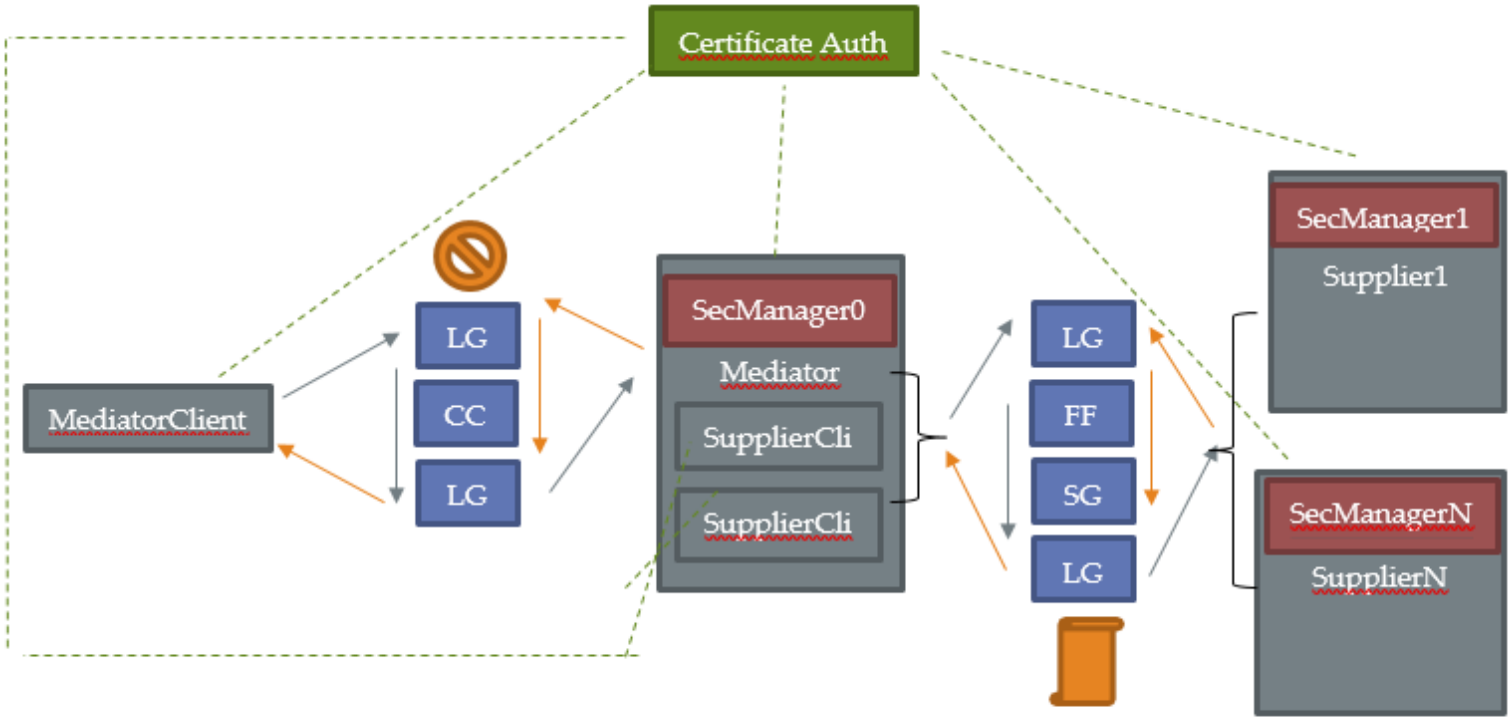
TÉCNICO LISBOA

Projeto de Sistemas Distribuídos

https://github.com/tecico-distsys/T64-Komparator

Grupo T64 – Aluno: Francisco Teixeira de Barros, nº: 85069 – LETI,
Segundo Semestre - Ano Letivo de 2016-2017

Diagrama de Solução



Legenda e explicação da solução[1]

Certifcate Authority (CA): É um modulo que detém as chaves públicas das restantes entidades.. Sempre que uma entidade servidora ou cliente precisa de obter uma chave pública de outra entidade, será através da **CA** que a deverá obter. O único módulo que depende da CA é no entanto o *security*.

Security (SEC): Este modulo não está representado na figura. Detém todas as classes *Handler*, representadas a azul. Para além disso contém ainda as classes: *CryptoUtil* que define diversos métodos relacionados com a conversão de dados, por exemplo de *byte[]* para *String*, assim como os métodos de cifra e decifra, obtenção de chaves a partir da **CA**, entre outros. A grande maioria são estáticos, o que permite a factorização de código. Finalmente contem ainda uma classe que implementa o padrão de desenho *Singleton*, denominada *SecurityManager* (**SM**), representados na figura a vermelho. Cada interveniente, pode ter a sua própria instância da classe **SM**, permitindo, que através desta sejam obtidas todas as informações necessárias e desejadas do modulo **CA**. A classe **SM**, providência ainda os métodos que permitem aos *handlers*, tratar da frescura das mensagens, evitar ataques por repetição e afins. A frescura é garantida rejeitando pedidos cujo o tempo, desde a partida no remessor até à chegada ao *servlet*, seja superior a três segundos. Os ataques por repetição são prevenidos mantendo um *vector* que mantem o ID dos pedidos que chegaram ao servidor e que foram respondidos. Esse ID é uma *String* obtida a partir da classe do Java, *SafeNumber* que gera números aleatórios seguros com *32bits* com o algoritmo "SHA1PRNG" que depois é convertido em texto. A confidencialidade do campo do cartão de crédito na operação *buyCart* que ocorre entre os *MediatorClient* $\mathbf{MC}_i$ e o *Mediator* **M** é assegurada através de cifra assimétrica com algoritmo "RSA/ECB/PKCS1Padding". A cifra assimétrica simples é ainda usada para a geração de assinaturas, que garantem a autenticidade das mensagens trocadas em ambos os sentidos entre os *SupplierClient* $\mathbf{SC}_i$ e os *Supplier* $\mathbf{S}_i$. Outra funcionalidade importante da classe *singleton SecurityManager*, é a seguinte: quando o **M** ou um qualquer $\mathbf{S}_i$ é instanciado este gera logo a sua instância de **SM** passando-lhe a informação acerca do seu nome de serviço. O que virá mais tarde a ser útil na troca de mensagens, mas não em todo o código.*

Os elementos da figura representados a cinza, são as já conhecidas entidades das entregas anteriores: *mediator-ws-cli*, *mediator-ws*, *supplier-ws-cli* e *supplier-ws*. O modulo *mediator-ws* depende do modulo *supplier-ws-cli*, dai as *N* instancias de *SupplierClient* representadas na figura nele contidas.

Finalmente os elementos azuis da figura representam cadeia de *Handlers* que permitem a gestão das comunicações entre entidades. Na realidade os elementos da figura ***CC***, ***FF*** *e* ***SG*** representam cada um, dois *Handlers*. As setas acinzentadas e alaranjadas representam a passagem de mensagens entre entidades e as setas dessas mesmas cores verticais representam a interceção de mensagens a saída ou chegada de um módulo e ainda a ordem pela qual os *Handlers* intervêm na mensagem.

*Note-se que a potencialidade do *SecurityManager* não foi aproveitada ao máximo. A classe embora funcional e encontra-se bastante ambígua por falta de tempo. A abstração da utilização da classe *CryptoUtil* por parte dos *Handler* também ficou incompleta pois a sua utilização devia ser feita só e apenas através dos métodos disponibilizados em *SecurityManager*.

Handlers

Os *Handler* **LG**, são no projecto os *LoggingHandler*, tem exatamente as mesmas funções em qualquer etapa da da comunicação entre qualquer entidade e apenas imprimem informação dos

cabeçalhos de cada mensagem no *standart output*. Não iremos considerar esta em nenhuma das explicações que se seguem. Assuma ainda que, sempre que não for dito o que é que um *Handler* **H**$_i$ faz sobre uma mensagem num determinado sentido, é porque não faz nada. Tememos como exemplo a operação buyCart, desde o princípio ao fim da cadeia passando por todos os intervenientes.

A primeira cadeia de *Handlerss* H1, representada na figura com o sinal de proibido laranja. É portanto constituída por uma etapa, **CC**. Esta etapa contem dois Handlers, um **CCMediatorClientHandler**, que só realiza trabalho para o MediatorClient **MC** e apenas em mensagens *outbound*, verificando se a mensagem corresponde a uma operação "*buyCart*" do contexto da mensagem WSDL_OPERATION. Se tal for verdade, então, obtém o SOAPBODY e este é percorrido até ser encontrada o *Node* cujo o nome é "*creditCardNr*". É feita uma operação de cifra através do método *CryptoUtil#asymCipher*, usando a chave pública do *Mediator* **M**, obtida através do **CA** usando o *singleton* **SM** que também verifica se realmente foi o **CA** que providenciou a chave. A obtenção da chave pública correta é feita através do nome do **M** que é passado por contexto, via *Handler Relays* aquando da instanciação de um MC. O mesmo é feito mais tarde nas instanciações de *SupplierClients* **SC**. Finalmente texto é salvaguardo e a mensagem é libertada. Chegando ao chegar ao **M** a mensagem *inbound* é intercetada pelo **CCMediatorHandler**, que ao verificar caso se trate de uma operação *buyCart*, vai percorrer toda ao *SOAPHEADER* buscar o seu nome, que usar para obter a sua chave privada como recurso usando os métodos de *CryptUtil*. Procura no *SOAPBODY* pela node *creditCardNr* e para que o possa decifrar. Terminada esta operação, o **M** instancia os devidos **SCs** e damos assim início à segunda cadeia de *Handlers*, representada na figura por um papiro laranja.

O **SC** instanciado, passa via *relays*, o nome do *Supplier* S, destino e o seu nome. Ao enviar a sua mensagem *outbound* esta é intercetada primeiro pelo **FreshnessSuplierClientHandler**, que adiciona ao cabeçalho da mensagem um *timestamp* e um *requestId*, como explicado em (1) e de seguida por **SignatureSuplierClientHandler**, que usa a chave privada do **M** como se fosse a sua, para assinar a mensagem. Isto é feito escrevendo a *SOAPMessage* para um *ByteArrayOutputStream*, do qual se obtém o *byte[]* e a respetiva representação *String*, que é digerida pelo método *CryptoUtil#makeDigitalSignature*. Sendo depois esta adicionada ao cabeçalho, juntamente com o nome da entidade remessora e receptora. Chegando ao fim desta cadeia a mensagem passa pela rede e ao chegar ao respetivo **S** é intercetada por **FreshnessSuplierHandler**, que verifica se deve ou não responder à mensagem, podendo rejeitá-la justificadamente. Caso decida positivamente, o **SignatureSuplierHandler**, obtém a chave pública do **M** via **SM**, guarda e apaga do *SOAPHEADER* a assinatura do **SC**. De seguida verifica a assinatura fazendo digest do resto da *SOAPMESSAGE* com a chave pública obtida. Se tudo tiver bem, o **S** executa as devidas funções.

Começa agora o caminho de retorno com mensagens de resposta. Neste regresso os *Handlers* **FF** e **CC** não fazem nada e os *Handlers* **Signature** trocam de papel, isto é, no sentido de interção inverso, a mensagem de resposta é assinada pelo **S** através do **SignatureSuplierHandler** através do seguindo a mesma lógica, já apresentada anteriormente e esta assinatura é verificada em **SignatureSuplierClientHandler** antes da mensagem ser entregue ao **SC**, permitindo que o regresso à cadeia **H1** até que chega a resposta ao **MC**. A diferença neste caminho inverso é que são usadas, respetivamente, para assinar e verificar, as chave privadas e pública do **SupplierN**, só que desta vez, a informação referente ao nome do **S** passado para os cabeçalhos é feita, não via *relays*, mas através do campo descrito em (1) presente em no *singleton* **SM**, preenchido aquando da sua instanciação, aquando da instanciação do próprio **S**. Esta funcionalidade podia ter sido aproveitada em outras cadeias, mas não foi por falta de planeamento.