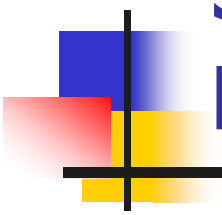
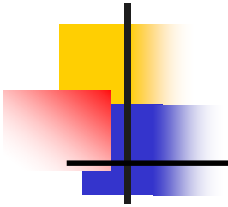


Secure Web Services
How to Do It, and at What Cost

Hermod Opstvedt

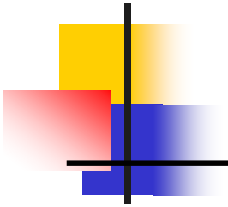
Chief Architect

DnB NOR ITU, Norway



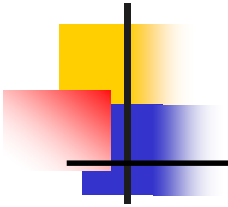
Secure Web Services

- In this presentation we will look at:
  - Exposing Web services
  - Consuming Web services
  - Adding Security
  - Look at performance impact



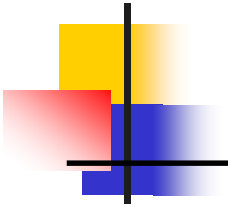
Secure Web Services

- Issues around exposing Web services
  - The number one concern that prevents many enterprises from implementing Web Services in a meaningful way is the lack of understanding of what the security risks are.



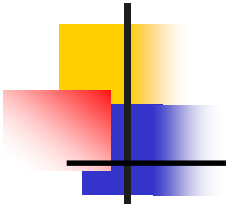
Secure Web Services

- But it is just HTTP!?
  - Web services are application to application.
  - How do you distinguish malicious users from others.
    - Information gathering/misuse.



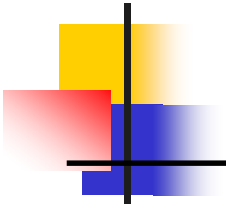
Secure Web Services

- Exposing a Web service - Issues
  - What are you trying to achieve?
    - Just for fun?
    - Adding a public service to the Internet?
    - Or doing business?



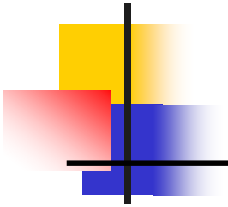
Secure Web Services

- What are you exposing:
  - An existing service?
  - An existing EJB?
  - Some existing database table?



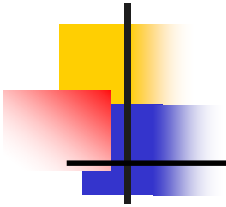
Secure Web Services

- Is security really needed?
  - Yes, sensitive data?
    - ✓ Must only be seen by intended receiver.
  - Yes, secure transactions?
    - ✓ Must be sure both ends are who they say the are.



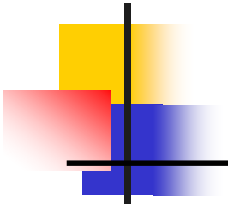
Secure Web Services

- So how can you secure a Web service?
  - Extranet?
    - Adds infrastructure
    - Adds complexity
    - Adds cost
  - VPN?
    - Adds infrastructure
    - Adds complexity
    - Adds cost



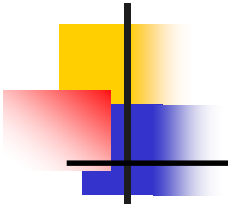
Secure Web Services

- Firewall
  - Defining IP-addresses or MAC's
    - ✓ Adds complexity
    - ✓ Adds maintenance cost



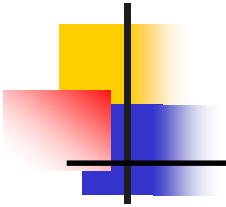
Secure Web Services

- Need something more generic
  - HTTPS – Transport layer security.
  - WS-Security – Message layer security.



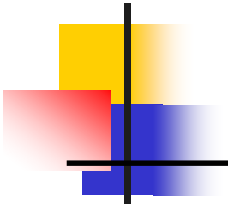
Secure Web Services

- What is actually security about
  - Authentication
    - Who is the caller and how do we prove they are who they say they are?
  - Confidentiality
    - How do we prevent snoopers viewing our messages and data?



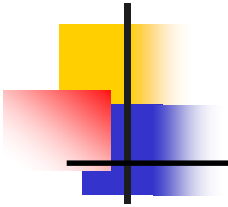
Secure Web Services

- Authorization
  - What is the caller authorized to do?
- Integrity
  - How do we prevent messages being tampered with between sender and receiver?
- Secure and reliable message delivery
  - How do we guarantee that the message arrives?



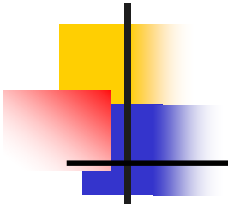
Secure Web Services

- Authentication
  - Who is at the other end
    - Basic authentication:
      - ✓ The user is prompted by the application they are using for a username and password.
      - ✓ The user fills these credentials in correctly remembering that the password is case sensitive.
      - ✓ The application encodes or prearranges the credentials and sends it to the server.
      - ✓ The server compares these credentials to its own list of accounts locally, in the domain or trusted domain, and then grants access to the resources that the client has been configured access for.



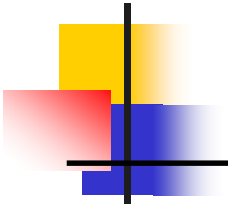
Secure Web Services

- Client certificates and server certificates
  - ✓ Server certificates are used to ensure that the server that the client talks to is the real one.
  - ✓ Client certificates are generated on the server side and installed on the client side, and is used by the server to verify that this the correct client.
  - ✓ Both are used with SSL for key-pair authentication (PKA).



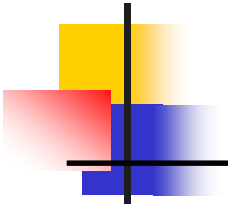
Secure Web Services

- Digest authentication
  - ✓Used in Windows environment.
  - ✓Generates a one-way hash.
- Integrated authentication (Client)
  - ✓Kerberos or the built in (NTLM) Challenge/response authentication protocol.



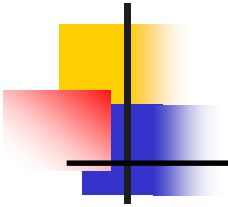
Secure Web Services

- Authorization
  - Is the process of verifying that the client has access to the requested resource.
    - Client must be identifiable.



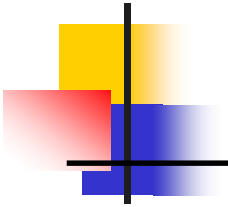
Secure Web Services

- Secure and reliable message delivery
  - Ensure that what we send out arrives at the other end in the order they where sent.
    - WS-Reliability
      - ✓ Is a SOAP-based protocol for exchanging SOAP messages with guaranteed delivery, no duplicates, and guaranteed message ordering.



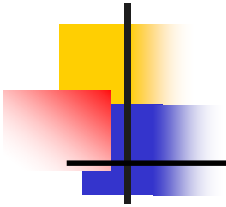
Secure Web Services

- Non-repudiation.
  - Non-repudiation means that it can be verified that the sender and the recipient were, in fact, the parties who claimed to send or receive the message, respectively. In other words, non-repudiation of origin proves that data has been sent, and non-repudiation of delivery proves it has been received.



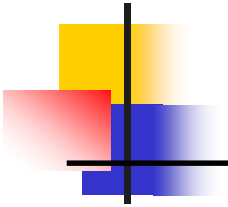
Secure Web Services

- How to's – Prerequisites for this session
  - A container
    - Tomcat.
  - A Web Service API
    - Axis – Apache foundation.
    - Installed in Tomcat.
  - A SSL implementation
    - Not necessary if JDK 1.4 and up – as now.



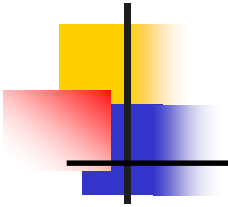
Secure Web Services

- The sample application
  - We want to expose a service that tells us how much pension we will get given a monthly payment.
  - The service will then be extended so that only companies that have a pension plan for their employees can access it.
  - The service will also be changed so that only an employee number is required.



Secure Web Services

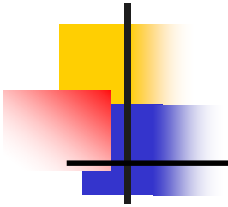
- The sample service is a POJO that just returns a random value.
- The sample client connects to the service and requests a value from the exposed method and returns the answer.



Secure Web Services

- The Service:

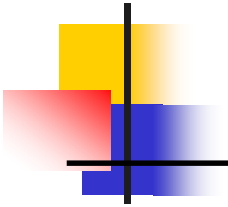
```
public class PensionService
{
   public Long getPension(Long savings)
   {
      return new Long(new
      Float(savings.longValue()*0.66f).lon
   gValue());
   }
}
```



Secure Web Services

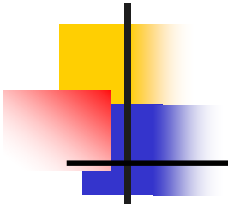
- The Client:

```
public class PensionClient
{
	public static void main(String[] args) throws Exception
	{
		if(args.length < 1)
		{
			System.err.println("Usage : java no.dnbnor.ee.css2004.wss.PensionClient <salary>");
			System.exit(-1);
		}
		String endpointAdress = "http://localhost:8080/axis/services/PensionService";
		String wsdlAddress = endpointAdress + "?wsdl";
		String nameSpaceUri = "http://localhost:8080/css2004-wss/";
		String serviceName = "PensionService";
		String portName = "PensionServicePort";
		URL wsdlUrl = new java.net.URL(wsdlAddress);
		ServiceFactory svcFactory = ServiceFactory.newInstance();
		QName queueName = new QName(nameSpaceUri, serviceName);
		Service svc = svcFactory.createService(wsdlUrl, queueName);
		Call call = (Call) svc.createCall();
```



Secure Web Services

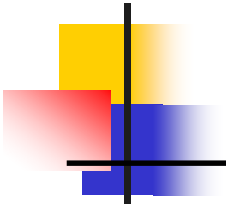
```
        call.setTargetEndpointAddress(endpointAdress);
        call.setOperationName(new QName(nameSpaceUri, "getPension"));
        call.setPortTypeName(new QName(nameSpaceUri, portName));

        Object salary=null;
        try
        {
              salary = new Long(args[0]);
        }
        catch (NumberFormatException e)
        {
              e.printStackTrace();
              System.exit(-1);
        }
        System.out.println("Calling PensionService with : " + salary);
        Long pension  = (Long) call.invoke(new Object[] { salary });
        System.out.println("Your pension will be : " + pension);
    }
}
```



Secure Web Services

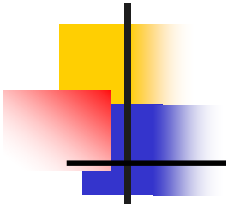
- The deployment descriptor

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
        <service name="PensionServicePort" provider="java:RPC">
            <parameter name="wsdlTargetNamespace"
value="http://localhost:8080/css2004-wss/"/>
            <parameter name="wsdlServiceElement" value="PensionService"/>
            <parameter name="wsdlServicePort" value="PensionServicePort"/>
            <parameter name="wsdlPortType" value="PensionService"/>
            <parameter name="scope" value="session"/>
            <parameter name="className"
value="no.dnbnor.ee.css2004.wss.PensionService"/>
            <parameter name="allowedMethods" value="*"/>
        </service>
</deployment>
```



Secure Web Services

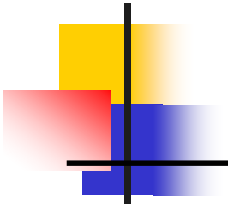
- Installing the service on Axis
  - Create a jar file containing the service.
  - Copy it to Axis WEB-INF/lib directory.
  - Start Tomcat.
  - Run the Axis AdminClient.
  - Verify *via* Axis web GUI – View command.
  - Test it with client.



Secure Web Services

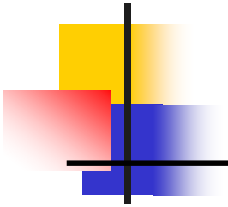
- The WSDL :

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost:8080/css2004-wss/" xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://localhost:8080/css2004-wss/"
xmlns:intf="http://localhost:8080/css2004-wss/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:message name="getPensionResponse">
  <wsdl:part name="getPensionReturn" type="xsd:long"/>
 </wsdl:message>
 <wsdl:message name="getPensionRequest">
  <wsdl:part name="in0" type="xsd:long"/>
 </wsdl:message>
 <wsdl:portType name="PensionService">
  <wsdl:operation name="getPension" parameterOrder="in0">
   <wsdl:input message="impl:getPensionRequest" name="getPensionRequest"/>
   <wsdl:output message="impl:getPensionResponse" name="getPensionResponse"/>
  </wsdl:operation>
 </wsdl:portType>
```



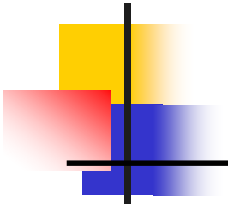
Secure Web Services

```
<wsdl:binding name="PensionServicePortSoapBinding" type="impl:PensionService">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getPension">
   <wsdlsoap:operation soapAction=""/>
   <wsdl:input name="getPensionRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://wss.css2004.ee.dnbnor.no" use="encoded"/>
   </wsdl:input>
   <wsdl:output name="getPensionResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://localhost:8080/css2004-
wss/" use="encoded"/>
   </wsdl:output>
  </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="PensionService">
  <wsdl:port binding="impl:PensionServicePortSoapBinding" name="PensionServicePort">
   <wsdlsoap:address location="http://localhost:8080/axis/services/PensionService"/>
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```



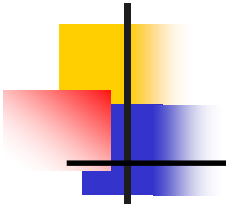
Secure Web Services

- Transport layer security – HTTPS.
  - When to use it
    - Single endpoint.
    - Synchronous delivery.
    - Single transport.
    - Just like a regular HTTP request/response.



Secure Web Services

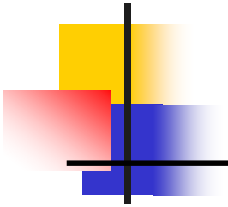
- Transport layer security.
  - What to do :
    - First we add authorization.
      - ✓ We have three options with Tomcat – Basic, Form & Digest
        - ❖ Use Basic for sample.
      - ✓ We can not use Form because our Client would never see it.
    - We must alter Tomcat so that it will secure us
      - ✓ Define a role in Tomcat's security realm (tomcat-users.xml)
      - ✓ Create a security-constraint with our new role in the Axis web.xml file.
    - Alter the client to use Basic authorization.



Secure Web Services

- Tomcat-users.xml

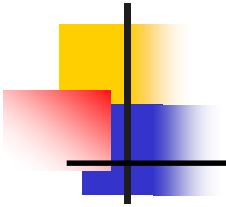
```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="manager"/>
  <role rolename="admin"/>
  <role rolename="wssrole"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="wsstester" password="css2004" roles="wssrole"/>
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="admin" password="sm63509" roles="admin,manager"/>
</tomcat-users>
```



Secure Web Services

- Web.xml - addendum

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Pension Service</web-resource-name>
    <description/>
    <url-pattern>/services/PensionService</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <description/>
    <role-name>wssrole</role-name>
    <role-name>tomcat</role-name>
  </auth-constraint>
</security-constraint>
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>CSS2004 Basic Security</realm-name>
</login-config>
<security-role>
  <description>CSS 2004 WS-S role</description>
  <role-name>wssrole</role-name>
</security-role>
```



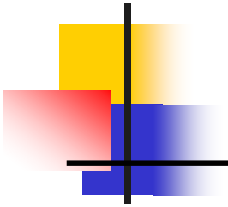
Secure Web Services

- Alter client program
  - We can not use the wsdl served by Axis – It is protected by username/password now. Must use a previously saved wsdl file.

```
//Use this when adding username/password
String wsdlAddress = "file:PensionService.wsdl";
```

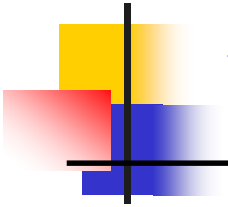
  - Addendum – It is possible to use it if we add the username/password to the uri. Ex.

```
String endpointAdress = "http://"+username+":"+password+
"@localhost:8080/axis/services/PensionService";
```



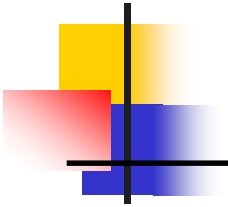
Secure Web Services

```
call.setTargetEndpointAddress(endpointAdress);

call.setOperationName(new QName(nameSpaceUri,
"getPension"));

call.setPortTypeName(new QName(nameSpaceUri, portName));


//Use this when adding username/password

call.setProperty(Call.USERNAME_PROPERTY, args[1]);

call.setProperty(Call.PASSWORD_PROPERTY, args[2]);


Object salary=null;
```



Web Service Security

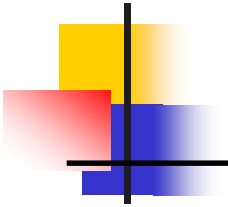
- We are now ready to
  - Add privacy and message integrity through the use of HTTPS.
    - ✓ Create server and client certificates in their respective keystore.
      - ❖ Use Java's keytool
    - ✓ Export both certificates from their respective keystore.
    - ✓ Import client certificate into server keystore.
    - ✓ Import server certificate into client keystore.



Secure Web Services

- The file that does it

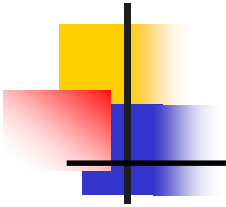
```
@echo %1
@if not "%JAVA_HOME%" == "" goto gotJavaHome
@echo You must set JAVA_HOME to point at your Java Development Kit installation
@goto cleanup
@:gotJavaHome
@echo Generating the ServerKeyStore in file server.keystore
@%java_home%\bin\keytool -genkey -alias tomcat-sv -dname "CN=localhost, OU=DNB NOR ITU, O=DNB NOR, L=BERGEN, S=HO, C=NO" -keyalg RSA -keypass changeit -storepass changeit -keystore server.keystore
@echo Exporting the certificate from keystore to an external file server.cer
@%java_home%\bin\keytool -export -alias tomcat-sv -storepass changeit -file server.cer -keystore server.keystore
@echo Generating the ClientKeyStore in file client.keystore
@%java_home%\bin\keytool -genkey -alias tomcat-cl -dname "CN=Client, OU=DNB NOR ITU, O=DNB NOR, L=BERGEN, S=HO, C=NO " -keyalg RSA -keypass changeit -storepass changeit -keystore client.keystore
@echo Exporting the certificate from keystore to external file client.cer
@%java_home%\bin\keytool -export -alias tomcat-cl -storepass changeit -file client.cer -keystore client.keystore
@echo Importing Clientcertificate into Serverkeystore
@%java_home%\bin\keytool -import -v -trustcacerts -alias tomcat -file server.cer -keystore client.keystore -keypass changeit -storepass changeit
@echo Importing Servercertificate into Client keystore
@%java_home%\bin\keytool -import -v -trustcacerts -alias tomcat -file client.cer -keystore server.keystore -keypass changeit -storepass changeit
```



Secure Web Services

- Alter Tomcat's server.xml file
  - Uncomment/add the https section using our newly created server keystore

```
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
        port="8443" minProcessors="5" maxProcessors="75"
        enableLookups="true"
            acceptCount="100" debug="0" scheme="https" secure="true"
        useURIValidationHack="false" disableUploadTimeout="true">
    <Factory className="org.apache.coyote.tomcat4.CoyoteServerSocketFactory"
        keystoreFile="E:/CSS/2004/WS-S/server.keystore"
        keystorePass="changeit"
        clientAuth="false" protocol="TLS" />
</Connector>
```



Secure Web Services

- Alter the client to use the secured wsdl

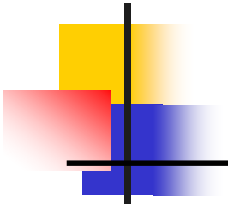
```
String wsdlAddress = "file:PensionService-https.wsdl";
```

- Add system (JVM) property to client

-Djavax.net.ssl.trustStore=client.truststore

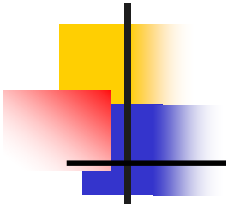
Or add the following line to the client code at the beginning :

```
System.setProperty("javax.net.ssl.trustStore","client.truststore");
```



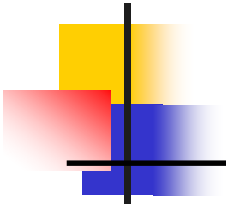
Secure Web Services

- Message layer security
  - When to use it
    - Multiple endpoints
    - Asynchronous
    - Multiple transports
    - Intermediaries



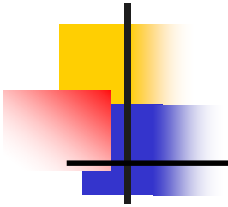
Secure Web Services

- Webservice-Security (WS-Security)
  - WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security also provides a general-purpose mechanism for associating security tokens with messages. No specific type of security token is required by WS-Security. Additionally, WS-Security describes how to encode binary security tokens. Specifically, the specification describes how to encode X.509 certificates and Kerberos tickets as well as how to include opaque encrypted keys.



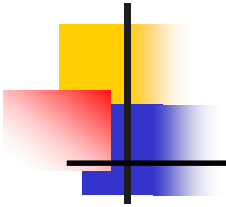
Secure Web Services

- Tokens
  - Represents a collection of claims where a claim is a declaration that an entity makes (*e.g.* name, identity, key, group, privilege, capability, *etc.*).
  - Unsigned Security Tokens – Username
  - Signed Security Tokens – x.509 certificates, Kerberos tickets.
    - ✓ A token that is asserted and cryptographically signed by a specific authority (*e.g.* an X.509 certificate or a Kerberos ticket).



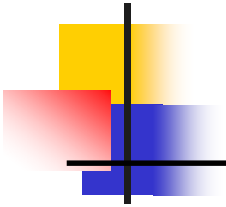
Secure Web Services

- Message layer security – WS-Security
  - How to do it
    - Examples are based on Axis and the Java Web Services Developer Pack.
    - Uses Security API (VeriSign Trust Services Integration Kit)
    - Messages are encrypted/decrypted and also signed.
    - Runs on Tomcat



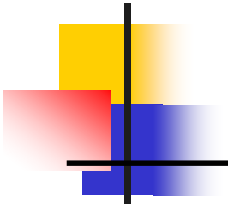
Secure Web Services

- A little about encryption & signing
  - Messages are encrypted by the sender with the public key of the receiver.
  - Messages are decrypted by the receiver with the private key of the receiver.
  - Messages are signed by the sender with the private key of the sender
  - Messages are verified by the receiver with the public key of the sender



Secure Web Services

- WS Security Handlers
  - Can be thought of as a form of Filters known from Servlet API
  - Axis call these before passing on the request to the service, and also before a response is sent back to the client.
  - Handlers can be chained.

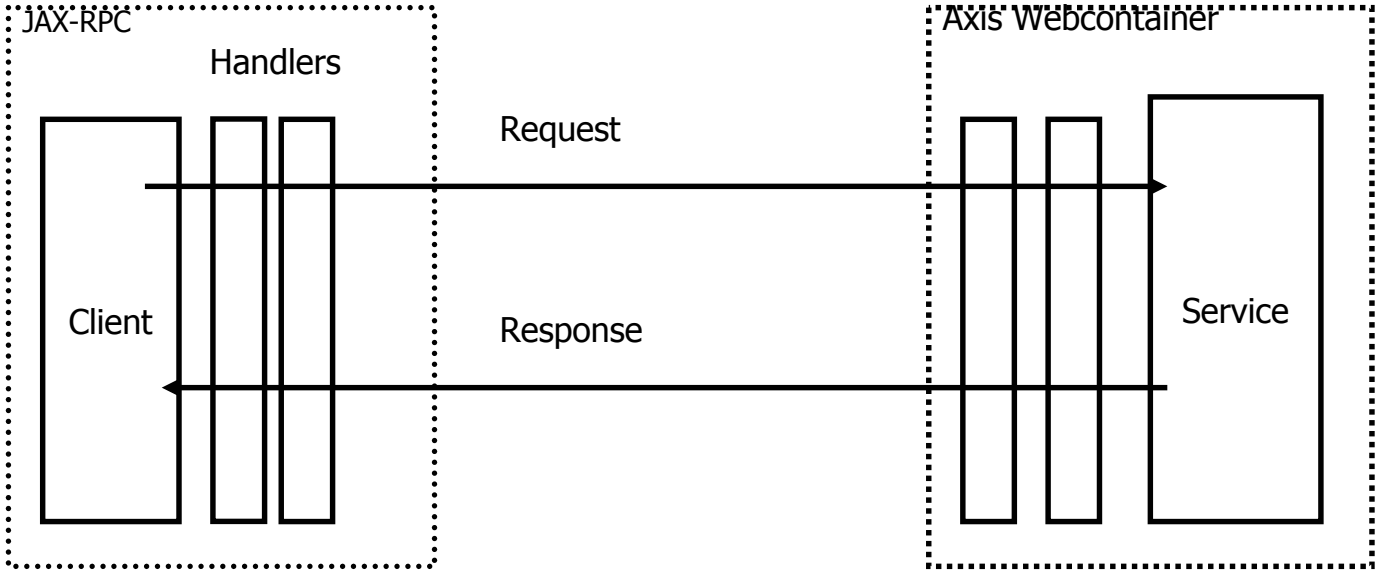


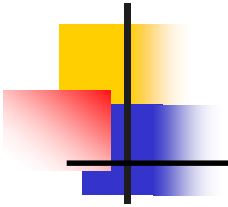
Secure Web Services

- Client side Handlers
  - Through the use of JAX-RPC one can define Handlers for the client also.

Secure Web Services

- Since Handlers are outside the application, they are transparent to it.

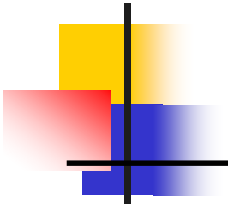




Secure Web Services

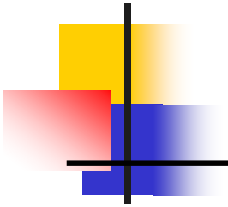
- The Client

```
public class BankClient
{
        public static void main(String[] args) throws Exception
        {
                try
                {
                if (args.length < 3)
                {
                        System.err.println(
                        "Usage : java no.dnbnor.ee.css2004.wss.BankClient amount fromaccount toaccount");
                        System.exit(-1);
                }
                String endpointAdress =
                "https://localhost:8080/axis/services/BankService";
                String wsdlAddress = endpointAdress + "?wsdl";
```



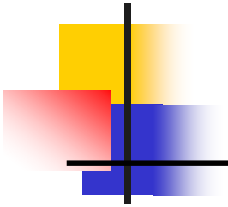
Secure Web Services

```
String nameSpaceUri = "http://localhost:8080/css2004-wss/";
String serviceName = "BankService";
String portName = "BankServicePort";
// Define some parameters for our handler
HashMap params = new HashMap();
params.put("keystore","client.keystore");
params.put("truststore","client.truststore");
params.put("alias","tomcat");
URL wsdlUrl = new java.net.URL(wsdlAddress);
ServiceFactory svcFactory = ServiceFactory.newInstance();
QName queueName = new QName(nameSpaceUri, serviceName);
Service service = svcFactory.createService(wsdlUrl, queueName);
// Get the client side handler class
Class swsHandler = SWSClientHandler.class;
```



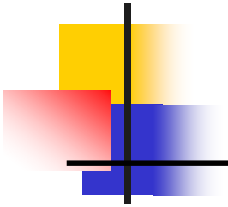
Secure Web Services

```
// Get the JAX-RPC handlerregistry
HandlerRegistry handlerRegistry = service.getHandlerRegistry();

// Define a JAX-RPC queuename for our namespace and port
QName qName = new QName(nameSpaceUri,portName);
// Get the handlerChain for the queuename
List handlerChain =handlerRegistry.getHandlerChain(qName);
// Declare handlerinfor for our handler
HandlerInfo handlerInfo = new HandlerInfo(swsHandler,params, null);
// Add handlerinfo to handlerchain
handlerChain.add(handlerInfo);
Call call = (Call) service.createCall();

call.setTargetEndpointAddress(endpointAdress);
call.setOperationName(new QName(nameSpaceUri, "transferMoney"));
call.setPortTypeName(new QName(nameSpaceUri, portName));

Object amount = null;
```



Secure Web Services

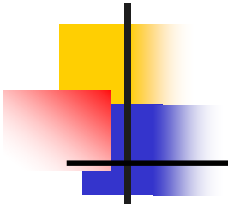
```
try
{
        amount = new Long(args[0]);
}
catch (NumberFormatException e)
{
    e.printStackTrace();
    System.exit(-1);
}
System.out.println("Calling BankService with : " + amount);
Long transferred = (Long) call.invoke(new Object[] { amount, args[1], args[2]});
System.out.println("Amount transferred: " + transferred);
}
catch (Exception e)
{
        e.printStackTrace();
}
}
}
```



Secure Web Services

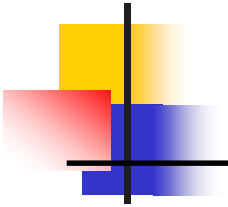
- The Service

```
public class BankService
{
        public Long transferAmount(Long amount, String fromAccount,
        String toAccount)
    {
        return new Long(amount.longValue()-2);
    }
}
```



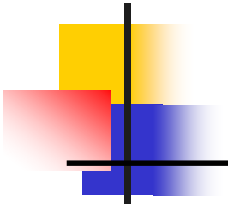
Secure Web Services

- The Service Handler
  - Implements javax.xml.rpc.handler.Handler
    - public boolean handleRequest(MessageContext arg);
    - public boolean handleResponse(MessageContext arg);
    - public boolean handleFault(MessageContext arg);
    - public void init(HandlerInfo arg);
    - public void destroy();
    - public QName[] getHeaders()



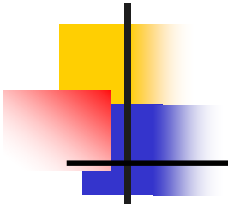
Secure Web Services

```
public boolean handleRequest(MessageContext messageContext)
{
    boolean retval = false;
    if (messageContext instanceof SOAPMessageContext)
    {
    try
    {
    SOAPMessageContext soapContext =
    (SOAPMessageContext) messageContext;
    SOAPMessage soapMessage = soapContext.getMessage();
    Document doc = toDocument(soapMessage);
```



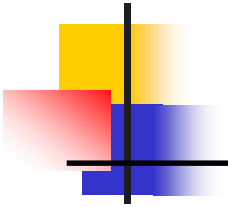
Secure Web Services

```
                // Call the implementation (Crypto API) dependent decryptor
                CSSDecryptor.decrypt(doc,keyStoreFilename,keyStoreType,keyStorePwd,keyEntryAlias,keyEntryPwd);
                // Call the implementation (Crypto API) dependent verifier
                retval =CSSVerifier.verify(doc,trustStoreFilename,trustStoreType,trustStorePwd, certEntryAlias);
                // Clean up extracted document, create soapmessage and add it to the context
                CSSUtility.removeExcessInfo(doc);
                soapMessage = CSSUtility.toSOAPMessage(doc);
                soapContext.setMessage(soapMessage);
            } catch (Exception e)
            {
            e.printStackTrace();
        }
    }
    else {
            System.out.println("Unknown messagecontext");
    }
            return retval;
    }
```



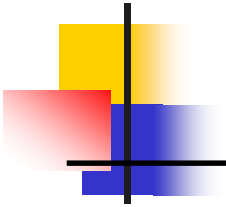
Secure Web Services

```
public boolean handleResponse(MessageContext messageContext)
{
    if (messageContext instanceof SOAPMessageContext)
    {
        try
        {
            SOAPMessageContext soapCtx = (SOAPMessageContext) messageContext;
            SOAPMessage soapMessage = soapCtx.getMessage();
            Document doc = CSSUtility.toDocument(soapMessage);
            // Call the implementation (Crypto API) dependent signer
            CSSSigner.sign(doc,keyStoreFilename,keyStoreType,keyStorePwd,keyEntryAlias,
                keyEntryPwd);
            // Call the implementation (Crypto API) dependent encryptor
            CSSEncryptor.encrypt(doc,trustStoreFilename,trustStoreType,trustStorePwd,
                certEntryAlias);
```



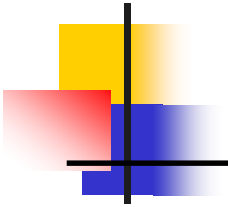
Secure Web Services

```
                    // Create soapmessage and add it to the context
                    soapMessage = CSSUtility.toSOAPMessage(doc);
                    soapCtx.setMessage(soapMessage);
                }
                catch (Exception e)
                {
                e.printStackTrace();
                return false;
                }
        }
        else
        {
        System.out.println("Unknown messagecontext");
        return false;
        }
        return true;
}
```



Secure Web Services

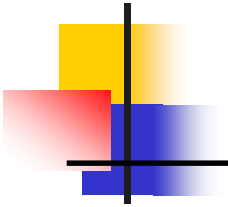
- The Client Handler
  - The same as the Service Handler except
    - handleRequest is called on outgoing
      - ✓ So the code from Service Handlers handleResponse goes here.
    - handleResponse is call on incomming
      - ✓ So the code from Service Handlers handleRequest goes here.



Secure Web Services

- The Axis deployement descriptor

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/" xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
    <service name="BankService" provider="java:RPC">
        <parameter name="wsdlTargetNamespace" value="http://localhost:8080/css2004-wss/"/>
        <parameter name="wsdlServiceElement" value="BankService"/>
        <parameter name="wsdlServicePort" value="BankServicePort"/>
        <parameter name="wsdlPortType" value="BankService"/>
        <parameter name="scope" value="session"/>
        <parameter name="className" value="no.dnbnor.ee.css2004.wss.BankService"/>
        <parameter name="allowedMethods" value="*"/>
        <requestFlow>
            <handler type="java:org.apache.axis.handlers.JAXRPCHandler">
            <parameter name="scope" value="session"/>
            <parameter name="className" value="no.dnbnor.ee.css2004.wss.SWSServiceHandler"/>
            <parameter name="keyStoreFilename" value="server.keystore"/>
```

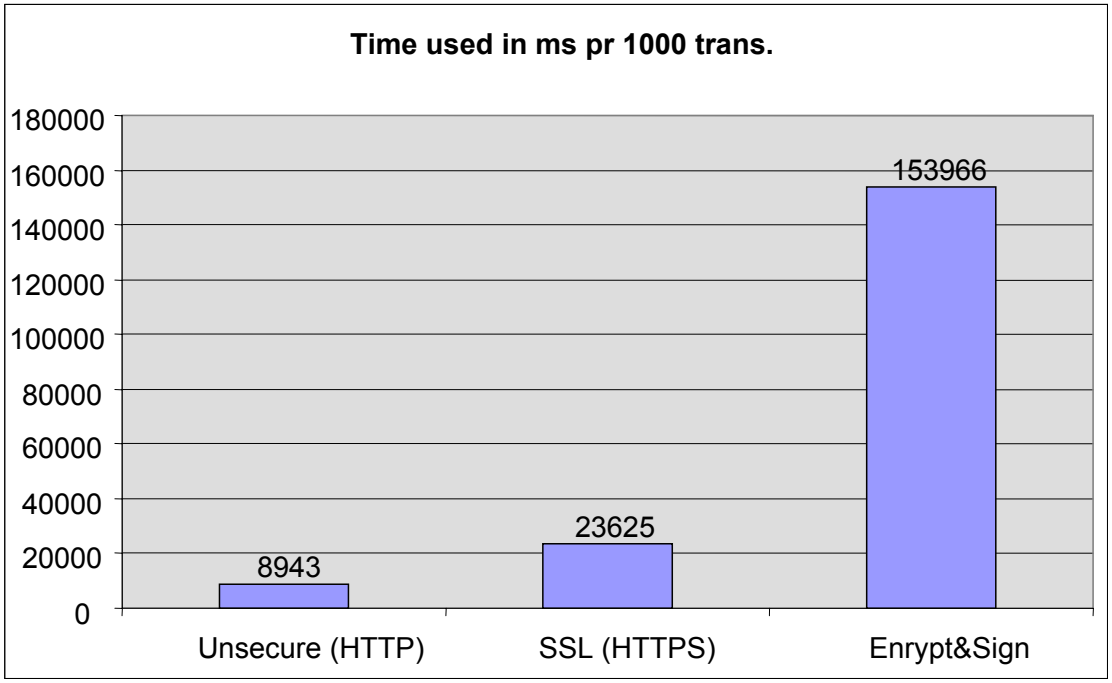


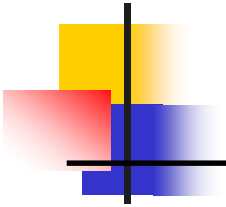
Secure Web Services

```
                <parameter name="trustStoreFilename" value="server.truststore"/>
                <parameter name="certEntryAlias" value="client"/>
            </handler>
            </requestFlow>
            <requestFlow>
                <handler type="java:org.apache.axis.handlers.JAXRPCHandler">
                <parameter name="scope" value="session"/>
                <parameter name="className" value="no.dnbnor.ee.css2004.wss.SWSServiceHandler"/>
                <parameter name="keyStoreFilename" value="server.keystore"/>
                <parameter name="trustStoreFilename" value="server.truststore"/>
                <parameter name="certEntryAlias" value="client"/>
            </handler>
            </requestFlow>
        </service>
</deployment>
```

Secure Web Services

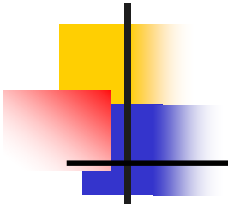
- Performance comparison chart





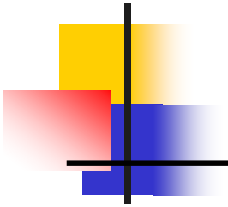
Secure Web Services

- What does it all cost ?
  - Added complexity
  - Larger packets = longer transport time.
  - Signing takes time
  - Encrypting/Decrypting takes time
  - Added cost of certificates.



Secure Web Services

- Resources :
  - http://www.oasis.org
  - http://www.apache.org/axis



Secure Web Services

- Questions