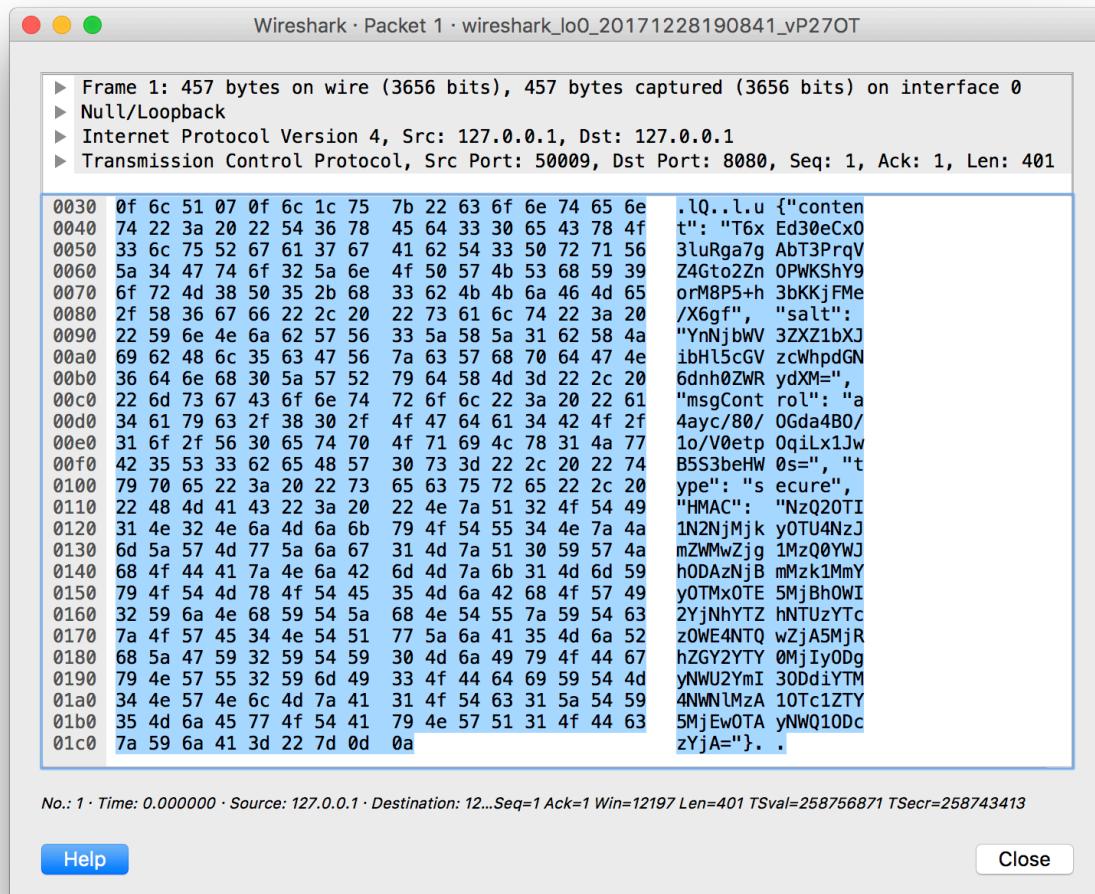
The screenshot shows a terminal window titled "Python — python server.py — 104x16". The window displays log messages from the "server_actions.py" and "server_client.py" files. The log entries are color-coded: blue for INFO level, purple for DEBUG level, and red for ERROR level. The log messages include:

```
INFO:root: server_actions.py:127: processSecure: Expected Message! SEQ CODE: 12
INFO:root: server_actions.py:127: processSecure: Secure Request username: novo_utilizador
INFO:root: server_actions.py:127: processSecure: Integrity Checked Sucessfully
INFO:root: server_actions.py:299: processList: Listing Users
DEBUG:root: server_actions.py:299: processList: Looking for all connected users
INFO:root: server_client.py: 76: processSecure: Secure Response

INFO:root: server_actions.py:127: processSecure: Expected Message! SEQ CODE: 6
INFO:root: server_actions.py:127: processSecure: Secure Request username: a
INFO:root: server_actions.py:127: processSecure: Integrity Checked Sucessfully
INFO:root: server_actions.py:299: processList: Listing Users
DEBUG:root: server_actions.py:299: processList: Looking for all connected users
INFO:root: server_client.py: 76: processSecure: Secure Response
```

Intercepção e Captura de Pacotes

Qualquer pedido após ter sido estabelecida uma sessão é disponibilizado da seguinte forma apenas disponibilizando os campos *type*, *content*, *salt* e *HMAC*, onde o *type* será sempre *secure*, o *HMAC* respectivo ao *content* e a *chave derivada de sessão*, o *content* que possui o pedido original, o *salt* necessário para gerar a nova *chave derivada* oriunda da *chave de sessão* e necessária para decifrar o *content* e o *msgCode* para o controlo do fluxo de mensagens.



Help

Close

CONCLUSÃO

No decorrer do projecto, deparamo-nos com vários problemas e tivemos que escolher entre várias opções, inicialmente optamos pela utilização de chaves assimétricas RSA em vez de EC pois planeávamos fazer o *refreshing* das chaves guardando-as com o objectivo de poder definir tempos de vida destas associando a este tempo de vida as mensagens recebidas, com a possibilidade de renovar a chave privada é criada uma barreira de segurança temporal, caso uma chave privada seja ‘roubada’ apenas as mensagens associadas ao espaço temporal definido pela chave são comprometidas, e não perdendo mensagens após o refrescamento das chaves.

Para que se proceda ao estabelecimento de uma nova sessão, o servidor requer que o cliente assine o *challenge* de modo a garantir autenticidade por parte do cliente.

Recorremos ao Diffie-Hellman para estabelecer uma sessão e correspondente chave de sessão, optámos por nunca utilizar a chave de sessão directamente, derivando uma chave secundária da chave de sessão com a utilização de um *salt*, a cada pedido é gerado um *salt* novo e consequentemente uma chave derivada nova, garantindo que dois pedidos seguidos são cifrados por chaves diferentes.

Sendo a chave derivada uma chave simétrica, conseguimos a compressão dos dados, para o seu controlo de integridade achamos conveniente utilizar um HMAC do conteúdo comprimido e utilizando a chave derivada calculada, sendo esta diferente a cada pedido, garantindo assim a confidencialidade, autenticação e integridade das mensagens trocadas pelo par cliente-servidor após estabelecida uma sessão.

Tanto como no envio de mensagens entre clientes como no envio de um *receipt*, optámos pelo uso de cifras híbridas para novamente comprimir o texto a ser enviado com cifras simétricas e sob este texto cifrado é aplicado uma cifra assimétrica utilizando a chave publica do cliente destino. Estas mensagens são assinadas garantido a sua autenticidade e integridade, sendo que a confidencialidade é assegurada pela cifra híbrida.

Conseguimos a validação das cadeias de confiança, assinatura e autenticação, a correcta validação dos certificados e carregamento dos CRLs, validação das assinaturas (sha1 e sha256) e da data de assinatura no espaço temporal (not_before, not_after).

No projecto todas as assinaturas são feitas recorrendo à chave de autenticação uma vez que o grupo no desenvolvimento do projecto errou inúmeras vezes o *pin* durante o processo de autenticação e por erros de programação bloqueamos a assinatura, pela mesma razão não é feito o login na sessão do cartão de cidadão uma vez que o cliente pode assinar algo de forma não intencional sendo assim a responsabilidade de assinar atribuída ao utilizador.

De forma a completar o sistema de autenticação, no caso em que é renovado o cartão de cidadão, o sistema deveria ser capaz de fazer uma atualização guardando os novos certificados.

Assim apóis a conclusão deste projeto foi possível implementar um repositório de mensagens com a utilização de cifras híbridas, funções de síntese e assinaturas capazes de garantir respectivamente confidencialidade, integridade e autenticidade das mensagens enviadas.

REFERÊNCIAS & CODE SNIPPETS

Algumas ideias e respectivo código desenvolvido no projecto foi baseado nas seguintes documentações:

pycrypto

- <https://www.dlitz.net/software/pycrypto/api/2.6/>
- <https://www.dlitz.net/software/pycrypto/api/2.6/Crypto.Protocol.KDF-module.html>

pyopenssl

- <https://pyopenssl.org/en/stable/api/crypto.html>
- <http://pyopenssl.sourceforge.net/pyOpenSSL.html/openssl-x509.html>

pkcs11

- <https://pkcs11wrap.sourceforge.io/api/>
- <http://python-pkcs11.readthedocs.io/en/latest/applied.html>

Cores do Terminal

- <https://gist.github.com/minism/1590432>