






# Sistemas Distribuídos

2015/2016

Prof. Miguel Pardal – 2ªF – 11h00

Grupo

**A40**

		
<b>João Santos</b> 67011	<b>Diogo Ferreira</b> 79018	<b>Carina Martins</b> 79153
<a href="https://github.com/tecnico-distsys/A_40-project.git">https://github.com/tecnico-distsys/A_40-project.git</a>		

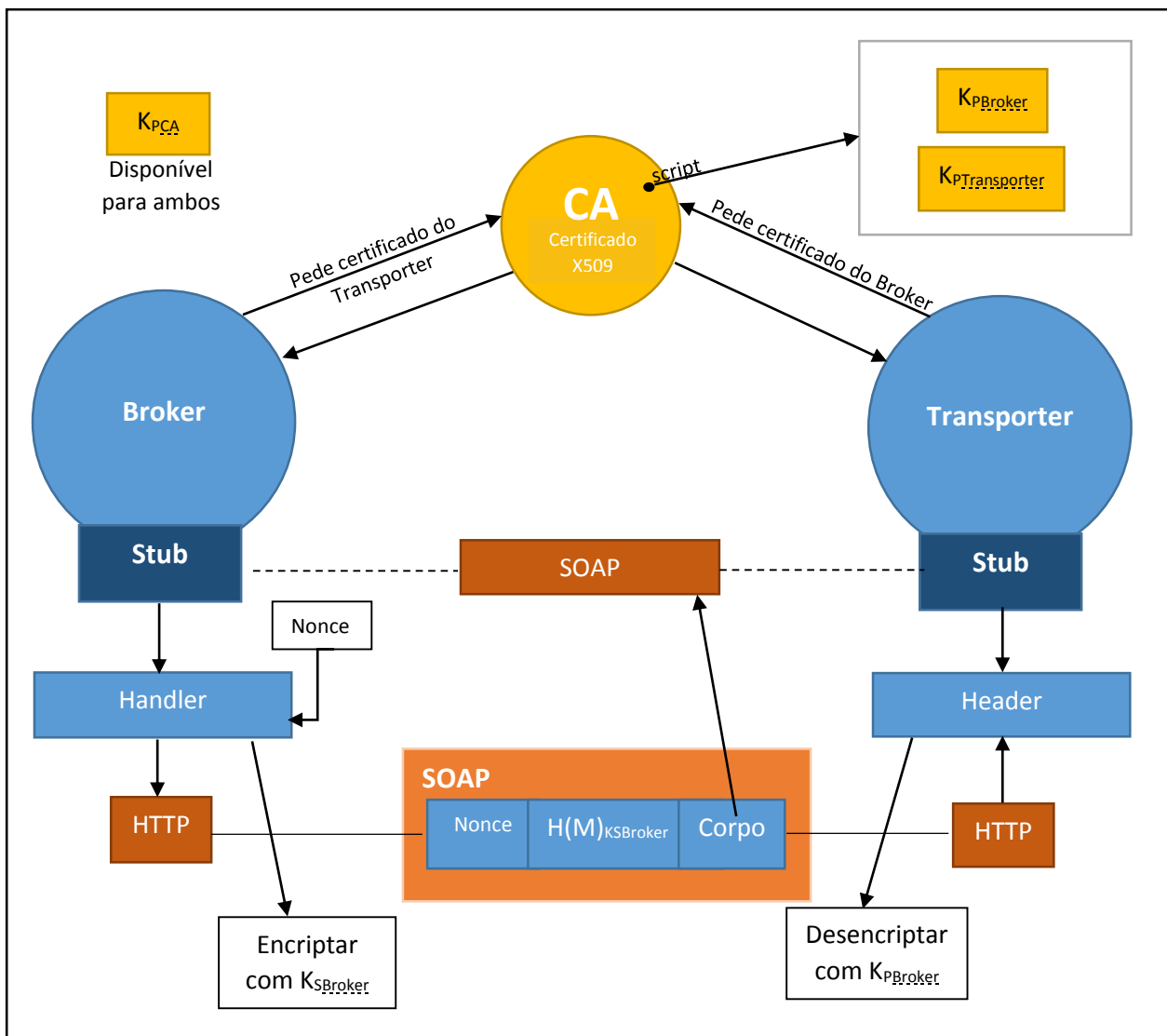


Figura – Esquema da solução de segurança

## AUTENTICIDADE

Para confirmar a fonte da mensagem, o recetor deve fazer o seguinte:

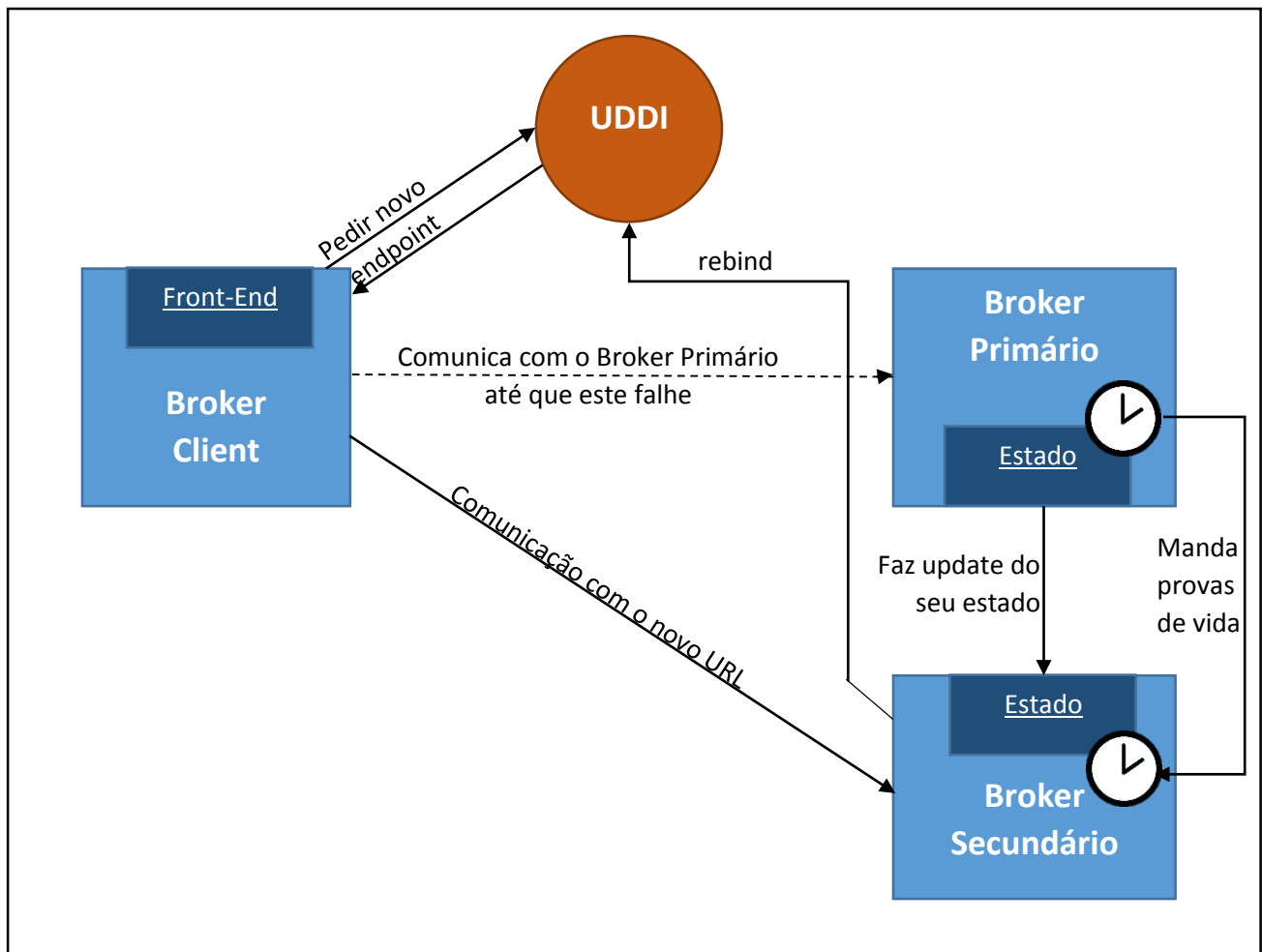
1. Pedir o certificado do emissor à CA;
2. Verificar se o certificado recebido está devidamente assinado pela CA, utilizando a chave pública desta.
3. Se (2) se verificar, confirmar se o *digest* assinado pelo emissor -  $H(M+N)_{K_{SEmissor}}$  – após descriptado com a chave pública dada pelo certificado do emissor, corresponde a fazer digest da mensagem recebida + *nonce*.
4. Caso (3) se verifique é porque o emissor é quem diz ser.

## NÃO-REPÚDIO

O não-repúdio consiste em garantir que a entidade que mandou a mensagem não possa negar que foi ela que efectivamente a mandou. Tal é conseguido **pela assinatura com a chave secreta** das mensagens.

## FRESCURA

A frescura das mensagens é garantida pela utilização de um *nonce* baseado na data atual. Admitiu-se que uma mensagem só é válida caso a diferença entre a *timestamp* na sua receção e no *nonce* é menor do que um minuto. **Guardam-se também todos os nonces atribuídos** para que não exista a possibilidade de os reutilizar. Com esta abordagem garante-se que não existe **replay attack**.



**Figura – Esquema da solução de tolerância a falhas (replicação do Broker)**

Para a replicação do Broker para tolerância a falhas, a nossa solução consiste em **lançar o broker primário (UpaBroker - B<sub>p</sub>) após a inicialização do broker secundário (UpaBrokerSec - B<sub>sec</sub>)**, pois foram criadas dependências para existir a garantia de que ambos existem quando se corre a aplicação pela primeira vez.

### IMPLEMENTAÇÃO DE UPDATES E PROVAS DE VIDA

Como sugerido nas aulas, o B<sub>p</sub> manda de 4 em 4 segundos uma prova de que ainda está a correr para o B<sub>sec</sub>, que espera receber essa prova de 6 em 6 segundos – este último *timeout* é ligeiramente maior, de modo a poder funcionar corretamente mesmo que ocorra um pequeno atraso nos Web Services na receção das provas de vida. Caso o tempo do B<sub>sec</sub> se passe, sem receber a prova de vida do B<sub>p</sub>, o B<sub>sec</sub> **toma controlo através de um pedido de *rebind* ao UDDI** – mantém o seu URL, mas muda o seu nome para *UpaBroker* (o nome do primário).

Enquanto o B<sub>p</sub> se mantém vivo, este também manda update do resultado das suas operações para o B<sub>sec</sub> ficar atualizado na ocorrência de falta. Este novo método foi definido no WSDL, **e leva como argumentos a operação que foi realizada e a TransportView nova/modificada correspondente**.

Tanto o método de update – UpdateSecondaryBroker – como o método de provas de vida – ImAlive – são **operações unidirecionais**, pois não esperam respostas.

### IMPLEMENTAÇÃO DO FRONT-END NO CLIENTE

Para a concretização da tolerância de falhas de comunicação com o servidor, foram utilizados os dois *timeouts* do JAX-WS: *connection* e *receive*. **Caso qualquer um destes timeouts passe, sem que ocorra a resposta ao pedido do cliente, é apanhada uma exceção e é pedido um novo endereço de endpoint ao UDDI**. O respetivo pedido do cliente é, então, novamente realizado, mas desta vez ao B<sub>sec</sub>.