

WS-Security

Is it really that simple?

Eelco Klaver (eklaver@econsulting.nl)

LinkedIn: <http://www.linkedin.com/in/eklaver>

Member of knowledge network for freelance software engineers (FSIKN)

Goal of this presentation

- Show that although the WS-Security specifications look very complicated, the current WS-Security frameworks make it relatively easy to implement real end-to-end message-level security for web services
- Scope
 - Focus on software solution with open-source frameworks
 - XML hardware appliances are out of scope

Contents

- Introduction
 - Why Web Services Security?
 - Message-level vs. transport-level security
- WS-Security in theory: the hard stuff
 - XML Signature and JSR-105
 - XML Encryption and JSR-106
 - OASIS Web Services Security (WS-Security)
- WS-Security in practice: frameworks really help
 - XWS-Security and Spring-WS
 - WSS4J and Axis
 - Comparison
- Conclusion
 - Best practices and lessons learned

Introduction

DOING SECURITY RIGHT FROM THE START!

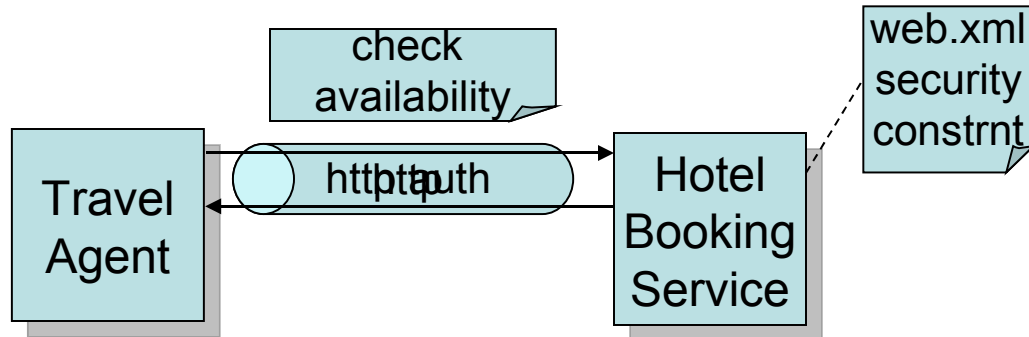
Why Web Services Security?

- History of Web Services Security
 - As with a lot of technologies, in the beginning there was nothing...
 - Submitted OASIS TC in September 2002
 - It became 1.0 specification in April 2004
- Security goals
 - Authentication
 - Authorization
 - Confidentiality
 - Integrity
 - Non-repudiation



Transport-level security

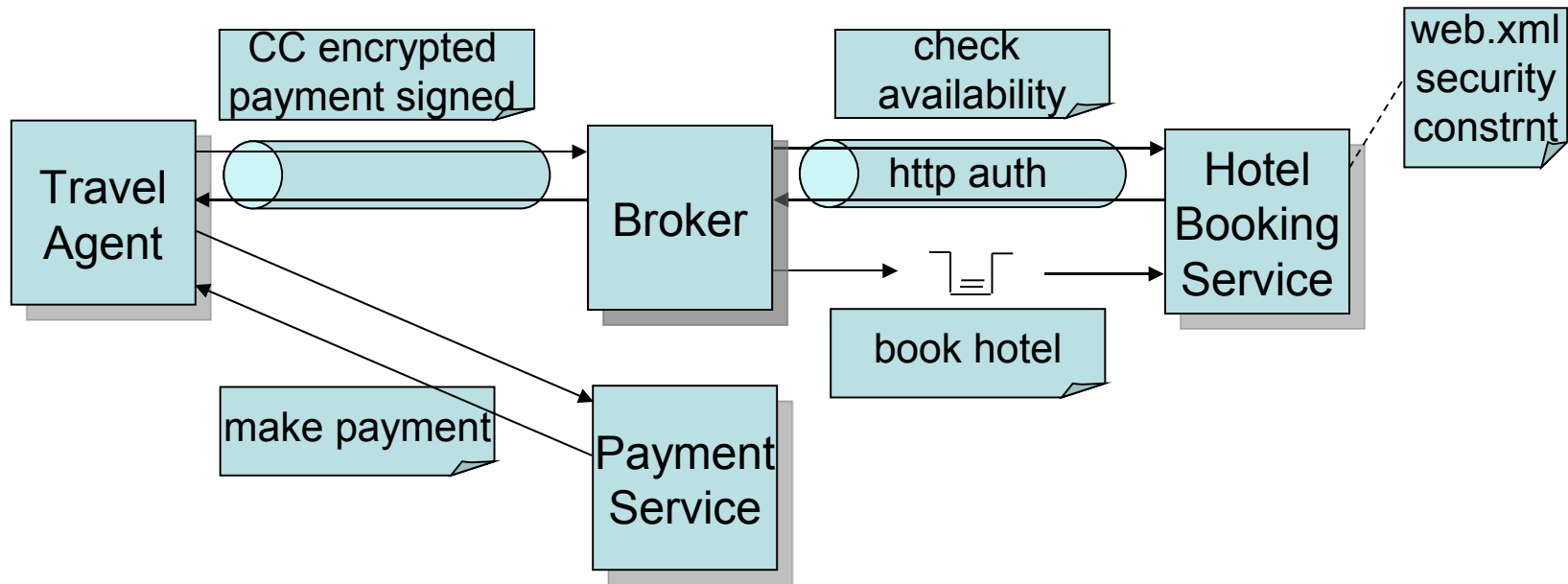
- Securing web services at the transport level



- Authentication: basic, digest or client-certificate authentication
- Authorization: role based access control on URL using standard J2EE security
- Confidentiality: enforce SSL
- Integrity: enforce SSL
- Non repudiation: Not supported

Message vs. transport level security

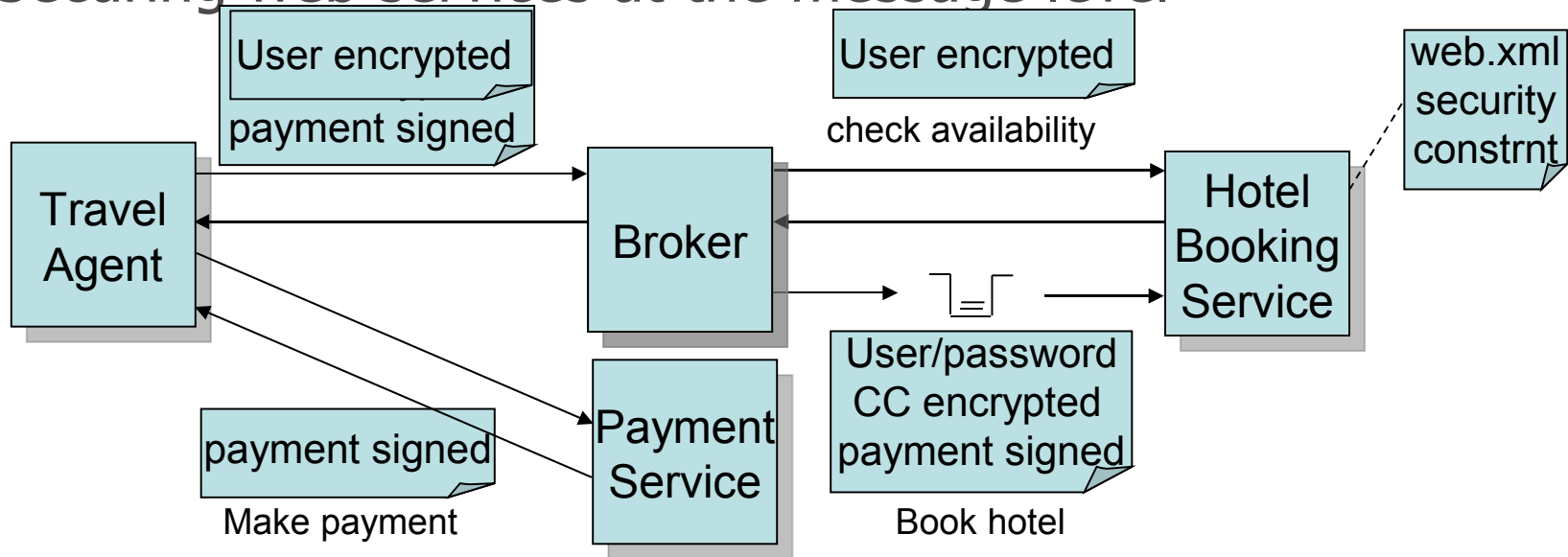
- Message-level security vs. transport-level security



- Different transport protocols
- End-to-end security when message travels intermediate nodes
- Only parts of message signed and/or encrypted

Message-level security

- Securing web services at the message level



- Authentication: SAML, Username Token, XML Signature
- Authorization: J2EE security, SAML Assertions
- Confidentiality: XML Encryption
- Integrity: XML Signature
- Non-repudiation: XML Signature, Logging

WS-Security in theory: the hard stuff

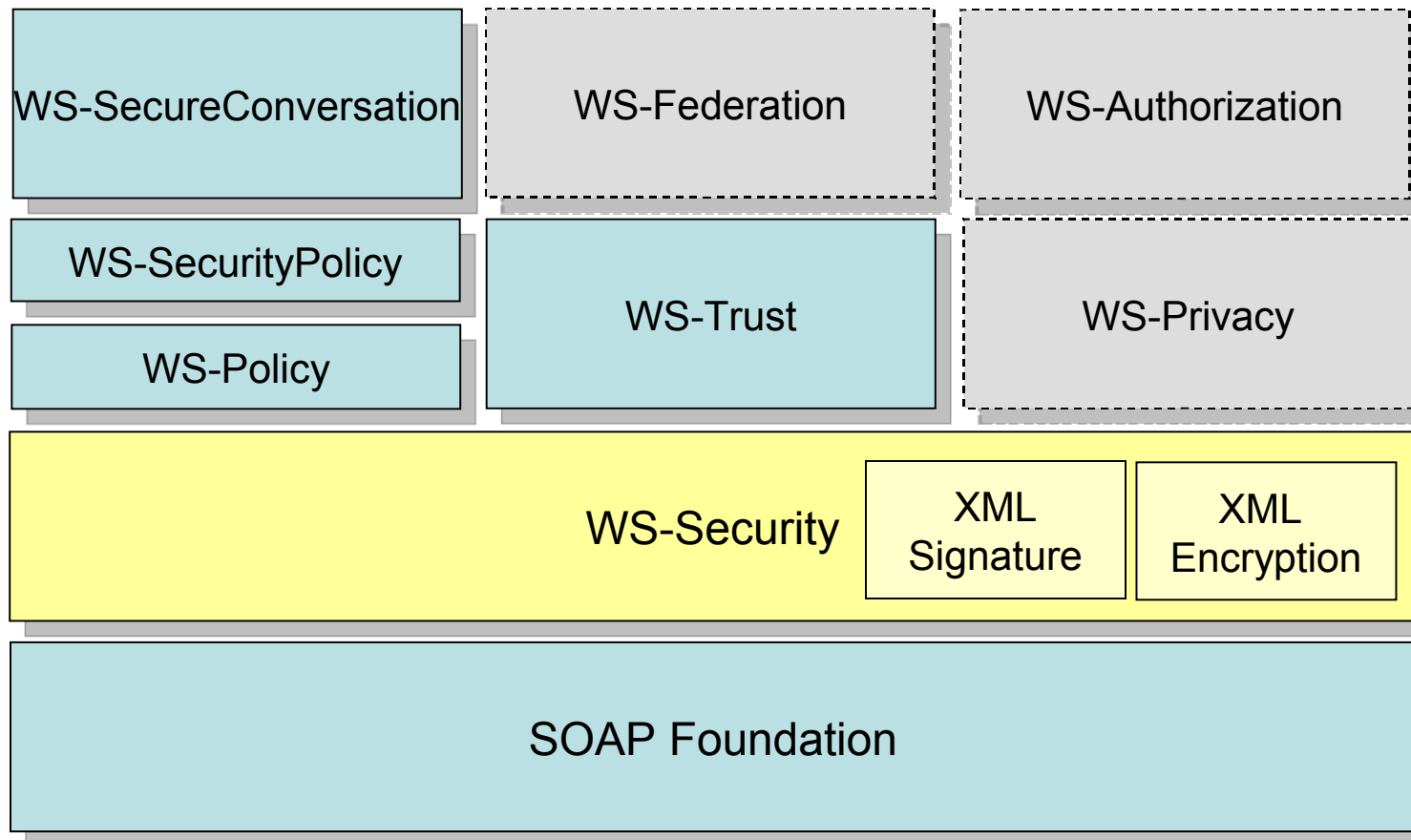


© Scott Adams, Inc./Dist. by UFS, Inc.

DOING SECURITY RIGHT FROM THE START!

WS-Security Stack

- Part of a roadmap



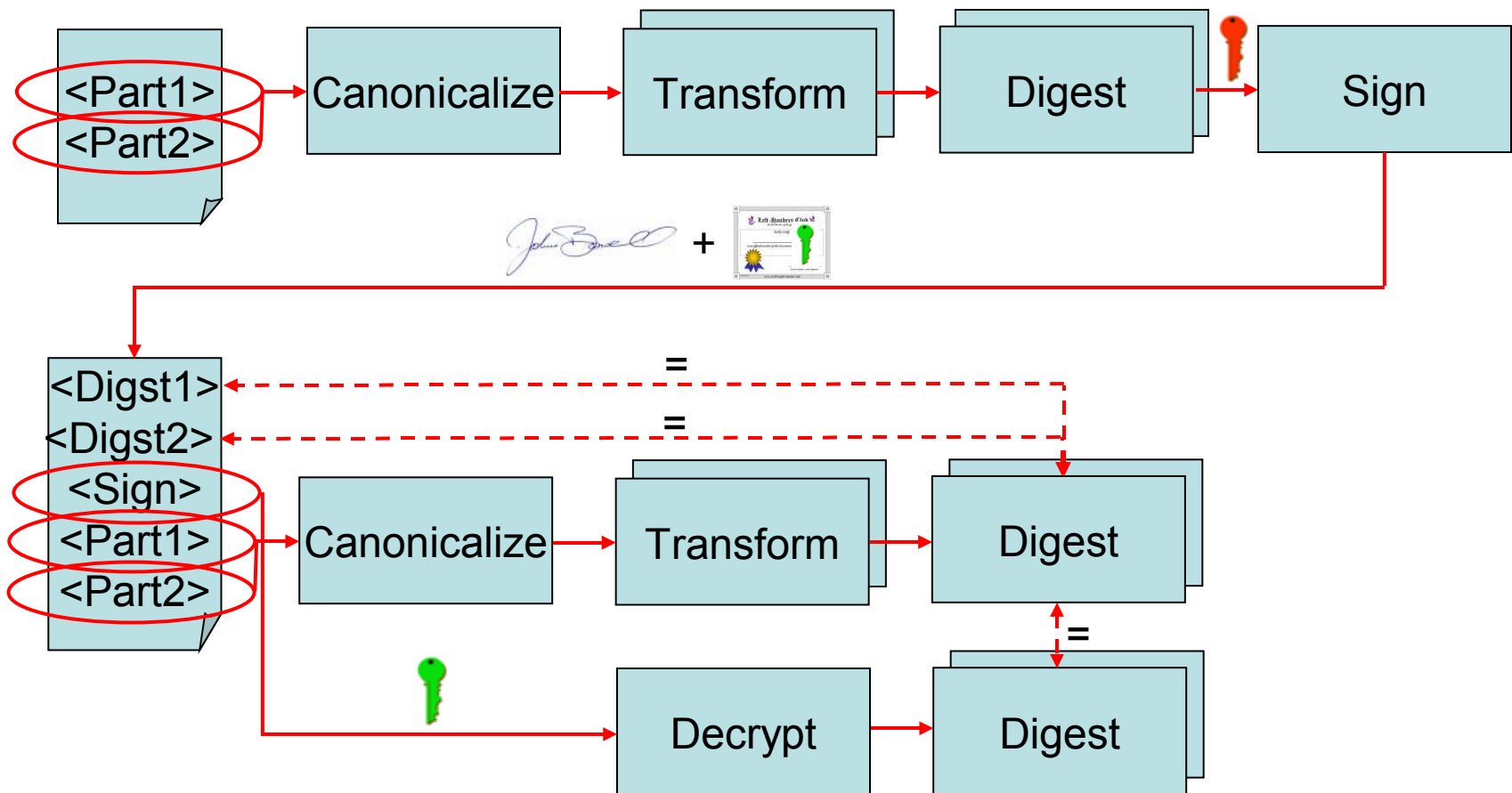
DOING SECURITY RIGHT FROM THE START!

XML Signature and JSR-105

- W3C Recommendation of 12 Februari 2002
- Goals
 - Ensure data integrity
 - Message authentication
 - Non-repudiation
- Motivation
 - Signing part of message
 - Involving multiple parties
- Signature representations
 - Enveloped signatures
 - Enveloping signatures
 - Detached signatures

XML Signature and JSR-105

- Processing



DOING SECURITY RIGHT FROM THE START!

XML Signature and JSR-105

- Syntax

```
<Signature ID?>
  <SignedInfo>
    <CanonicalizationMethod/>, e.g. CanonicalXML
    <SignatureMethod/>, e.g. DSA-SHA1 and PKCS1
    (<Reference URI?>
      (<Transforms/>)?, e.g. XSLT and XPath
      <DigestMethod/>, e.g. SHA1
      <DigestValue/>
    </Reference>)+
  </SignedInfo>
  <SignatureValue/>
  (<KeyInfo/>)?
  (<Object ID?>)*
</Signature>
```

XML Signature and JSR-105

- Example: Signed XML document using enveloped signature

```
<?xml version="1.0" encoding="UTF-8"?>
  <Envelope xmlns="urn:envelope">
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
        <CanonicalizationMethod Algorithm="http://.../REC-xml-c14n-
20010315#WithComments"/>
        <SignatureMethod Algorithm="http://.../xmldsig#dsa-sha1"/>
          <Reference URI="">
            <Transforms>
              <Transform Algorithm="http://.../xmldsig#enveloped-
signature"/>
            </Transforms>
            <DigestMethod Algorithm="http://.../xmldsig#sha1"/>
              <DigestValue>uooqbWYa5VCqcJCbuymBKqm17vY=</DigestValue>
          </Reference>
        </SignedInfo>
        <SignatureValue>KedJuTob5g...
BwVbEQRl26S2tmXjqNND7MRGtoew==</SignatureValue>
      <KeyInfo>
        <KeyValue><DSAKeyValue>...</DSAKeyValue></KeyValue>
      </KeyInfo>
    </Signature>
  </Envelope>
```

XML Signature and JSR-105

- JSR 105: XML Digital Signature APIs
 - Implementation of W3C standard based on Apache's XML-DSig.
- Example

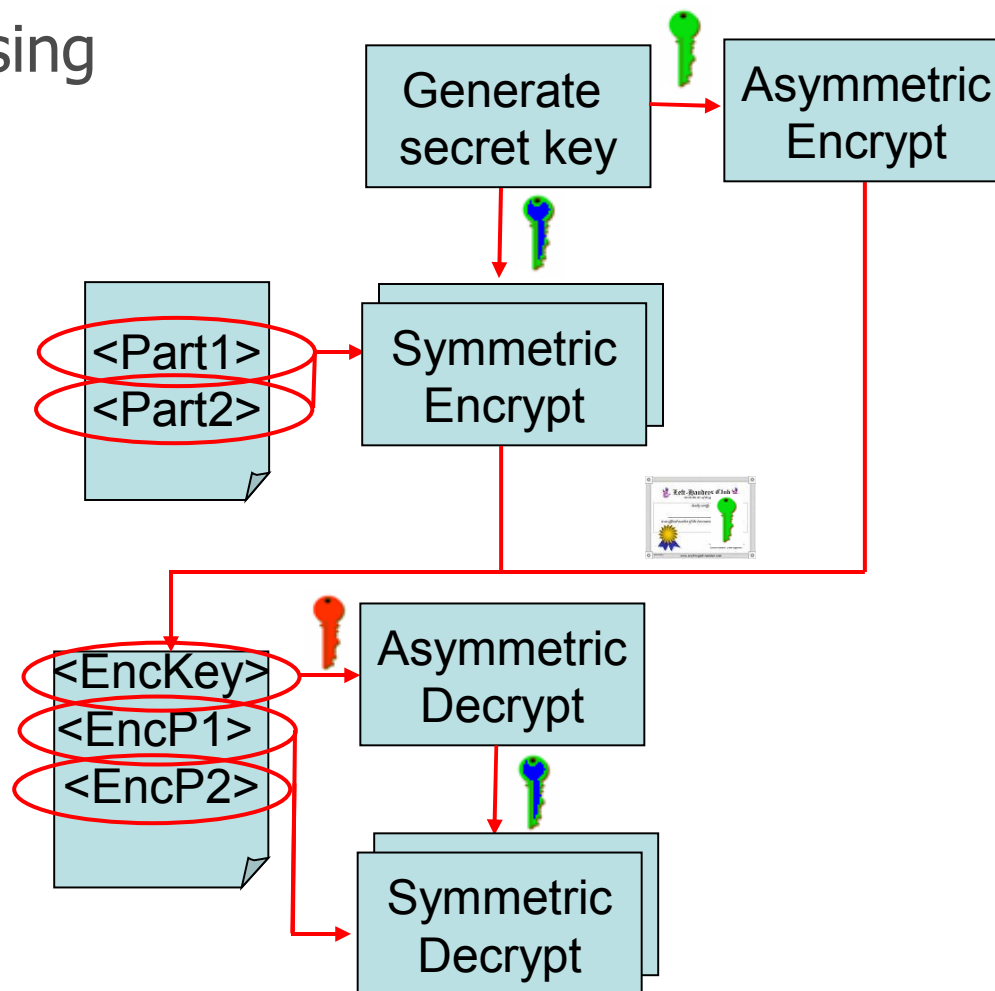
```
XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM");
Reference ref = fac.newReference("",
    fac.newDigestMethod(DigestMethod.SHA1, null),
    Collections.singletonList(
        fac.newTransform(Transform.ENVELOPED, null)),
    null, null);
SignedInfo si = fac.newSignedInfo(
    fac.newCanonicalizationMethod(
        CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS, null),
    fac.newSignatureMethod(SignatureMethod.DSA_SHA1, null),
    Collections.singletonList(ref));
KeyInfoFactory kif = fac.getKeyInfoFactory();
KeyValue kv = kif.newKeyValue(kp.getPublic());
KeyInfo ki = kif.newKeyInfo(Collections.singletonList(kv));
XMLSignature signature = fac.newXMLSignature(si, ki);
signature.sign(dsc);
```

XML Encryption and JSR-106

- W3C Recommendation of 10 December 2002
- Goals
 - Data confidentiality
 - Ensure end-to-end security of messages
- Motivation
 - Apply encryption to portions of the message
 - Multiple encryptions to different parts for multiple parties
- Types of encryption
 - Entire document encryption
 - Single element encryption
 - Element content encryption

XML Encryption and JSR-106

- Processing



DOING SECURITY RIGHT FROM THE START!

XML Encryption and JSR-106

- Syntax

```
<EncryptedData Id? Type? MimeType? Encoding?>
  <EncryptionMethod/>?, e.g. TRIPLEDES, AES-128, RSA
  <ds:KeyInfo>
    <EncryptedKey>?
    <AgreementMethod/>?, e.g. Diffie-Hellman
    <ds:KeyName>?
    <ds:RetrievalMethod/>?
    <ds:*>?
  </ds:KeyInfo>?
  <CipherData>
    <CipherValue>?
    <CipherReference URI?>?
  </CipherData>
  <EncryptionProperties>?
</EncryptedData>
```

XML Encryption and JSR-106

- Example: Single element encryption

```
<PurchaseOrder>
  <Order>
    <Id>123-958-74598</Id>
    <Quantity>12</Quantity>
  </Order>
  <Payment>
    <CardId>123654-8988889-9996874</CardId>
    <CardName>visa</CardName>
    <ValidDate>12-10-2004</ValidDate>
  </Payment>
</PurchaseOrder>
```

```
<PurchaseOrder>
  <Order>
    <Id>123-958-74598</Id>
    <Quantity>12</Quantity>
  </Order>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
    xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <CipherData><CipherValue>A23B45C564587</CipherValue></CipherData>
  </EncryptedData>
</PurchaseOrder>
```

XML Encryption and JSR-106

- JSR-106: XML Digital Encryption APIs

```
// Create DOM factory
```

```
XMLEncryptionFactory fac = XMLEncryptionFactory.getInstance();
```

```
// Create decryption context
```

```
KeySelector ks = null; //XXX - code missing to create KeySelector
```

```
Element elem = null; //XXX - code missing to identify element to decrypt
```

```
XMLDecryptContext xdc = new DOMDecryptContext(ks, elem);
```

```
// Unmarshal EncryptedData
```

```
EncryptedData ed = (EncryptedData) fac.unmarshalEncryptedType(xdc);
```

```
// Decrypt EncryptedData
```

```
InputStream decData = ed.decrypt(xdc);
```

WS-Security - Introduction

- OASIS SOAP Message Security 1.1 Standard, Feb. 2006
- Goals
 - End-to-end message-level security for SOAP messages
 - Integrate multiple security standards, i.e. multiple security token formats, signature formats, encryption technologies, trust domains
- Motivation
 - Standard set of SOAP extensions for message authentication, message integrity, message confidentiality, security token propagation.
- Integrated technologies
 - XML Signature
 - XML Encryption
 - SAML and REL

WS-Security - Overview

- Security headers
 - One header for each actor
 - Security tokens in separate profiles
 - Username/password token
 - X509 Token
 - Kerberos Token
 - SAML Token
 - Rights Expression Language (REL) Token
 - Timestamp
 - Signature
- Encryption of elements

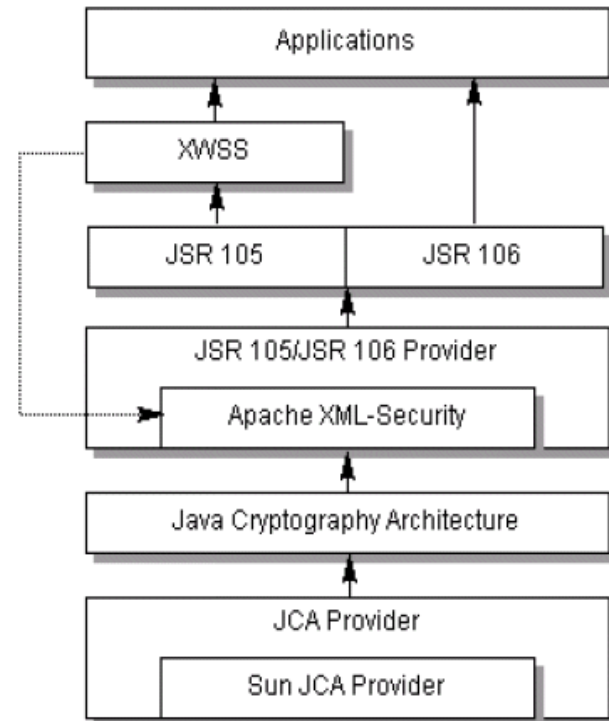
- **Syntax**

```
<SOAP:Envelope>
  <SOAP:Header>
    <wsse:Security actor="" mustUnderstand="">
      <wsse:BinarySecurityToken>?
      <ds:Signature>
        <ds:SingedInfo>
          (<ds:Reference>)+
        </ds:SingedInfo>
        <ds:SignatureValue>
        <ds:KeyInfo>?
      </ds:Signature>
      <xenc:EncryptedKey>
        <ds:KeyInfo>?
      </xenc:EncryptedKey>?
      <wsu:TimeStamp>?
    </wsse:Security>
  </SOAP:Header>
  <SOAP:Body>
    <xenc:EncryptedData>?
  </SOAP:Body>
</SOAP:Envelope>
```

WS-Security in practice: frameworks really help

WS-Security toolkits

- XWS-Security
 - Part of SunJWSDP
 - XML Digital Signature
 - Apache XML-ENC
 - Integrates with Spring-WS
- WSS4J
 - Apache WS-Security
 - XML Digital Signature
 - XML Encryption
 - Integrates with Axis and XFire



XWS-Security - Overview

- An implementation of the OASIS Web Services Security
 - Packaged with SunJWSDP
 - Uses JSR-105, SAAJ 1.3, JAX RPC
- Implements
 - OASIS Web Services Security: SOAP Message Security 1.1 Standard, Februari 2006
 - OASIS WSS UsernameToken profile 1.1 (partial)
 - OASIS X.509 Token Profile 1.1 (partial)
 - OASIS SAML Token Profile 1.1 (partial)
- Reference
 - <https://xwss.dev.java.net/>

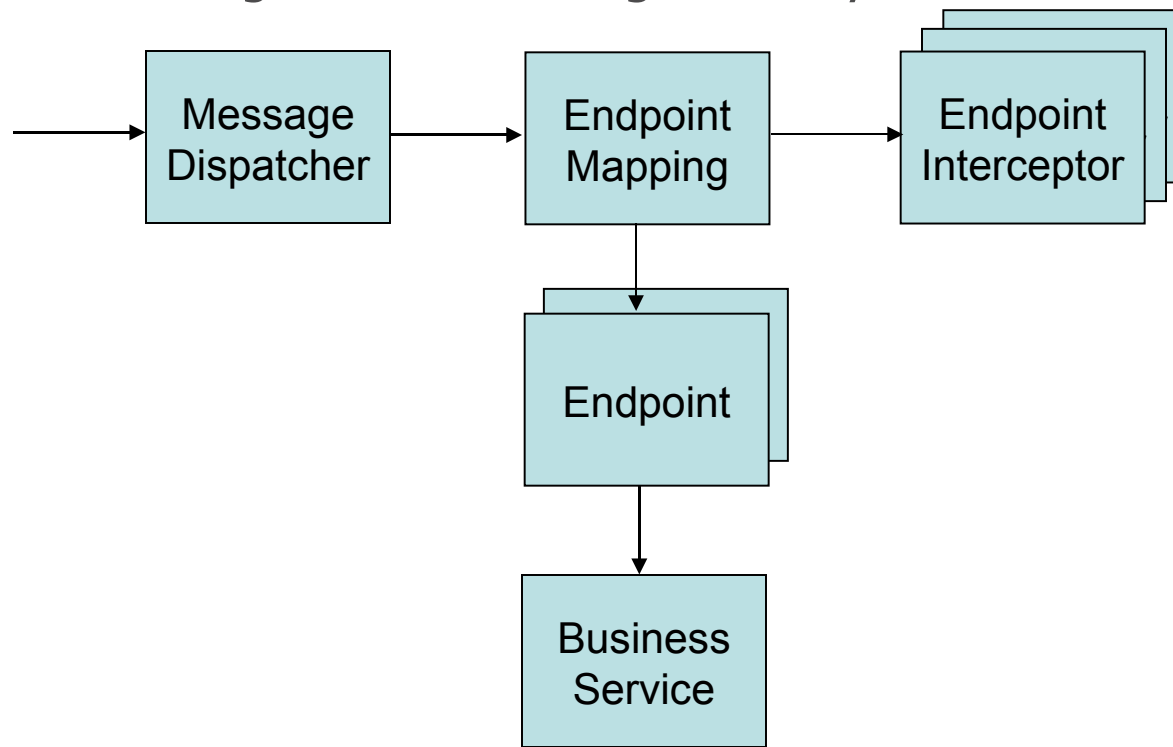
XWS-Security - API

- Main classes



Spring-WS – Overview

- WS-Security implemented by interceptors
 - Based on XWS-Security 2.0 and XML-Security
 - Integration with Acegi Security Framework



DOING SECURITY RIGHT FROM THE START!

Spring-WS – UsernameToken example

- **Server security configuration (server_security_config.xml)**

```
<xwss:SecurityConfiguration
  xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
  <xwss:RequireUsernameToken passwordDigestRequired="false"
    id="username"/>
</xwss:SecurityConfiguration>
```

- **Server Spring configuration**

```
<bean id="wsSecurityInterceptor" class="...XwsSecurityInterceptor">
  <property name="policyConfiguration"
    value="classpath:server_security_config.xml"/>
  <property name="callbackHandlers">
    <list><ref bean="passwordValidationHandler"/></list>
  </property>
</bean>
<bean id="passwordValidationHandler" class="...
  SimplePasswordValidationCallbackHandler">
  <property name="users">
    <props><prop key="Eelco">Passw@rd</prop></props>
  </property>
</bean>
```

Spring-WS – UsernameToken example

- Client security configuration (client_security_config.xml)

```
<xwss:SecurityConfiguration
    xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
    <xwss:UsernameToken digestPassword="false" id="username"/>
</xwss:SecurityConfiguration>
```

- Client Spring configuration

```
<bean id="securityCallback" class="ClntSecurityMessageCallback">
    <property name="interceptor" ref="wsSecurityInterceptor"/>
</bean>
<bean id="wsSecurityInterceptor"
    class="XwsClientSecurityInterceptor">
    <property name="policyConfiguration"
        value="classpath:client_security_config.xml"/>
    <property name="callbackHandlers">
        <list><ref bean="usernamePasswordHandler"/></list>
    </property>
</bean>
```

DOING SECURITY RIGHT FROM THE START!

Spring-WS – UsernameToken example

- Request message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="...">
<SOAP-ENV:Header>
  <wsse:Security xmlns:wsse="..." SOAP-ENV:mustUnderstand="1">
    <wsse:UsernameToken xmlns:wsu="..." wsu:Id="username">
      <wsse:Username>Eelco</wsse:Username>
      <wsse:Password Type="...#PasswordText">Passw@rd</wsse:Password>
      <wsse:Nonce EncodingType="...">F1I0woRT5UT1j8imsSK2znXU</wsse:Nonce>
      <wsu:Created>2007-10-09T19:49:55.750Z</wsu:Created>
    </wsse:UsernameToken>
  </wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
...
<SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Spring-WS – UsernameToken example

- Replay attack on message

```
<SOAP-ENV:Envelope xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Client</faultcode>
      <faultstring xml:lang="en">
com.sun.xml.wss.impl.WssSoapFaultException: Invalid/Repeated
Nonce value for Username Token; nested exception is
com.sun.xml.wss.XWSecurityException:
com.sun.xml.wss.impl.WssSoapFaultException: Invalid/Repeated
Nonce value for Username Token</faultstring>
      </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Spring-WS – Signature example

- Server security configuration (server_security_config.xml)

```
<xwss:SecurityConfiguration
  xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
  <xwss:RequireSignature/>
</xwss:SecurityConfiguration>
```

- Server Spring configuration

```
<bean id="wsSecurityInterceptor" class="...XwsSecurityInterceptor">
  <property name="policyConfiguration"
            value="classpath:server_security_config.xml"/>
  <property name="callbackHandlers">
    <list><ref bean="keyStoreHandler"/></list>
  </property>
</bean>
<bean id="keyStoreHandler" class="...KeyStoreCallbackHandler">
  <property name="trustStore" ref="trustStore"/>
</bean>
```

Spring-WS – Signature example

- **Client security configuration (client_security_config.xml)**

```
<xwss:SecurityConfiguration
    xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
    <xwss:Sign>
</xwss:SecurityConfiguration>
```

- **Client Spring configuration**

```
<bean id="wsSecurityInterceptor"
        class="XwsClientSecurityInterceptor">
    <property name="policyConfiguration"
        value="classpath:client_security_config.xml"/>
    <property name="callbackHandlers">
        <list><ref bean="keyStoreHandler"/></list>
    </property>
</bean>
<bean id="keyStoreHandler" class="..KeyStoreCallbackHandler">
    <property name="keyStore" ref="keyStore"/>
    <property name="privateKeyPassword" value="welkom"/>
    <property name="defaultAlias" value="client cert"/>
</bean>
```

Spring-WS – Signature example

- Request message

```
<wsse:Security xmlns:wsse="..." SOAP-ENV:mustUnderstand="1">
  <wsse:BinarySecurityToken wsu:Id="123">...cR+i2qxh3Th5g==</wsse:BinarySecurityToken>
  <ds:Signature xmlns:ds="...">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
      <ds:Reference URI="#XWSSGID-2">
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <ds:DigestValue>LRUBFOQuswY8SPN6cxAW0FWGsBo=</ds:DigestValue>
      </ds:Reference>
      <ds:Reference URI="#XWSSGID-1">
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <ds:DigestValue>QKLcAro6/E550+BrVYud8q9vbBs=</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>...Ly/K8he11WVME1CXVJE=</ds:SignatureValue>
    <ds:KeyInfo>
      <wsse:SecurityTokenReference xmlns:wsu="..." wsu:Id="XWSSGID-1191962859250104407195">
        <wsse:Reference URI="#XWSSGID-11919628588432044887650" ValueType="...#X509v3" />
      </wsse:SecurityTokenReference>
    </ds:KeyInfo>
  </ds:Signature>
  <wsu:Timestamp xmlns:wsu="..." wsu:Id="XWSSGID-1">
    <wsu:Created>2007-10-09T20:47:39.171Z</wsu:Created>
    <wsu:Expires>2007-10-09T20:47:44.171Z</wsu:Expires>
  </wsu:Timestamp>
  DOING SECURITY RIGHT FROM THE START!
</wsse:Security>
<SOAP-ENV:Body wsu:Id="XWSSGID-2"></SOAP-ENV:Body>
```

Spring-WS – Encryption example

- **Server security configuration (server_security_config.xml)**

```
<xwss:SecurityConfiguration
  xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
  <xwss:RequireEncryption/>
</xwss:SecurityConfiguration>
```

- **Client security configuration (client_security_config.xml)**

```
<xwss:SecurityConfiguration xmlns:xwss="...">
  <xwss:UsernameToken digestPassword="false" id="username"/>
  <xwss:Encrypt>
    <xwss:X509Token certificateAlias="client certificate"
      keyReferenceType="Identifier"/>
    <xwss:Target type="qname">
      {http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd}UsernameToken
    </xwss:Target>
  </xwss:Encrypt>
</xwss:SecurityConfiguration>
```

DO NOT START FROM THE START!

Spring-WS – Encryption example

- Request message

```
<wsse:Security xmlns:wsse="..." SOAP-ENV:mustUnderstand="1">
  <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <wsse:SecurityTokenReference>
        <wsse:KeyIdentifier EncodingType="...">WxwGoLUHeny/ZNE2OolR+sh00BE=</wsse:KeyIdentifier>
      </wsse:SecurityTokenReference>
    </ds:KeyInfo>
    <xenc:CipherData>
      <xenc:CipherValue>...1ia6xkXd+ndyIYcj2nk=</xenc:CipherValue>
    </xenc:CipherData>
    <xenc:ReferenceList>
      <xenc:DataReference URI="#123"/>
    </xenc:ReferenceList>
  </xenc:EncryptedKey>
  <wsse:UsernameToken xmlns:wsu="..." wsu:Id="username">
    <xenc:EncryptedData xmlns:xenc="..." Id="123" Type="...#Content">
      <xenc:EncryptionMethod Algorithm="...#tripleDES-cbc"/>
      <xenc:CipherData>
        <xenc:CipherValue>...d69w6HrVWRwV/3/ZYwtdh5P8B9YpIkUL68LUD</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </wsse:UsernameToken>
</wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
```

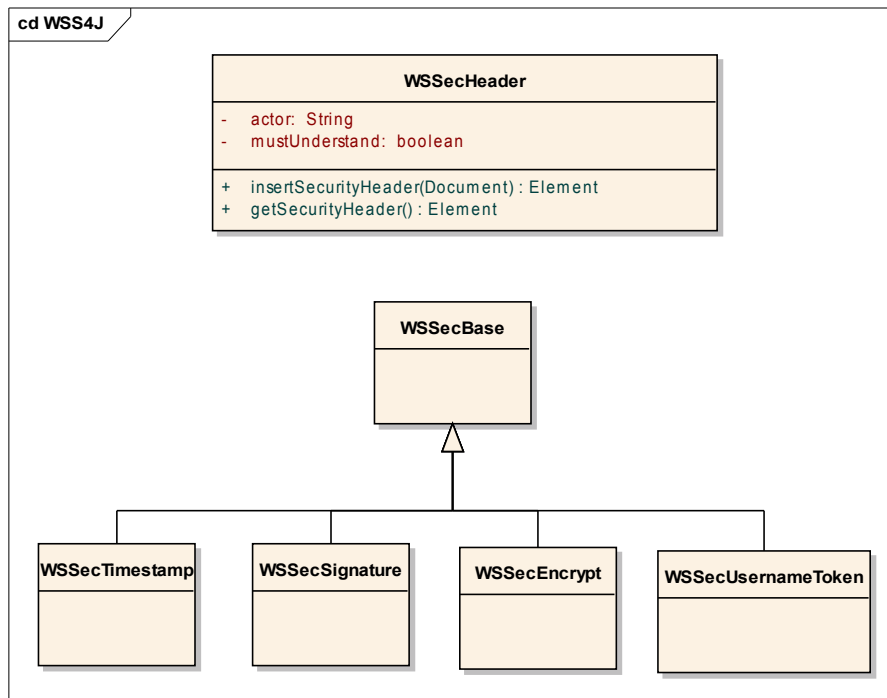
DOING SECURITY RIGHT FROM THE START!

WSS4J - Overview

- An implementation of the OASIS Web Services Security
 - Primarily a Java library for signing and verifying SOAP messages
 - Uses Apache Axis and Apache XML-Security projects
 - Interoperable with JAX-RPC based server/clients and .NET server/clients
- Implements
 - OASIS Web Services Security: SOAP Message Security 1.0 Standard 200401, March 2004
 - Username Token profile V1.0
 - X.509 Token Profile V1.0
- Reference
 - <http://ws.apache.org/wss4j/>

WSS4J - API

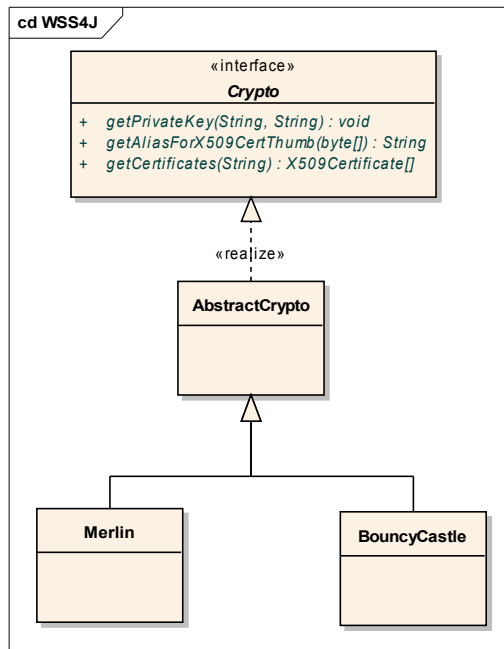
- Main classes



```

<SOAP:Envelope>
  <SOAP:Header>
    <wsse:Security>?
      <wsse:BinarySecurityToken>?
      <ds:Signature>
        <ds:SignedInfo>
          (<ds:Reference>)+
        </ds:SignedInfo>
        <ds:SignatureValue>
        <ds:KeyInfo>?
      </ds:Signature>
      <xenc:EncryptedKey>
        <ds:KeyInfo>?
      </xenc:EncryptedKey>?
      <wsu:Time Stamp>?
    </wsse:Security>
  </SOAP:Header>
  <SOAP:Body>
    <xenc:EncryptedData>?
  </SOAP:Body>
</SOAP:Envelope>
  
```

- Manager for certificates



```
# Classname of Crypto provider
org.apache.ws.security.crypto.provider=

# The name of the keystore file
org.apache.ws.security.crypto.merlin.file=

# The type of the keystore
org.apache.ws.security.crypto.merlin.keystore.type=

# The password of the keystore
org.apache.ws.security.crypto.merlin.keystore.password=

# The default alias of private key/certificate
org.apache.ws.security.crypto.merlin.keystore.alias=
```

Axis - Overview

- Integration of WSS4J with Axis
 - Special handlers to add a WS-Security layer to the web service
 - org.apache.ws.axis.security.WSDoAllSender
 - org.apache.ws.axis.security.WSDoAllReceiver
- Configuration
 - action: Signature and/or Encryption
 - user: The user/alias used for signing
 - passwordCallbackClass: Used to retrieve the password
 - signaturePropFile: The property file with keystore information
 - signatureParts: References that should be signed
 - signatureKeyIdentifier: issuer-serial (WSS4J default) and direct-reference mode
- Handler chaining

Axis – UsernameToken example

- Client configuration (client-deploy.wsdd)

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="MessageService" provider="java:RPC">
    <parameter name="allowedMethods" value="*" />
    <parameter name="className" value="nl.jfall.services.MessageServiceAxisEndPoint" />
    <beanMapping qname="ro:ResponseMessage" xmlns:ro="urn:MessageService"
      languageSpecificType="java:nl.jfall.services.ResponseMessage" />
    <beanMapping qname="ro:RequestMessage" xmlns:ro="urn:MessageService"
      languageSpecificType="java:nl.jfall.services.RequestMessage" />
    <requestFlow>
      <parameter name="action" value="UsernameToken"/>
      <parameter name="user" value="Eelco"/>
      <parameter name="passwordCallbackClass"
        value="nl.jfall.security.PasswordCallbackHandler" />
      <parameter name="passwordType" value="PasswordText" />
      <parameter name="addUTElements" value="Nonce Created" />
    </requestFlow>
  </service>
</deployment>
```

Axis – Signature example

- Client configuration (client-deploy.wsdd)

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="..." xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="MessageService" provider="java:RPC">
    <parameter name="allowedMethods" value="*" />
    <parameter name="className" value="nl.jfall.services.MessageServiceAxisEndPoint" />
    <beanMapping qname="ro:ResponseMessage" xmlns:ro="urn:MessageService"
      languageSpecificType="java:nl.jfall.services.ResponseMessage" />
    <beanMapping qname="ro:RequestMessage" xmlns:ro="urn:MessageService"
      languageSpecificType="java:nl.jfall.services.RequestMessage" />
    <requestFlow>
      <handler name="DoSecurityReceiver"
        type="java:org.apache.ws.axis.security.WSDoAllReceiver">
        <parameter name="action" value="Timestamp Signature" />
        <parameter name="user" value="client certificate" />
        <parameter name="passwordCallbackClass"
          value="nl.jfall.security.PasswordCallbackHandler" />
        <parameter name="signaturePropFile" value="cryptoClient.properties" />
        <parameter name="signatureKeyIdentifier" value="DirectReference" />
        <parameter name="timeToLive" value="5" />
        <parameter name="signatureParts" value="{Element}{http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd}Timestamp;{Element}{http://schemas.xmlsoap.org/soap/envelope/}Body"/>
      </handler>
    </requestFlow>
  </service>
</deployment>
```

Axis – Encryption example

- Client configuration (client-deploy.wsdd)

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="..." xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="MessageService" provider="java:RPC">
    <parameter name="allowedMethods" value="*" />
    <parameter name="className" value="nl.jfall.services.MessageServiceAxisEndPoint" />
    <beanMapping qname="ro:ResponseMessage" xmlns:ro="urn:MessageService"
      languageSpecificType="java:nl.jfall.services.ResponseMessage" />
    <beanMapping qname="ro:RequestMessage" xmlns:ro="urn:MessageService"
      languageSpecificType="java:nl.jfall.services.RequestMessage" />
    <requestFlow>
      <handler name="DoSecurityReceiver" type="java:WSDoAllReceiver">
        <parameter name="action" value="UsernameToken Encrypt" />
        <parameter name="user" value="Eelco"/>
        <parameter name="encryptionUser" value="client certificate" />
        <parameter name="passwordCallbackClass" value="...PasswordCallbackHandler"/>
        <parameter name="passwordType" value="PasswordText" />
        <parameter name="addUTElements" value="Nonce Created" />
        <parameter name="encryptionPropFile" value="cryptoClient.properties"/>
        <parameter name="encryptionKeyIdentifier" value="IssuerSerial"/>
        <parameter name="encryptionParts"
          value="{Element}{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd}UsernameToken"/>
      </handler>
    </requestFlow>
  </service>
</deployment>
```

Comparison

	Pros	Cons
XWSS	<ul style="list-style-type: none">• Powerful configuration file defines requirements.• Clean separation between security and business logic	<ul style="list-style-type: none">• Only JDK 1.5 or higher supported.• Dependency on Xerces schema validation.• Logging very minimal.• WS-SecurityPolicy only supported in XWSS 3.0/WSIT
WSS4J	<ul style="list-style-type: none">• Even works with JDK 1.3 with small modifications.• Very easy to configure. Minimal impact to existing code.• Clean separation between security and business logic.	<ul style="list-style-type: none">• Limited configurability.• Documentation is poor.• Error handling and logging very minimal.• In combination with Axis only two types of key info supported• WS-SecurityPolicy not supported (Axis2/Rampart must be used)

Conclusion

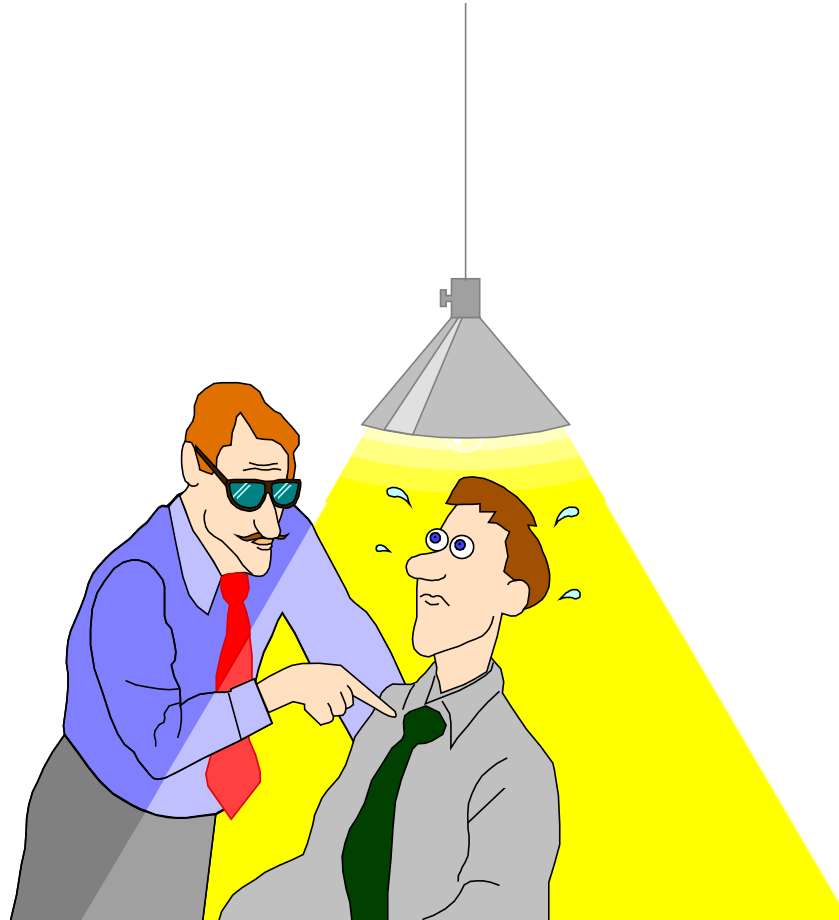
DOING SECURITY RIGHT FROM THE START!

Summary

- Transport-level security is fine for simple cases, but...
- ...current WS-Security tools are almost as simple and support basic functionality:
 - XML Signature and validation
 - XML Encryption
 - Timestamp token
 - Configurable to some extent
- Next versions support all WS-*, such as WS-SecurityPolicy:
 - WSIT and XWSS 3.0
 - Axis2



Questions



DOING SECURITY RIGHT FROM THE START!

References

- XML Signature – <http://www.w3.org/TR/xmlsig-core/>
- XML Encryption – <http://www.w3.org/TR/xmlenc-core/>
- WS-Security – <http://docs.oasis-open.org/wss/v1.1/>
- XWSS – <https://xwss.dev.java.net/>
- WSS4J – <http://ws.apache.org/wss4j>
- WSIT – <https://wsit.dev.java.net/>
- XFire – <http://xfire.codehaus.org/WS-Security>
- Spring-WS – <http://static.springframework.org/spring-ws/>
- Axis – <http://ws.apache.org/axis/>
- Comparison – <http://wiki.apache.org/ws/StackComparison>