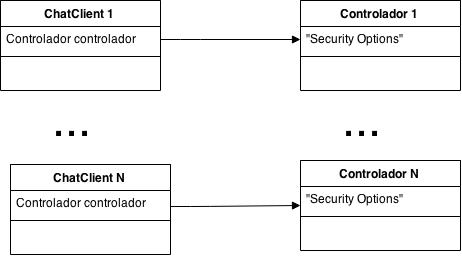
|  |
| --- |
| Universidade de Coimbra |
| Relatório STI |
| Trabalho Prático III |
|  |
| **Erbi Silva e João Batista** |
| **02/05/2015** |

|  |
| --- |
|  |

**Classe “Controlador”**

****

O sistema programado lança diversos clientes (“ChatClient”). Cada cliente terá um controlador que será responsável por controlar todas as acções que esse cliente precisa tais como, encriptação das mensagens, , manter a integridade das mensagens, entre outras, basicamente controla toda a segurança do cliente.

Classe “Mensagem”

A classe mensagem foi criada para transportar a mensagem encriptada, a chave, o ID de quem a envia e a “hash” para controlar se a mensagem foi ou não alterada pelo caminho.

Confidencialidade

A confidencialidade é garantida com encriptação e desencriptação das mensagens. Após o cliente escrever a mensagem, o controlador irá ser responsável por encriptar e enviar a mensagem do cliente. Para isso, cada cliente tem uma chave que é renovada de X em X tempo para garantir mais segurança ao utilizador. Assim sendo, são utilizados os seguintes algoritmos:

|  |  |
| --- | --- |
| **Gerar a chave:** | AES |
| **Encriptar:** | AES/CBC/PKCS5Padding |
| **Desencriptar:** | AES/CBC/PKCS5Padding |
| **Tamanho:** | 16 bytes |

Existe uma classe Mensagem que é responsável por transportar a mensagem encriptada bem como a chave de quem envia a mensagem.

A chave é criada com o seguinte algoritmo:

|  |
| --- |
|  |

Algoritmo Encriptação

O algoritmo de encriptação que utiliza a chave acima gerada é o seguinte:

|  |
| --- |
|  |

Algoritmo Desencriptação

O algoritmo de desencriptação que utiliza a chave acima gerada é o seguinte:

|  |
| --- |
|  |

Autenticidade/Integridade

Para garantir estes mecanismos de segurança, a mensagem guarda uma hash que é criada logo após o cliente envia a mensagem. Esta hash é gerada de acordo com o corpo da mensagem de quem a envia para garantir toda esta segurança. O algoritmo que cria a hash é:

|  |
| --- |
|  |

Manutenção da chave

Para garantir a manutenção da chave, esta é renovada de X em X tempo graças a uma thread que o servidor lança. Essa irá apagar todas as chaves anteriores, gerar novas, guardar as mesmas e é responsável por enviar uma mensagem a cada cliente indicando-lhe a sua nova chave.

|  |
| --- |
|  |

Non-repudiation

Para garantir “non-repudiation” cada cliente tem um par de chaves. O cliente encripta a sua mensagem com a sua chave privada e, o receptor receberá a chave pública para desencriptar a mesma.

Assinatura para encriptar:

|  |
| --- |
|  |

Com este pedaço de código o utilizador instancia uma “Signature” e cria uma fábrica de chaves. Isto para que se possa ir buscar a chave privada e, criar uma assinatura com a chave privada sobre a mensagem deste utilizador.

Assinatura para desencriptar:

|  |
| --- |
|  |

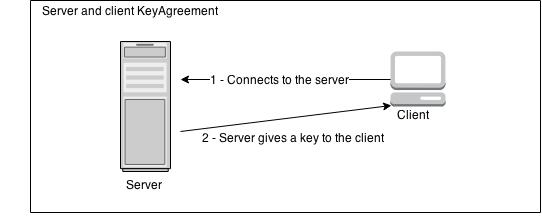
Aqui podemos ver que o código faz praticamente o mesmo mas, vai buscar a chave pública para verificar se combina com a assinatura.

**Modelos Segurança**

Neste capítulo vão ser descritos e ilustrados os modelos de segurança utilizados:

**Criação de chaves**

Sempre que um cliente se liga ao servidor, este faz um acordo de chaves com cada cliente. Isto é, o servidor é responsável por dar uma chave que ficou acordada entre ele e o cliente. Assim sendo, cada cliente terá uma chave para encriptar e, o servidor quando receber uma mensagem de determinado cliente, saberá sempre o que fazer com a mesma.

****

No exemplo do nosso projecto, o cliente encripta a mensagem com a sua chave e envia a mensagem. À chegada do servidor, o mesmo reconhece quem envia a mensagem e, antes de enviar para o destinatário, vai encriptar a chave de remetente com a chave do destinatário. Assim, caso a mensagem seja apanhada a meio, a chave estará sempre segura.

Note-se que o servidor guarda todas as chaves mas, também as encripta com uma chave própria do servidor.

**Renovação das chaves**

A renovação das chaves é feita de X em X tempo pelo servidor. Este é responsável por criar uma mensagem que será enviada para cada cliente. Estes, ao identificarem essa mensagem, receberão os dados da nova chave e irão actualizar os dados de si próprios.

**Testes**

Para garantir que todas as funcionalidades estão bem implementadas, foi criada uma classe “Ataques” que é responsável, por exemplo, de alterar o conteúdo de uma mensagem.

1

2

**Chaves encriptadas no servidor**

Com isto, o primeiro teste é para provar como o servidor guarda as chaves encriptadas, pelo que mostramos o seguinte exemplo com um servidor e um cliente:

|  |
| --- |
|  |

O servidor recebeu uma mensagem do cliente. Então, o servidor pegou na chave encriptada (1) e desencriptou a mesma (2). Como vemos os resultados são diferentes. O mesmo acontece com privadas e públicas.

**Alterar a mensagem pelo caminho e comprovar que é diferente da original**

No controlador, basta colocar o false a true:

|  |
| --- |
| private static Boolean ALTERA\_MENSAGEM\_PARA\_MD5 = true; |

De seguida correr o cliente e enviar uma mensagem. O resultado é o seguinte:

|  |
| --- |
|  |

**NOTA:** Aparece que a assinatura é válida uma vez que nesta situação a mensagem só está a ser alterada, depois e só depois de verificar a assinatura.

**Alterar assinatura da mensagem**

No controlador, basta colocar o false a true:

|  |
| --- |
| private static Boolean ALTERA\_MENSAGEM\_ASSINATURA = true; |

De seguida correr o cliente e enviar uma mensagem. O resultado é o seguinte:

|  |
| --- |
|  |

**Alterar chave da mensagem a meio**

No controlador, basta colocar o false a true:

|  |
| --- |
| private static Boolean ALTERA\_MENSAGEM\_PARA\_MD5 = true; |

De seguida correr o cliente e enviar uma mensagem. O resultado é o seguinte:

|  |
| --- |
|  |

**NOTA:** Aparece que a assinatura é válida uma vez que nesta situação a chave só está a ser alterada, depois e só depois de verificar a assinatura.