universidade de aveiro

última revisão: 04/02/2017

SEGURANÇA

Ano Letivo 2017/2018

Secure Messaging Repository System

Milestone - Relatório

Professores: André Zúquete, andre.zuquete@ua.pt
João Paulo Barraca, jpbarraca@ua.pt

Grupo P1G4:

Ana Laura Costa Almeida	N.MEC.: 71661	*ana.laura.almeida@ua.pt*
Vítor Hugo Brandão Barros	N.MEC.: 68287	*vitor.hugo.barros@ua.pt*

Índice

Introdução

No âmbito da cadeira de Segurança do 4ºano de MIECT, foi-nos proposto implementar alguns tópicos de segurança num serviço seguro de comunicação entre clientes baseado num repositório de mensagens, que tem como intermediário um servidor (Rendezvous Point).

Este documento vai resumir a solução produzida. Inicia-se com uma breve descrição do projeto e a arquitetura da solução. De seguida apresenta-se a solução, com especial destaque na implementação das funcionalidades exigidas.

Descrição do Projeto

Este sistema de comunicação baseado num repositório de mensagens é composto por múltiplos clientes que utilizam um servidor para comunicar entre si.

O servidor guarda localmente alguma informação privada dos clientes de forma individual, tanto de identificação como de medidas de segurança necessárias.

Um cliente pode pedir ao servidor para criar uma message box, para listar as boxes dos outros clientes, as suas mensagens recebidas (todas ou apenas as novas), as suas mensagens enviadas e os comprovativos de entrega/leitura.

É, também, através do servidor que um cliente envia uma mensagem para outro cliente. Para este processo, o servidor vai guardar a mensagem no diretório de mensagens do cliente de destino, e uma cópia no diretório de mensagens enviadas do utilizador.

As mensagens, quer entre cliente e servidor, quer entre dois clientes, são privadas, sendo por isso necessário garantir a sua confidencialidade(assegurar que não é lida por terceiros), a sua integridade(assegurar que não é modificada por terceiros) e a sua autenticidade(assegurar que não é enviada por impostores).

É ainda necessário estabelecer uma associação única entre um utilizador e um Cartão de Cidadão e envolvê-lo em todo o processo de identificação do cliente.

Arquitectura

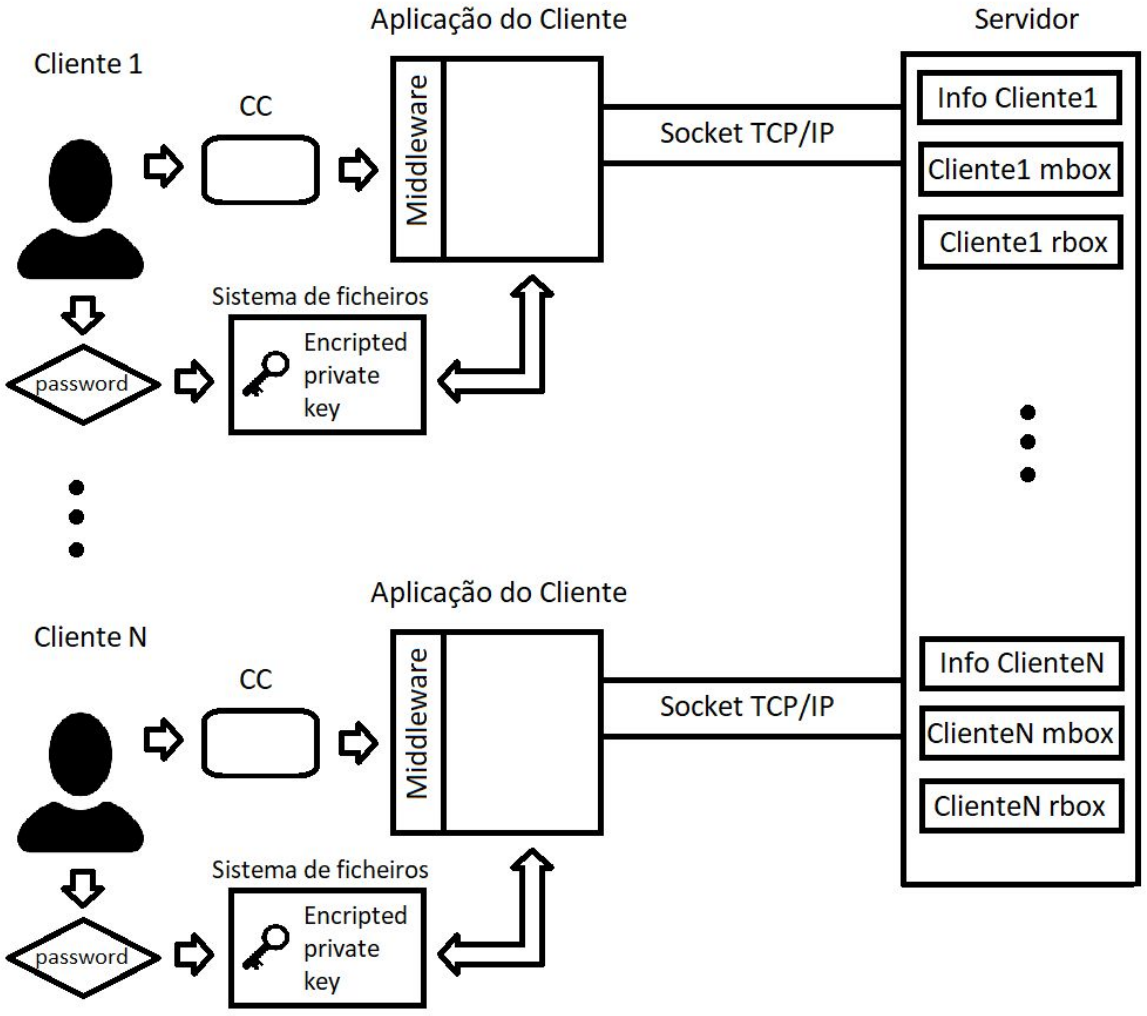


Figura 1: Arquitetura da solução a desenvolver.

A aplicação do cliente foi desenvolvida em Python, assim como o servidor. O cliente recorre ao seu Cartão de Cidadão para se autenticar na aplicação, e foi usada a biblioteca PKCS #11 library como middleware para efetuar a comunicação entre a aplicação e o smartcard.

A aplicação e o servidor comunicam através de mensagens em formato JSON por um stream Socket TCP/IP (bidirecional, o que significa que tanto um como outro podem enviar e receber mensagens) e o servidor irá guardar alguma informação do cliente, da primeira vez que o cliente se conectar a ele. Esta informação não pode ser acedida diretamente por nenhum utilizador, apenas alguma informação pode ser transmitida como resposta a pedidos dos clientes, mediante o que é permitido pela ideologia do projeto.

A aplicação socorre-se, ainda, do sistema de ficheiros do utilizador para guardar informação de segurança que será necessária utilizar em qualquer sessão que o cliente estabeleça.

O servidor possui um par de chaves assimétricas que vão ser usadas para a assinatura das mensagens trocadas com os clientes. Existe ainda um certificado dessa chave pública e a respetiva cadeia de certificação para validar as assinaturas. Esses certificados foram gerados com base na criação de uma Entidade Certificadora através do software XCA.

Funcionalidades implementadas

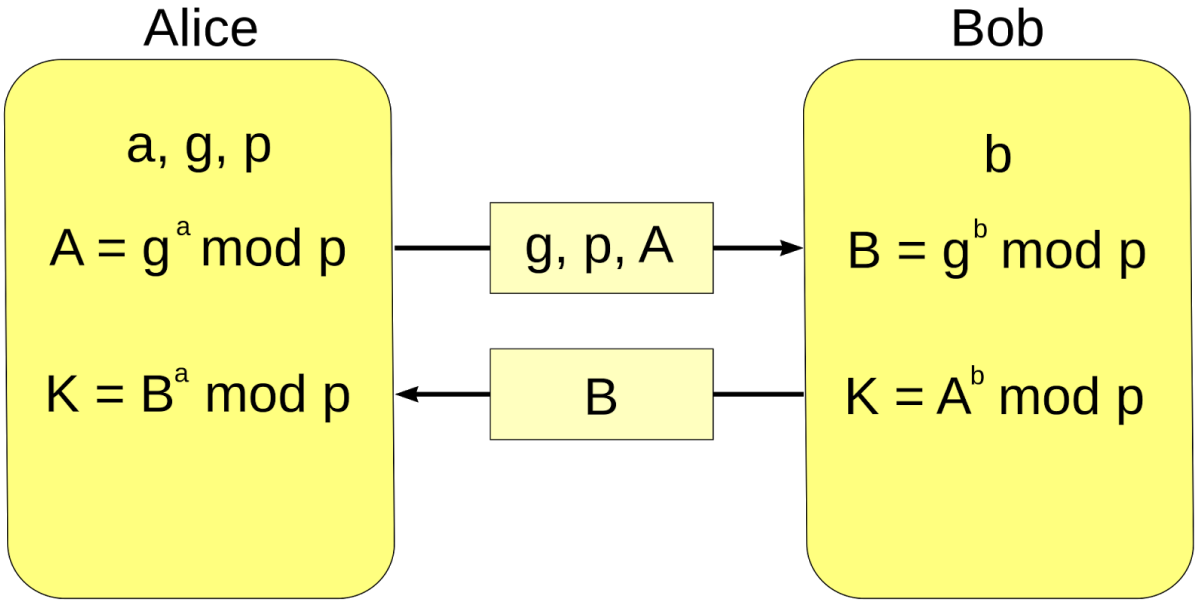
Início

Quando um novo cliente inicia a aplicação, é-lhe pedido um nome apenas para simplificar a identificação da respetiva chave privada gerada guardada localmente.

De seguida, o cliente estabelece a comunicação com o servidor pelo socket e o servidor inicia o processo de estabelecimento da chave de sessão com o cliente, antes de poder ser trocada qualquer mensagem entre os dois.

Chave de sessão

Foi escolhido o método Diffie-Hellman Key Exchange para realizar esse processo, pois permite que duas entidades que não possuem qualquer conhecimento *a priori* uma da outra, troquem valores entre si sob um canal de comunicação inseguro de modo a chegarem a uma chave simétrica secreta que pode ser usada para cifrar as mensagens posteriores. Uma vez que o Diffie-Hellman está sujeito a ataques Man in the Middle, é preciso assinar alguma mensagem trocada entre o cliente e o servidor durante o processo para assegurar a autenticação dos agentes de comunicação e assim prevenir esses ataques.



$K = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p = (g^b \bmod p)^a \bmod p = B^a \bmod p$

Figura 2: Diffie-Hellman Key Exchange

O servidor inicia o processo de estabelecimento da chave de sessão e gera *a, g* (base)*, p* (número primo) e calcula *A* a partir de *a* e *p*.

A é assinado com a chave privada do servidor e enviado para o cliente juntamente com o certificado da chave pública da respetiva chave privada, assim como *g* e *p* para o cliente poder calcular *B*.

O cliente verifica a assinatura de *A* e, se válida, prossegue com o algoritmo.

Após calcular *B*, envia o resultado para o servidor assinado com a sua chave privada de assinatura do cartão de cidadão, juntamente com o certificado da chave pública da respetiva chave privada para que o servidor possa também verificar a validade dessa assinatura.

Se a assinatura for válida, o servidor calcula K e envia uma mensagem “OK” para o cliente.

Se o cliente receber a mensagem e tiver conseguido calcular K sem qualquer problema, responde igualmente com “OK” e a chave de sessão está estabelecida.

Adicionar um genuiness warrant

Para garantirmos que a resposta que o cliente recebe do servidor é a correta ao seu pedido, estamos a gerar um *sequence number* aleatório por mensagem do cliente e a enviá-lo juntamente com a mensagem. O servidor, quando responde ao cliente, incrementa esse *sequence number* de uma unidade e envia-o na resposta. Assim, quando o cliente recebe a resposta, apenas tem de comparar se esse *sequence number* corresponde ao que foi enviado + 1.

Assegurar a autenticação e a integridade de todas as mensagens trocadas entre o cliente e o servidor com a chave de sessão

A autenticação das mensagens trocadas entre o cliente e o servidor é assegurada pela assinatura de um campo da mensagem JSON. Quando se trata de uma mensagem de *erro* ou de *list* sem um id em específico, por exemplo, onde só existe o tipo da mensagem ou o erro, e o *sequence number*, o campo que é sempre assinado é o do *sequence number*. Quando se tratam de mensagens com um campo *msg*, é esse campo que é assinado.

A assinatura é feita através da chave privada de assinatura do cartão de cidadão, no caso do cliente, e com a chave privada gerada pelo XCA, no caso do servidor. Juntamente com a mensagem são enviados os certificados das respetivas chaves públicas para que a outra entidade possa validar a assinatura. Os certificados necessários para construir a cadeia de certificação e, assim, validar também esse certificado enviado, encontram-se no diretório *CCCerts/certs* e as CRLs no diretório *CCCerts/crls*.

Uma vez que o certificado necessário para validar a assinatura da mensagem está a ser enviado a cada mensagem e uma vez que a validação da assinatura se baseia na validação do certificado e da data em que a assinatura foi feita, não existe nada que verifique se o certificado enviado corresponde à mesma entidade com a qual se estabeleceu a chave de sessão, e não a alguém que entretanto se tenha apoderado do canal de comunicação e que envie mensagens assinadas com o seu cartão de cidadão e

com o certificado para validar a sua assinatura. Assim sendo, após o estabelecimento da chave de sessão, o servidor guarda o *Common Name* do certificado usado para validar a assinatura do cliente nesse momento, e a cada mensagem que receba compara o *Common Name* do certificado recebido com o que ficou guardado para aquele cliente. Por sua vez o cliente guarda o *Common Name* do certificado do servidor.

Relativamente à integridade das mensagens, decidiu-se usar *HMAC*, um autenticador de mensagens que gera uma síntese de uma mensagem a partir de uma chave simétrica secreta partilhada pelas duas entidades e da própria mensagem. A chave simétrica utilizada foi a chave de sessão. Não achámos necessário usar uma derivada da chave de sessão pois gera-se uma nova chave por cada sessão e as sessões são muito breves, o que dificulta a descoberta da chave de sessão num tempo tão limitado. Além disso, após haver a suspeita de que o canal de comunicação está comprometido, este é fechado e é estabelecido um novo com uma nova chave de sessão.

A mensagem a partir da qual se gera a cifra é a mensagem JSON por inteiro. Antes de se calcular a síntese, esta mensagem é convertida para base64 de modo a garantir que a síntese é feita sobre um tipo objeto que não sofre alterações quando é enviado e recebido pelos sockets (ao contrário do JSON, que não garante que os campos recebidos estejam pela mesma ordem com que foram enviados). Deste modo, decidiu-se encapsular a mensagem enviada num outro JSON, cujos campos seriam a mensagem JSON em base64 e o hmac calculado:

```
{ type: secure,
  payload: mensagem em base64,
  hmac: síntese calculada }
```

A síntese é validada pelo servidor (ou pelo cliente, no caso de ser uma resposta do servidor), calculando outra síntese a partir do payload (sem converter do formato de base64) e da chave de sessão e comparando com a hmac recebida. Se corresponder, a síntese é validada. Caso contrário, indica-se que há suspeita de intrusos no canal de comunicação e encerra-se o socket, abrindo um novo e estabelecendo uma nova chave de sessão.

Guardar informação importante, relacionada com segurança, no processo de criação de um novo user

É guardado o uuid fornecido pelos users, de forma a poder reconhecer o utilizador caso este se desconecte do servidor e se volte e conectar.

É também guardada a chave pública dos clientes para que o servidor a possa fornecer a um cliente que queira comunicar com outro. Essa chave pública serve para cifrar as mensagens trocadas entre clientes, de modo que só o cliente de destino possa ler a mensagem depois de a decifrar com a sua chave privada.

Envolver o Cartão de Cidadão no processo de criação do user

O uuid do cliente é uma síntese do certificado da chave pública de autenticação do cliente. O cliente envia o certificado ao servidor numa mensagem do tipo *Create*, que o valida através da cadeia de certificação e do *Common Name.* Se o certificado for válido,

calcula e guarda o uuid. Se não for, indica-se que há suspeita de intrusos no canal de comunicação e encerra-se o socket, abrindo um novo e estabelecendo uma nova chave de sessão.

Request ID/Create user message box

Quando aparece um novo cliente, este deve enviar uma mensagem ao servidor do tipo *Create* para se registar no servidor.

Neste caso, antes de se enviar a mensagem, é gerado o par de chaves assimétricas usados para a cifra e decifra de mensagens entre clientes, de forma a poder enviar na mensagem *Create* a chave pública para ser guardada no servidor (A chave privada fica guardada num ficheiro local do lado do cliente, encriptada com uma password escolhida pelo utilizador. Para este processo foi utilizada a biblioteca getpass do Python, que permite não apresentar a password em cleartext no terminal).

No caso de se tratar de um user que já foi criado e que abriu um novo socket, não é possível fazer um novo registo, e por isso criou-se um novo tipo de mensagens, *Request ID*, que envia o certificado a partir do qual se calculou o uuid na criação do user.

O servidor, perante este tipo de mensagens, deve validar o certificado e comparar se corresponde ao *Common Name* guardado para aquele user. Se corresponder, o user é validado e o servidor responde com o id atribuído às suas mbox e rbox. Do lado do cliente, este consulta o ficheiro, onde está guardada a chave privada, perante autorização com a validação da password escolhida. Caso não corresponda, a comunicação é fechada.

Cifrar mensagens enviadas entre clientes

As mensagens trocadas entre utilizadores são cifradas através de cifras híbridas. O campo *msg* é cifrado por uma chave simétrica gerada pelo cliente que quer enviar a mensagem. A chave simétrica é enviada e guardada juntamente com a mensagem no servidor, pelo que deve ser escondida para que, no caso do servidor ficar comprometido, não tenham acesso à chave para decifrar as mensagens. Assim, a chave simétrica da *msg* é cifrada pela chave pública do cliente de destino, obtida através da resposta do servidor a um request do tipo *List*.

Cifrar mensagens guardadas na receipt box

As mensagens guardadas na receipt box são as cópias das mensagens e os receipts enviados pela leitura das mensagens enviadas. Os receipt não são cifrados porque não requerem confidencialidade, mas sim autenticidade. No caso das cópias das mensagens, estas são cifradas com recurso a cifras híbridas, tal como a mensagem enviada. A única diferença é que a chave simétrica gerada é diferente e a chave com que se cifra a chave simétrica é a chave pública do emissor da mensagem.

Para concluir, um exemplo de uma mensagem *Send* seria:

```
{ "type": "send",
  “sn”: 1000,
  "src": 1,
```

"dst": 2,
"msg": mensagem cifrada com a chave simétrica K_X (base64),
“msgKey”: K_X cifrada com a chave pública do cliente do destino,
"copy": mensagem cifrada com a chave simétrica K_Y(base64),
“msgKey”: K_Y cifrada com a chave pública do cliente,
“sign”: campo msg assinado com a chave privada do Cartão de Cidadão,
“cert”: certificado da chave pública correspondente à chave privada da assinatura,
“datetime”: data da assinatura }

Assinatura das mensagens trocadas entre utilizadores com o Cartão de Cidadão e validação da mesma

As mensagens trocadas entre utilizadores são assinadas através da chave privada de assinatura do cartão de cidadão do emissor e são validadas pelo recetor através do respetivo certificado de chave pública (enviado juntamente com a mensagem).

Enviar um receipt seguro depois de se ler uma mensagem

De cada vez que se lê uma mensagem, a aplicação envia um receipt automaticamente. O emissor do receipt assina, com a chave privada de assinatura do cartão de cidadão, a mensagem de resposta do servidor ao pedido de leitura da mensagem, que inclui a mensagem em si e a assinatura do servidor da resposta ao pedido. Assim, numa mensagem do tipo *Receipt* é enviada a assinatura da resposta do servidor e a própria resposta para se poder validar a assinatura, a data da assinatura e o certificado para a sua validação.

Validar receipts de mensagens enviadas

Para validar o(s) campo(s) receipt no resultado de uma mensagem STATUS valida-se a assinatura do user no receipt, tendo em conta a data e o certificado fornecido. De seguida, valida-se a assinatura do servidor na sua resposta (é enviado num campo do receipt) para garantir que foi o servidor que forneceu a mensagem para ser lida num determinado instante temporal, e por último verifica-se se o id do receipt corresponde ao destinatário da mensagem, para garantir que o receipt foi enviado pelo destinatário da mensagem que foi enviada.

Validação dos certificados de chaves públicas do Cartão de Cidadão

A validação das chaves públicas (do cartão de cidadão ou do servidor) é feita através da criação da cadeia de certificação desse certificado e da validação do mesmo face a essa cadeia de certificação. Para além da verificação anterior, verifica-se ainda se o

serial number desse certificado de chave pública não se encontra em nenhuma lista de certificados revogados.

As assinaturas com o cartão de cidadão são validadas através da biblioteca M2Crypto. As assinaturas do servidor são validadas através da biblioteca PyOpenSSL.

Impedir que um user leia mensagens de outra message box que não seja a sua

O servidor compara o ID enviado na mensagem do tipo *All* ou *New* com o ID do Cliente que o servidor tem associado ao socket por onde recebeu a mensagem. Se corresponder, o servidor deixa o user consultar a message box. Caso contrário, não deixa e considera que a comunicação foi comprometida, e fecha a conexão ao cliente.

Impedir que um user envie um receipt de uma mensagem que ele não leu

O receipt é construído em cima da resposta do servidor a um pedido de leitura de mensagem, e é enviado automaticamente pela aplicação após a decifra da mensagem.

Cifras escolhidas

Utilizámos RSA para as cifras assimétricas, pois é considerada uma das cifras mais seguras e o seu algoritmo não permite a descoberta das chaves por funcionamento inverso.

Para as cifras simétricas escolhemos o algoritmo AES, pois apresenta um bom desempenho, flexibilidade, alta resistência a ataques e exige pouca memória.

Temos ainda as funções de hash disponibilizadas pela biblioteca PyCrypto para calcular as sínteses necessárias à verificação da integridade das mensagens.

Conclusão

Consideramos que as funcionalidades exigidas foram implementadas com sucesso, e que abordámos ainda outros aspetos de segurança em programação que tornam uma aplicação mais robusta.

Como aplicação exemplo, conseguimos demonstrar os aspetos fundamentais de segurança relativamente a trocas de informação entre um cliente e servidor e ainda como lidar em caso de falha de segurança num dos canais de comunicação protegidos.