# Secure coding training

## *Day 1 – Security of Web Services*

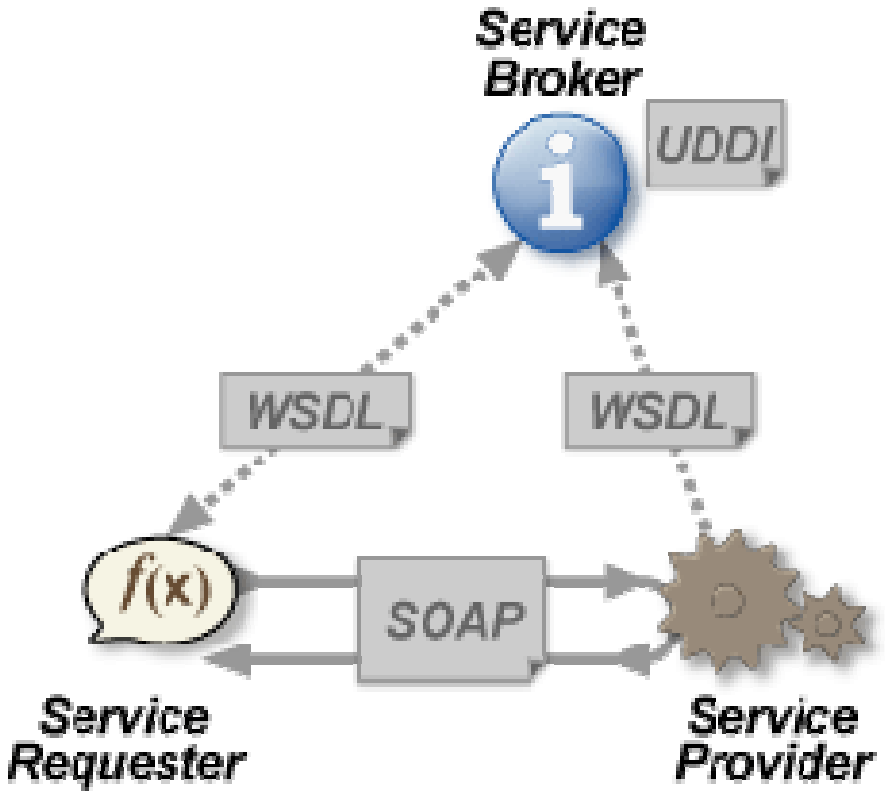
Tomasz Nowak, PSNC

Poznań, 22-23 June, 2010

# Agenda

- WS-Security origin
- Security Header block
- Security tokens
- Signatures
- Encryption
- DEMO

# Web Services
# Quickest introduction




Example message:

```
<soap:Envelope>
  <soap:Header>
    <m:User>Ervin</m:User>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice>
      <m:StockName>
        IBM
      </m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

# Application layer security

- WS-Security (Web Services Security) SOAP extension
- Provides:
  - signing
  - encryption
  - handling security tokens

# Web Services Security: OASIS Standards & WS-I Organization

- Web Services Security v1.0 (WS-Security 2004)
  - Web Services Security: SOAP Message Security 1.0
  - Web Services Security UsernameToken Profile 1.0
  - Web Services Security UsernameToken Profile 1.0
- Web Services Security v1.1 (February 2006)
  - **WS-Security Core Specification 1.1**
  - **WS-Security SOAP Message Security 1.1**
  - **Username Token Profile 1.1**
  - SAML Token Profile 1.1
  - X.509 Token Profile 1.1
  - Kerberos Token Profile 1.1
  - Rights Expression Language (REL) Token Profile 1.1
  - SOAP with Attachments (SWA) Profile 1.1
- WS-I (interoperability) Profiles: Basic, **Basic Security**, Attachments and more (in several versions)

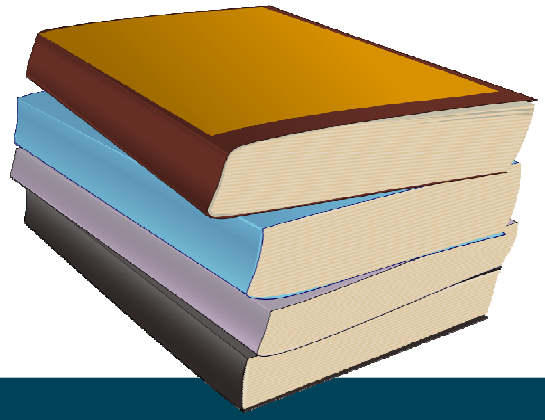
http://www.oasis-open.org/specs

# WS-Security Core
# <wsse:Security> Header Block (1)

- **<wsse:Security>** header blocks is a mechanism for attaching security-related information
- Targeted at a specific recipient – SOAP actor or role (ultimate recipient of the message or an intermediary)
- A message MAY have multiple <wsse:Security> header blocks if they are targeted for separate recipients (actor or role can't reapeat)
- Only one MAY omit the S11:actor or S12:role attributes
  - MAY be processed by anyone
  - MUST NOT be removed prior to the final destination or endpoint

# WS-Security Core
# <wsse:Security> Header Block (2)

- An active intermediary on the message path
  - MAY add new headers for additional targets
  - MAY add sub-elements to an existing <wsse:Security> header block if they are targeted for its SOAP node
- Elements added to a <wsse:Security> header block
  - SHOULD be prepended to the existing elements
  - represent the signing and encryption steps the message producer took to create the message



# WS-Security Core
# <wsse:Security> Header Block (3)

- <wsse:Security> may include mustUnderstand attribute
- Default value = 0
- When mustUnderstand = "true", receiver (role/actor):
  - MUST generate a SOAP fault if does not implement specification corresponding to the namespace
  - MUST generate a fault if unable to interpret or process security tokens contained in the <wsse:Security> header
  - MAY ignore elements or extensions within the <wsse:Security> element, based on local security policy.

# WS-Security tokens (claims)
# User Name Token

- <wsse:UsernameToken> is a way of providing a username
- Optionally included in the <wsse:Security> header
- Syntax:

```
<wsse:UsernameToken
    wsu:Id="...">
  <wsse:Username>
    ...
  </wsse:Username>
</wsse:UsernameToken>
```

- A form of **claim confirmation** should be used

# WS-Security tokens (claims)
## Other security tokens


- <wsse:BinarySecurityToken>
  - X.509 certificates
  - Kerberos tickets
  - Needs special encoding: EncodingType attribute (default: base64 encoded)
- Custom XML Tokens
- <xenc:EncryptedData> for encypted tokens

# WS-Security
# Security Timestamps

- Determine the freshness of security semantics
- Recipient may decide to ignore security header block if too old
- Assumption: time is trusted or additional mechanisms are employed to prevent replay
- xsd:dateTime type (XML Schema) and MUST be in UTC time

# WS-Security Core
# XML Signatures

- Based on ***XML Signature Syntax and Processing*** (DS, XML-DSIG) by W3C
- Allows multiple signatures and formats in one message

```
<Signature ID?>
  <SignedInfo>
    <CanonicalizationMethod/>
    <SignatureMethod/>
    (<Reference URI? >
      (<Transforms>)?
      <DigestMethod>
      <DigestValue>
    </Reference>)+
  </SignedInfo>
  <SignatureValue>
   (<KeyInfo>)?
   (<Object ID?>)*
</Signature>
```

# WS-I Basic Security Profile 1.1
# XML Signatures

- WS-Security core does not specify details
- Precised with WS-I Basic Security Profile
- Signature types according to Security Profile:
  - MUST NOT be an Enveloping Signature (disrupts SOAP processing):

```
<ds:Signature>
  <ds:SignedInfo>...</ds:SignedInfo>
  <ds:SignatureValue>...</ds:SignatureValue>
  <ds:KeyInfo>...</ds:KeyInfo>
  <ds:Object>...</ds:Object>
</ds:Signature>
```

  - SHOULD NOT be an Enveloped Signature
  - SHOULD be a Detached Signature

# Detached XML Signature

```
<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='#TheBody'>
  <ds:Transforms>
     <ds:Transform Algorithm='http://www.w3.org/2001/10/xml-exc-c14n#' />
  </ds:Transforms>
  <ds:DigestMethod Algorithm='http://www.w3.org/2000/09/xmldsig#sha1' />
  <ds:DigestValue>i3qi5GjhHnfoBn/jOjQp2mq0Na4=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>PipXJ2Sfc+LTDnq4pM5JcIYt9gg=</ds:SignatureValue>
<ds:KeyInfo>
  <wsse:SecurityTokenReference>
  <wsse:Reference URI='#SomeCert'
    ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" />
  </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

# XML Signatures
# Signed Element References

- URI attribute containing a „Shorthand Xpointer” to an element with wsu:Id attribute
- XPath where necessary, but involves a second <Transform> (before canonicalization), i.e.:

```
<ds:Transform
Algorithm='http://www.w3.org/2002/06/xmldsig-filter2'
xmlns:dsxp='http://www.w3.org/2002/06/xmldsig-filter2'>

<dsxp:XPath Filter='intersect'>
ancestor-or-self::soap:Body[parent::node()=/soap:Envelope]
</dsxp:XPath>

</ds:Transform>
```

# XML Signatures
# KeyInfo structure

- ds:KeyInfo element allows for many different child elements
- Must contain only one of them
- Basic Security Profile mandates wsse:SecurityTokenReference (to reference security tokens)
- Example – X.509 certificate marked with **wsu:Id=”SomeCert”**

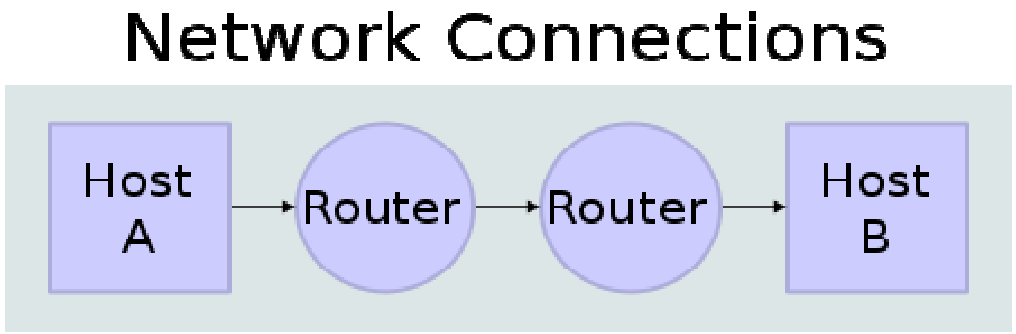
```
<ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#' >

<wsse:SecurityTokenReference>

<wsse:Reference URI='#SomeCert'
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" />

</wsse:SecurityTokenReference>

</ds:KeyInfo>
```

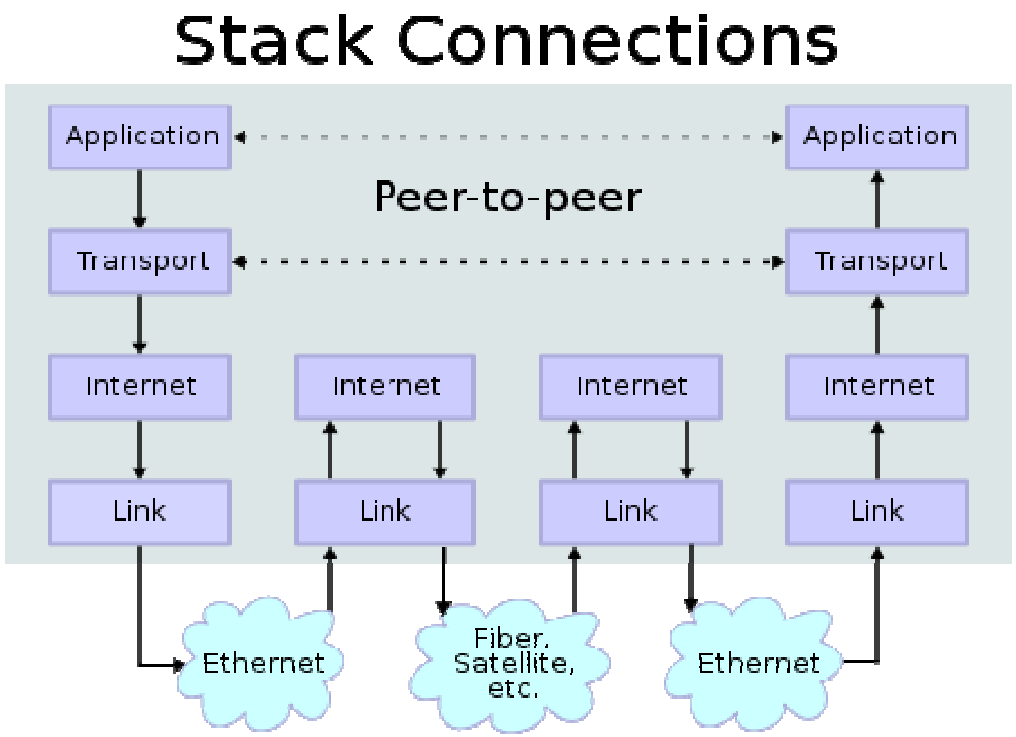
```
<o:Security s:mustUnderstand="1" xmlns:o="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
  <a:Timestamp a:Id="_0" xmlns:a="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <a:Created>2008-08-15T01:39:46.121Z</a:Created>
    <a:Expires>2008-08-15T01:44:46.121Z</a:Expires>
  </a:Timestamp>
  <o:BinarySecurityToken a:Id="_kt" EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
ValueType="http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ"
xmlns:a="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd">...</o:BinarySecurityToken>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
      <Reference URI="#_0">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>...</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>...</SignatureValue>
    <KeyInfo>
      <o:SecurityTokenReference a:TokenType="http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ" xmlns:a="h
ttp://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd">
        <o:Reference URI="#_kt" ValueType="http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ"/>
      </o:SecurityTokenReference>
    </KeyInfo>
  </Signature>
</o:Security>
```

# XML Encryption
# Theory recall

- **Secure channels in cryptography (wikipedia)**
  - **A confidential channel** is a way of transferring data that is resistant to interception, but not necessarily resistant to tampering.
  - **An authentic channel** is a way of transferring data that is resistant to tampering but not necessarily resistant to interception.
  - **A secure channel** is a way of transferring data that is resistant to interception and tampering.

# Transport layer security




- examples: SSL, TLS
- operates between transport (TCP) and application layers
- provides **point-to-point authentication**, **confidentiality**, **integrity** (for transport layer connection)

# Transport layer security

- Problem: it doesn't provide security for application layer communication (when decapsulated after transport):
  - if the recipient forwards/routes application messages, it has to be fully trusted or communication is unsecure
  - both sides have full insight in the content
- To allow routing of messages and selective security **end-to-end** security is required

# XML Encryption Syntax and Processing


- http://www.w3.org/TR/xmlenc-core/
- W3C Recommendation 10 December 2002
- Ground for WS-I Basic Security Profile

- Mini-agenda (next slides):
  - Encryption Granularity
  - Syntax
  - Security Considerations

# XML Encryption (xmlenc-core)
# Granularity: element

```
<?xml version='1.0'?>
<PaymentInfo>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000'
    Currency='USD'>
    <Number>4019 2445 0277
     5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Element <CreditCard> encrypted →

```
<?xml version='1.0'?>
<PaymentInfo>
  <Name>John Smith</Name>
  <EncryptedData
    Type='http://www.w3.org/2001/04/xml
    enc#Element'
    xmlns='http://www.w3.org/2001/04/x
    mlenc#'>
    <CipherData>
     <CipherValue>A23B45C56
     </CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

# XML Encryption (xmlenc-core) Granularity: element content (elements)

```
<?xml version='1.0'?>
<PaymentInfo>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000'
    Currency='USD'>
    <Number>4019 2445 0277
     5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

<CreditCard> content encrypted →

```
<?xml version='1.0'?>
<PaymentInfo
    xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <EncryptedData
     xmlns='http://www.w3.org/2001/04/xmlenc
     #'
     Type='http://www.w3.org/2001/04/xmlenc#
     Content'>
      <CipherData>
        <CipherValue>A23B45C56
      </CipherValue>
      </CipherData>
    </EncryptedData>
  </CreditCard>
</PaymentInfo>
```

# XML Encryption (xmlenc-core) Granularity: element content (character data)

```
<?xml version='1.0'?>
<PaymentInfo>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000'
    Currency='USD'>
    <Number>4019 2445 0277
    5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

<Number> CDATA encrypted →

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>
      <EncryptedData
      xmlns='http://www.w3.org/2001/04/xmlenc#'
      Type='http://www.w3.org/2001/04/xmlenc#Conte
      nt'>
        <CipherData>
          <CipherValue>A23B45C56</CipherValue>
        </CipherData>
      </EncryptedData>
    </Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

# XML Encryption (xmlenc-core) <EncryptedData> syntax


```
<EncryptedData Id? Type? MimeType?
  Encoding?>
 <EncryptionMethod/>?
 <ds:KeyInfo>
  <EncryptedKey>?
  <AgreementMethod>?
  <ds:KeyName>?
  <ds:RetrievalMethod>?
  <ds:*>?
 </ds:KeyInfo>?
 <CipherData>
  <CipherValue>?
  <CipherReference URI?>?
 </CipherData>
 <EncryptionProperties>?
</EncryptedData>
```

```
<CipherReference
  URI="http://www.example.com/CipherValues.xml
  ">
 <Transforms>
  <ds:Transform
  Algorithm="http://www.w3.org/TR/1999/REC-xpath-
  19991116">
    <ds:XPath
  xmlns:rep="http://www.example.org/repository">
self::text()[parent::rep:CipherValue[@Id="example1"
  ]]
    </ds:XPath>
  </ds:Transform>
  <ds:Transform Algorithm=
  "http://www.w3.org/2000/09/xmldsig#base64"/>
 </Transforms>
</CipherReference>
```

# xmlenc-core
# Security Considerations (1)

- Interaction of encryption with signatures
  - Signature computed over **encrypted or unencrypted form** of elements?
  - Clear-text digital signatures may allow **plaintext guessing attacks** – especially for XML
  - Recommended **encrypting** all signatures and digests
  - Recommended using **nonces** or **initialization vectors**
  - For messages with encrypted envelope: signatures secure plaintext which is signed, but not other unsigned information, even if it is encrypted (everyone can use public key to encrypt data)

# xmlenc-core
# Security Considerations (2)


- Information Revealed
  - When shared symmetric key is used, it should *only* be used for data intended for *all* recipients
  - Be careful about parameters (e.g. URIs) or algorithm identifiers
- Nonce and IV (Initialization Value or Vector)
  - Many encryption algorithms/modes result with the same ciphertext for the same plaintext
  - Prepending a random or/and secret unique value can help for Cipher Block Chaining (CBC) modes

# xmlenc-core
# Security Considerations (3)

- Denial of Service attack scenarios
  - Recursive processing (allowed)
  - EncryptedKey A requires EncryptedKey B to be decrypted, which itself requires EncryptedKey A
  - EncryptedData referencing network resources (very large or continually redirected)
- Unsafe Content
  - Obscured content that applications (firewalls, virus detectors) consider unsafe (executable code, viruses)
  - Can be disallowed
  - Or inspected after decryption
  - Or ensured that receiving app. can process data safely

# WSS – NetBeans IDE

- DEMO in NetBeans and SoapUI
  - New → Sample Web Service (Calculator)
  - Discard generated client application
  - Check source of the Web Service
  - Deploy, check WSDL, generate Tester
  - Enable Message Authentication over SSL
  - SSL: https and port 8181
  - Login: wsitUser, changeit (configured in Glassfish)
  - Enable WS-Addressing message-id

# There is a lot more

- WS-SecurityPolicy language (capabilities and requirements of security mechanisms as policies)
- Designing adequately secure SOA environments is tricky (performance, compatibility, ...)
- There are more implementations besides the JAX-WS reference impl. (Metro)
  - Axis2, CXF for Java
  - Apache Rampart/C – security module for Axis2/C
  - Web Services Framework for PHP
- Every year it gets more complex
  - But there is more support for REST-ful Web Services

# Resources

- Web Service Security Patterns
  - http://msdn.microsoft.com/en-us/library/aa480545.aspx
- Basic Security Profile Version 1.1
  - http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html
- OASIS Web Services Security (WSS) TC
  - http://www.oasis-open.org/committees/wss