TÉCNICO LISBOA

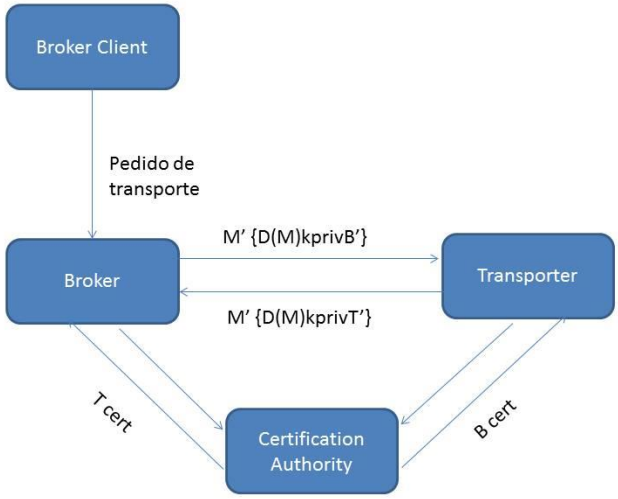
Sistemas Distribuídos 2015-2016

T23

https://github.com/tecnico-distsys/T_23-project

		
Rafael Koener	Fernando Liça	João Ribeiro
76475	77207	77209

1 O Problema da Segurança



Racional – A nossa solução

Para garantir a autenticação e não repudiação das mensagens, decidiu-se utilizar assinaturas de chave públicas.

No inicio do sistema todas as entidades têm uma keystore, onde se encontram a sua chave publica e privada, a CA tem os certificados de todas as entidades, que contêm as suas chaves publicas.

Quando uma entidade envia uma mensagem a outra, digamos uma mensagem M correspondente ao corpo (body) de um envelope SOAP, o seu AuthenticantionHandler, irá pegar na mensagem, converter M de xml SOAP para String (Java), de seguida converter a mesma para byte array (byte[]), gerar um resumo (digest D(M)) e cifrar o mesmo com a sua chave privada ({D(M)}KprivEm), o que garante autenticidade, integridade e não repúdio.

Quando a entidade recetora recebe esta mensagem, o seu AuthenticantionHandler vai agora verificar se é de facto, autêntica.

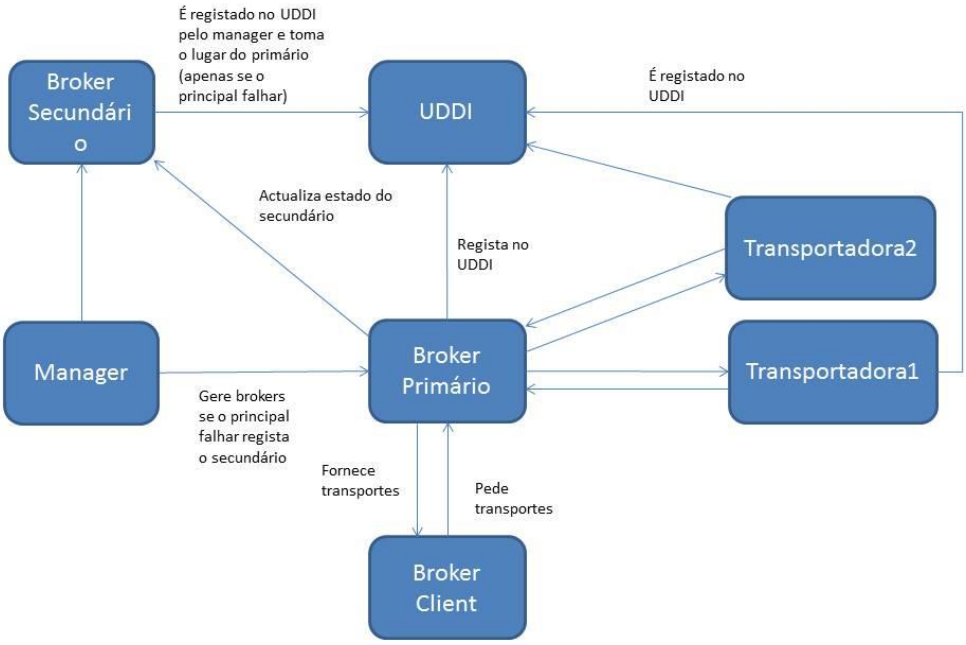
Começa por retirar o body do envelope (M'), acede ao header onde está o nome do emissor e o resumo de M cifrado, converte M' de xml SOAP para String, de String para byte array, e gera o resumo do mesmo (D(M')). Após ter o resumo de M cifrado, irá contactar a autoridade CA, para obter o certificado do emissor. Após obter o certificado, verifica se é autêntico a partir da sua cópia do certificado da CA, e caso o seja, retira de lá a chave pública do emissor.

Com a chave pública do emissor, decifra o resumo de M cifrado e obtém o resumo de M (D(M)).

Com o resumo de M que decifrou e o resumo de M' que calculou, compara para ver se são de facto idênticos para saber se a mensagem é autêntica.

A nível de código, utilizaram-se as funções fornecidas pelo corpo docente - makeDigitalSignature(byte array da mensagem, chave privada) e verifyDigitalSignature(byte array do resumo, byte array da mensagem, chave pública emissor) – para realizar o mecanismo acima descrito.

2 O Problema da Tolerância a Faltas



A nossa solução

Um problema que pode existir nestes sistemas é a falha de um componente importante, que poderá por em causa o funcionamento do sistema como um todo.

Para resolver este problema criou-se um corretor secundário que serve de backup ao primário, e uma entidade "manager" que verifica se o servidor primário se encontra em execução através de pings (requisitados pelo manager, enviados pelo servidor primário) com intervalos de 3 segundos.

```
while(reply.equals("Hello Live Checker. Im a Broker") && _running){

        Thread.sleep(3000);
        reply = _primary.ping("Live Checker");
```

Sempre que o estado interno do servidor primário é alterado (lista de transportes, lista de transportadoras, e lista de trabalhos) é atualizado também no servidor secundário. Para tal acrescentou-se uma flag, na criação dos servidores, se o servidor em questão é primário ou secundário. Isto permite que ao alterar o estado interno do servidor primário, é possível atualizar o estado do servidor secundário recorrendo à função updateStatus(TransportView), que atualiza a lista de transportes reservados do servidor secundário, ou updateViewTransportStatus(String), que atualiza o estado de um dado transporte.

```
if(_primary){
    _secondary.updateStatus(transport);
}
```

Quando o manager não recebe resposta do servidor primário, faz deploy do servidor secundário, registando-o no servidor UDDI, tomando este, a partir desse momento, o papel de servidor primário.