









T_19_29

 <p>68593 – André Pires</p>	 <p>70483 – Lucas Terto</p>	 <p>70554 – Alexandre Pimentel</p>
 <p>70571 – Filipe Apolinário</p>	 <p>71037 – Filipe Bento</p>	 <p>73893 – Hugo Martins</p>

Índice

CHEQUE-REFEIÇÃO	3
Detalhes da Implementação	3
Protocolo Base	3
Protocolo de Endosso	4
Protocolo de Endosso Desligado.....	4
REGISTO-FATURA	5
Descrição da Implementação.....	5
Tolerância a Faltas	5
Replicação	6

CHEQUE-REFEIÇÃO

Detalhes da Implementação

A nossa implementação começou por alterar a maneira como as mensagens são comunicadas com o objectivo de evoluir o projecto e facilitar as futuras comunicações quando estivessem implementados ambos os protocolos e as respectivas dependências – cifras e assinaturas digitais.

A comunicação foi por isso alterada de maneira a que as mensagens trocadas pudessem ser alteradas, tanto ao receber, quanto ao enviar. Isto foi feito de maneira simples, adicionando um *handler* do lado do servidor e outro do lado do cliente. Desta forma, foi possível alterar as mensagens enviadas, de forma a tornar a comunicação segura, e as mensagens recebidas, verificando se as mensagens não foram corrompidas ou alteradas.

O corpo de cada mensagem SOAP, ao ser enviada, é encriptado com uma chave simétrica gerada no *handler* ao enviar uma mensagem. Esta chave é depois encriptada e adicionada ao cabeçalho da mensagem, sendo tal encriptação feita com a chave pública do utilizador ou servidor dependendo da sua origem. Após essa encriptação, cada mensagem é assinada com a chave privada do utilizador/servidor que enviar a mensagem, criando um resumo. Esse resumo é anexado ao envelope SOAP para que se possa verificar a sua validade aquando da leitura.

Quando uma mensagem é recebida, é verificada a validade da assinatura para garantir que a mensagem não foi alterada desde que foi enviada e, caso a validação seja bem sucedida, é então desencriptada a chave simétrica recebida pelo cabeçalho da mensagem, com a chave privada do utilizador ou do servidor, a depender do remetente. Posteriormente a mensagem é desencriptada com tal chave simétrica para ser lida no seu formato original.

Para que este processo seja possível, e sabendo que as chaves assimétricas são conhecidas *a priori* cada um dos programas **chequerefeicao-ws** e **chequerefeicao-ws-cli** têm, respectivamente, a chave privada do servidor e as chaves públicas dos utilizadores registados, e a chave pública do servidor e as chaves privadas dos utilizadores registados.

Para além da cifras e assinaturas digitais, dentro do corpo do envelope SOAP (a parte cifrada) está também uma tag específica que indica o nome do utilizador que fez o pedido que depois é comparado com o nome do utilizador cuja chave foi usada para garantir que o utilizador que fez o pedido é legítimo.

Protocolo Base

Este protocolo foi implementado em pleno.

1. O Titular pede para emitir um cheque: garante-se que CR é capaz de confirmar que foi T quem escreveu o pedido e que o mesmo não foi modificado através da assinatura digital e da verificação de utilizadores descritas no tópico anterior; garante-se que apenas CR consegue ler o pedido cifrado com a sua chave pública sendo CR o único possuidor da sua chave privada.
2. O CHEQUEREFEICAO emite o cheque: garante-se que o segredo foi gerado de forma não previsível usando objectos da classe *SecureRandom* do Java; garante-se que CR foi quem escreveu o pedido e que o mesmo não foi modificado através da assinatura digital e da verificação de utilizadores descritas no tópico anterior; garante-se que apenas T consegue ler o pedido cifrado com a sua chave pública sendo T o único possuidor da sua chave privada.
3. O Titular apresenta o número de cheque como pagamento ao Beneficiário: passo manual.
4. O Beneficiário saca o cheque: garante-se que B foi quem escreveu o pedido e que o mesmo não foi modificado através da assinatura digital e da verificação de utilizadores descritas no tópico

anterior; garante-se que apenas CR consegue ler o pedido cifrando o pedido com a sua mensagem pública sendo CR o único possuidor da sua chave privada.

5. O CHEQUEREFEICAO marca o cheque como usado e devolve o seu valor: garante-se que CR foi quem escreveu o pedido e que o mesmo não foi modificado através da assinatura digital e da verificação de utilizadores descritas no tópico anterior; garante-se que apenas B consegue ler o pedido cifrando o pedido com a sua mensagem pública sendo B o único possuidor da sua chave privada.

Protocolo de Endosso

Este protocolo foi implementado em pleno.

A segurança garante-se implementando a cada passo os mesmos sistemas que no protocolo base. Se fragmentarmos este protocolo obtemos:

1. Comunicação de mensagem. Mensagem esta que vai encriptada e assinada de maneira que são garantidos os requisitos que especificam que foi T quem escreveu o pedido e que apenas CR consegue ler o pedido e que o mesmo não foi mudado.
2. Gera um novo id+segredo. Operação semelhante à da emissão de um CHEQUEREFEICAO cujos requisitos de segurança garantimos no tópico anterior.
3. Os restantes passos são manuais.

Protocolo de Endosso Desligado

Este protocolo foi implementado em pleno.

O protocolo de endosso desligado é feito a partir de uma invocação do método remoto endossar do **chequerefeicao-ws**, quando este se encontra desligado.

Quando o cheque é endossado *offline*, o *ClientHandler* modifica a mensagem SOAP que vai ser enviada como descrito no tópico acima, sendo passada ao cliente com auxílio do atributo *endossarOffline* do *ClientHandler* (1). A mensagem retornada, guardada no atributo, serve como comprovativo que o cheque foi endossado, sendo possível, posteriormente, sacar o cheque apresentando tal comprovativo (2). Para sacar o cheque a partir do comprovativo, foi modificado o método de sacar para poder receber o comprovativo como se fosse o id+segredo. O método sacar, internamente, analisa a segurança do comprovativo (3), assinatura e encriptação, e a validade do comprovativo (4).

Caso o comprovativo seja aceite é sacado o cheque, caso contrário é retornada uma exceção de cheque inexistente.

1. Foi optado por enviar a resposta pelo *endossarOffline* visto que a invocação de um método remoto *offline*, lança uma exceção conexão remota falhada.
2. Para facilitar o processo de endossar o cheque foi criado um objecto *EndossarOffline* que abstrai todo o processo de invocar o endossar do **chequerefeicao-ws** e receber a mensagem SOAP Enviada. Para do ponto de vista de abstracção este objecto escreve a mensagem SOAP num ficheiro especificado em argumento para facilitar o processo de teste e demonstração.
3. Utilizado objecto auxiliar, *VerificadorMsg*, que verifica se o comprovativo é uma mensagem SOAP e se é uma mensagem segura, seguindo a mesma metodologia do *handleMessage* do *handler.Java*.
4. O comprovativo só é válido se o titular no comprovativo for efetivamente o dono do cheque e se o cheque é endossável.

REGISTO-FATURA

Descrição da Implementação

A nossa solução consiste numa lista ligada de gestores de réplicas, onde o primário é a cabeça da lista e os secundários são inseridos no *fim da lista* pela ordem com que são iniciados. A inserção é feita através do envio de uma mensagem SOAP ao endpoint do primário (descoberto com um lookup ao UDDI), que lhe pergunta qual o secundário (endpoint) a que está ligado. Ao obter o secundário que vem a seguir na lista, é novamente perguntado a esse secundário qual o próximo, e assim sucessivamente, até atingir o fim da lista.¹

Quando o último secundário recebe a mensagem SOAP que lhe pede o próximo endpoint, verifica que não aponta para ninguém e passa a apontar para o secundário que se inicia. Ao fazê-lo, lança uma thread que começa o envio de mensagens SOAP “I’m Alive” para este, e envia-lhe uma mensagem SOAP para o notificar que é agora o último elemento. O secundário que se inicia, ao receber esta mensagem, lança uma thread que aguarda por mensagens SOAP “I’m Alive” com um timeout definido.

Com este modelo, cada réplica recebe “I’m Alive” da réplica que se encontra imediatamente acima e envia “I’m Alive” para a réplica que se encontra abaixo, com excepção do Primário, que apenas envia, e do último secundário, que apenas recebe.

O filtro destas mensagens (do tipo “GetNextAdress” ou do tipo “I’m Alive”) é feito utilizando SOAP Handlers. O handler identifica qual o tipo de mensagem a entrar e especifica o comportamento correcto consoante a mensagem. Para mensagens do tipo “GetNextAdress” deve retornar o seu nextEndpoint à repica que enviou a mensagem; para mensagens do tipo “I’m Alive” deve interromper a thread receptora de forma a actualizar a janela de timeout. Estas mensagens são envelopes vazios com um atributo no header que as identifica correctamente.

Visto que o protocolo é implementado exclusivamente com SOAP Handlers, o processo de replicação e recuperação de faltas é totalmente transparente para o cliente que executa o código. Este não consegue distinguir as máquinas com quem comunica e o *delay* da recuperação é baixo (de, aproximadamente, entre 1 e 2 segundos - assumindo um tráfego normal na rede). Não obstante, é independente do domínio em que é utilizado: é possível utilizar o protocolo com qualquer serviço que comunique com um Webservice sem que este tenha de alterar a sua implementação.

Tolerância a Faltas

O protocolo define uma janela máxima de *timeout* de recepção de “I’m Alive” de 2000 ms, após a qual o secundário que os recebe assume que a réplica que aponta para si (que lhe envia “I’m alive”) falhou silenciosamente e trata de reiniciar o processo de inserção, de forma a actualizar a lista. Desta forma, é necessário actualizar apenas o “elo” que se partiu, sem que nenhuma das outras réplicas seja afectada ou faça qualquer tipo de processamento. O intervalo de envio de “I’m alive” é de 500 ms.

Os valores definidos para envio e para timeout são restritivos, assumindo uma aplicação real do protocolo.

Este mecanismo permite que as falhas sejam rapidamente descobertas e que se recupere da situação de erro com um impacto mínimo na performance do serviço para o cliente (de acordo com o modelo de interacção dado).

¹ <http://imgur.com/PDPpYik> (Ilustração simples do processo de interacção entre réplicas ao iniciar)

Caso o primário falhe, a réplica abaixo de si (e apenas esta) deixa de receber “I’m Alive” e trata de se reinserir na lista. Ao contactar o primário registado no UDDI, apercebe-se de que este falhou e, como tal, regista-se como primário no UDDI (rebind). Caso o cliente contacte o primário e este ainda não tenha sido substituído, os pedidos são reenviados (dentro de um período de tempo) até que uma nova réplica (a que se encontra na 2ª posição) se registre como primário. Desta forma, são toleradas potenciais faltas silenciosas do servidor primário.

Caso aconteçam múltiplas faltas silenciosas em simultâneo, o sistema consegue sempre recuperar pois cada réplica insere-se no fim da lista, o que permite que a comunicação entre as réplicas seja sempre restabelecida. Caso não acontecesse, prejudicaria a replicação de mensagens e, por conseguinte, a sua consistência.

Este mecanismo foi testado lançando múltiplas réplicas (2, 3, 4 e 5) e em cada caso, verificando a recuperação de todas quando uma ou várias réplicas, incluindo o primário, falham silenciosamente. Foi verificado o processo de recuperação (troca de mensagens entre as réplicas da lista) e a confirmação de que todas as réplicas se inserem como devem e onde devem e que o lançamento e o término de threads de envio e recepção de I’m alive é iniciado e terminado como descrito acima. De referir que os testes foram minuciosos e exaustivos quanto ao processo de recuperação/replicação.

Replicação

Quando o primário recebe um pedido do cliente, e antes de este ser executado e a resposta retornada, o pedido é propagado pela lista de réplicas (através do Handler), sendo o pedido executado em todas as réplicas, e sendo, assim, garantida a consistência sequencial em cada uma delas, já que estas são executadas sempre com a mesma ordem com que foram comunicadas, pelo cliente, ao primário.

Para conseguir garantir a consistência sequencial na totalidade, o protocolo aplica também uma semântica de pedidos apenas-uma-vez.

A semântica apenas-uma-vez é implementada introduzindo um número de sequência nas mensagens que são enviadas pelo cliente e fazendo com que cada réplica descarte mensagens com um número de sequência inferior ao da última mensagem recebida, e respondendo com a mensagem enviada anteriormente. Desta forma, não há duplicação de mensagens, garantindo que uma mensagem é executada uma só vez. Assim, caso o cliente comunique uma mensagem ao primário e este morra *após* ter replicado a mensagem e *antes* de ter retornado a resposta, o pedido do cliente é novamente enviado e o secundário que substitui o primário detecta o pedido repetido e retorna a última mensagem de resposta saída, garantindo a resposta correcta ao cliente.

Este mecanismo foi testado lançando várias réplicas e forçando o primário a falhar silenciosamente, *após* ter replicado a mensagem do cliente e antes de ter retornado, por forma a garantir a situação descrita. O protocolo executa o comportamento descrito: o servidor secundário assume o lugar do primário e, ao receber mensagens com o id das mensagens já passadas ao servidor primário, retorna o valor da respectiva mensagem de resposta, sem executar o código desse pedido. Confirmámos que o cliente recebe a resposta correcta aos pedidos a que o primário não deu resposta.

Mensagens recebidas por uma réplica que falhou silenciosamente não são replicadas, visto que isso obrigaria a reiniciar a réplica que falhou e a actualizar o seu estado e a colocá-la consistente com as outras réplicas, o que forçaria uma implementação com um nível de complexidade bastante elevado. Este é o único caso em que o protocolo não garante uma consistência correcta ou uma devolução de respostas certas ao cliente.