



# Web Services Security

## “How does it really work...”

---

Denise Hatzidakis

Perficient, Inc.

Director, Chief Technologist

[denise.hatzidakis@perficient.com](mailto:denise.hatzidakis@perficient.com)

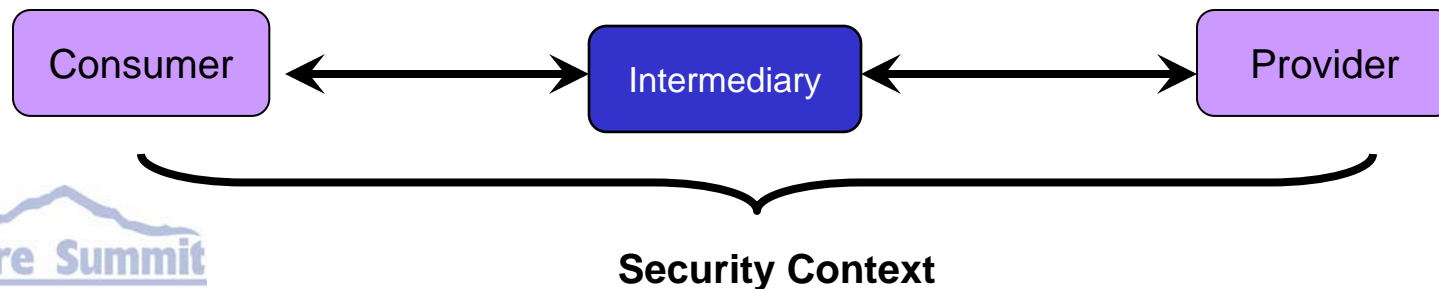


# Web Services and Security

SSL/TLS offers several security features including authentication, data integrity and data confidentiality but only for individual hops.



What is needed in a comprehensive Web service security architecture is a mechanism that provides end-to-end security and greater functionality





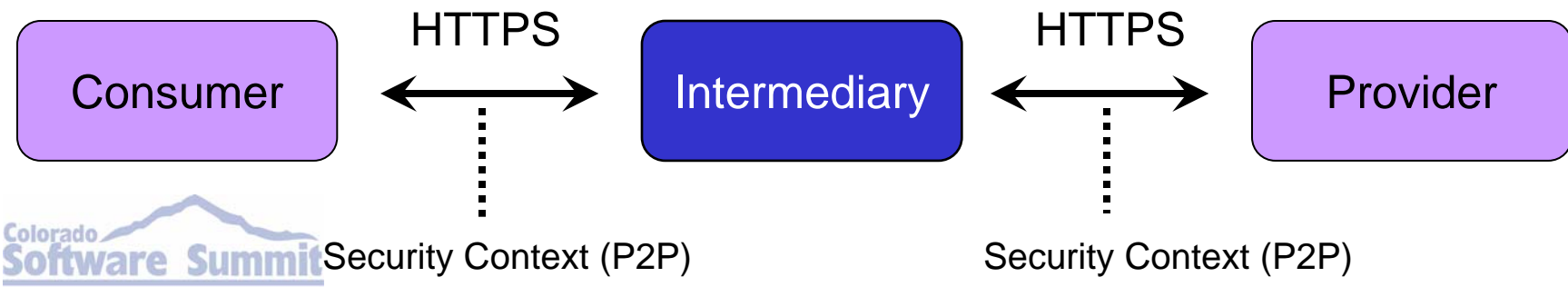
# Message Security Characteristics

---

- Security information is contained in and travels with each SOAP message.
  - Security information becomes transport-independent.
  - Identity/Authentication
  - Integrity
  - Confidentiality
  - Non-Repudiation
  - Based on:
    - XML digital signature to provide integrity
    - XML encryption to provide confidentiality
    - Security tokens to provide identity/authentication

# Transport Level Security (TLS) with Secure Sockets Layer (SSL)

- HTTP is the most commonly used internet protocol for Web services and one of the most insecure.
  - Messages are plain text.
  - Communicating peers are unauthenticated.
  - Messages are sent over an unsecured network.
- Secure Sockets Layer (SSL) secures HTTP at the transport-level:
  - Introduced in 1994 and supported by all modern browsers. Also sometimes referred to as TLS (Transport Level Security).
  - Uses Public Key Cryptography to authenticate client and/or server using digital certificates and to encrypt communications between them.
  - Represented by the prefix **https://** in URIs for Web service ports.



# WS-Security – Why?

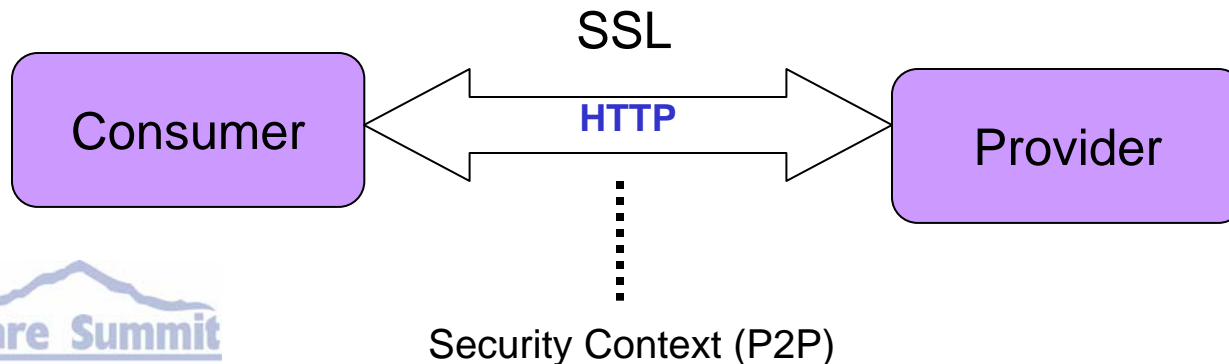
- Why do we need message level security?
  - With HTTP
    - We can authenticate the caller, sign the message, and encrypt the contents of the message thus the message is secure in several dimensions
      - ✓ the caller is known,
      - ✓ the receiver of the message can verify that the message did not change in transit, and
      - ✓ entities watching the wire traffic cannot figure out what data is being exchanged
  - For those looking at SOAP messaging to solve bigger problems, however, HTTP-based security simply isn't enough.
    - ✓ Bigger problems involve sending the message along a path **more complicated than request/response** or over a transport that **does not involve HTTP**.
    - ✓ The identity, integrity, and security of the message and the caller need to be **preserved over multiple hops**
    - ✓ More than one encryption key may be used along the route.
    - ✓ Trust domains will be crossed

## Bottom Line:

HTTP and its security mechanisms only address point-to-point security. More complex solutions need end-to-end security baked in. WS-Security addresses how to maintain a secure context over a multi-point message path.

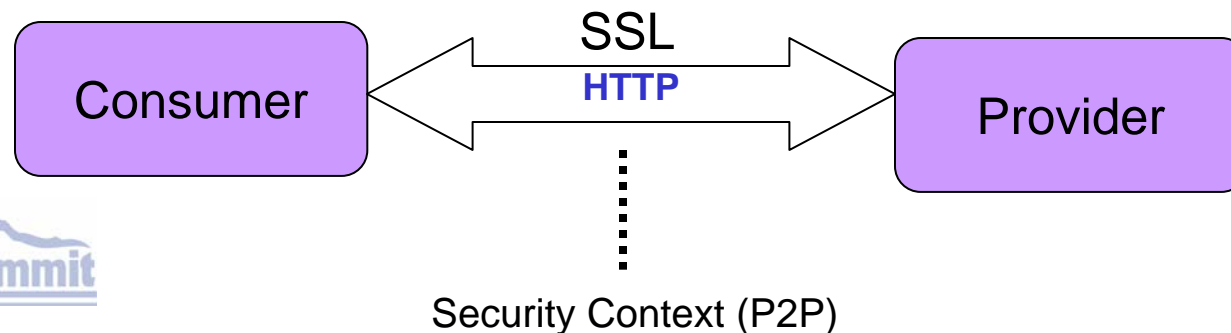
# Benefits of using SSL

- Benefits
  - SSL provides connection security through:
    - Communication privacy:
      - ✓ The data on the connection is encrypted.
    - Communication integrity.
      - ✓ The protocol includes a built-in integrity check.
    - Authentication.
      - ✓ The client knows who the server is (and potentially vice-versa).
- Fast
- SSL creates a secure channel, securing the data using a combination of symmetric and asymmetric encryption.



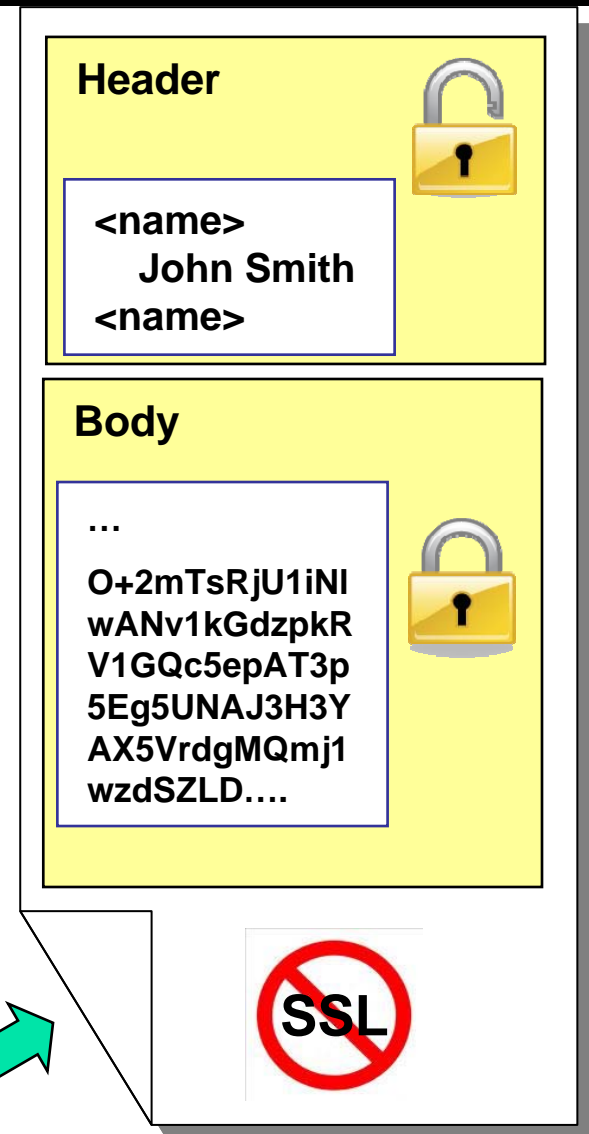
# Limitations of using SSL

- **The whole message is encrypted**
  - inefficient if you don't need whole message encrypted
  - cannot use content based routing without terminating the SSL connection and decrypting the whole message
  - cannot encrypt different parts using different keys for different "actors" in the message processing path
- **Does not provide non-repudiation**
  - For non-repudiation information has to be stored "signed" using asymmetric key cryptography.
  - SSL doesn't provide access to the signed packets, and it uses symmetric key cryptography for the data packets
- **Connection oriented (single hop) not message oriented (end to end)**
  - SSL connection is terminated at intermediaries and a new SSL connection must be established for each hop, thus the identity of the original client is lost on the second hop
  - Intermediaries get to see unencrypted message (may want some info encrypted from end to end)
  - Intermediaries get to modify the message undetected (no guaranteed integrity)



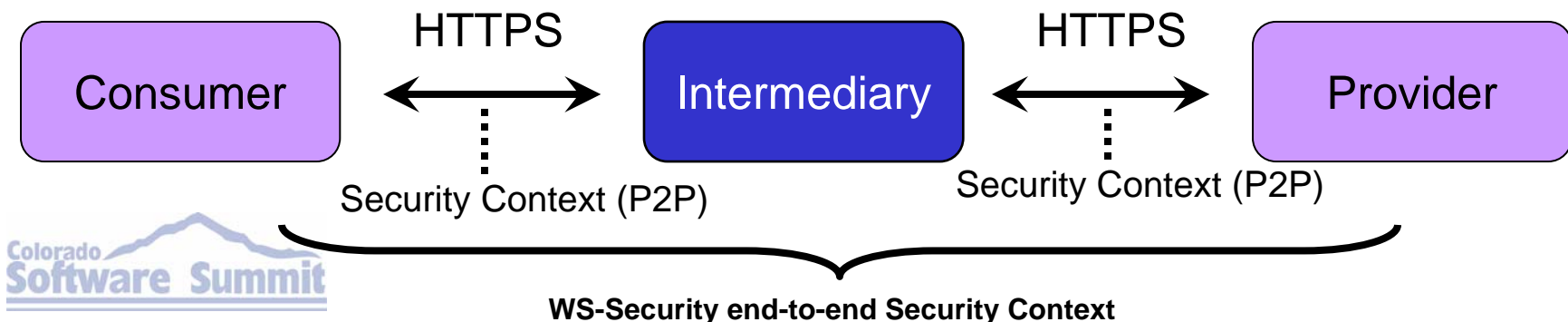
# Transport-level security and Web services

- Use transport level security with Web services when:
  - No intermediaries are used in the Web service environment.
    - For example, an application-layer network firewall.
  - The transport protocol is only ever HTTP.
    - For example, it can not be JMS.
  - The Web services client is unmanaged.
    - Managed clients run within a J2EE container.
    - WS-Security can only be used with managed clients.
  - With SSL only the whole message can be encrypted.
    - There is **no** ability to encrypt only parts of a message.
    - As shown here, if you need to use or modify information in the header of the message for audit logging it must be visible as plaintext.



# Implementing message level security with WS-Security

- The WS-Security specification (discussed next) provides message-level security, which is used when building secure Web services.
- Some of the advantages of using WS-Security over SSL are:
  - It can provide end-to-end message-level security.
    - This means that message security can be protected even if the message goes through multiple services (intermediaries).
  - It is independent of the transport layer protocol.
    - It can be used for any Web service binding, for example, HTTP, SMTP, FTP, or RMI.
- Provides a flexible set of mechanisms for using a range of security protocols.
  - For example, you can associate security-related claims with a message.
  - It does **not** define a set of security protocols.





# The WS-Security specification

---

- The WS-Security specification, Web Services Security:
  - SOAP Message Security 1.0 – released April 2004
  - WSS 1.1 – released by Oasis February 2006.
  
- Defines a framework for placing security information in the SOAP Headers.
  
- The specification is flexible and designed to be used as the basis for securing Web services using a wide variety of security models, including PKI, Kerberos, and SSL
  - Provides support for:
    - multiple security token formats
    - multiple trust domains
    - multiple signature formats
    - multiple encryption technologies based on XML signature and XML encryption to provide message integrity and confidentiality.
    - security token propagation.
  
- Does not address all aspects of Web services security.
  - Represents one layer in a complete, layered security solution for Web services.





# WS-Security Overview

---

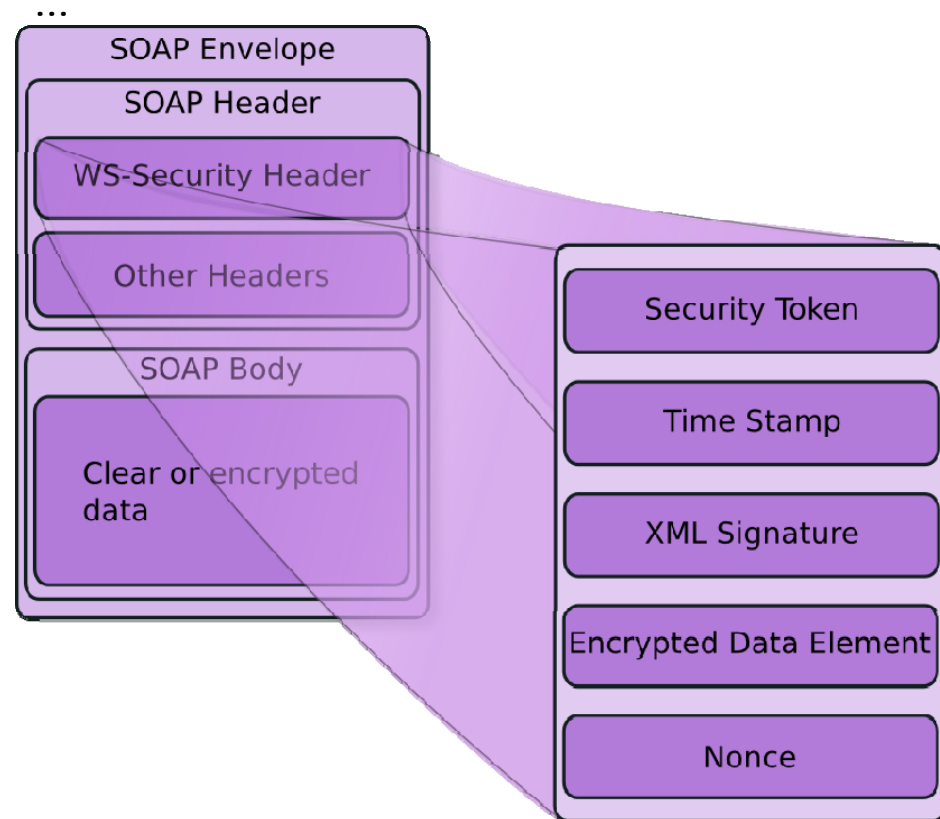
- A set of mechanisms to help developers of Web Services secure SOAP message exchanges.
  - These basic mechanisms can be combined in various ways to accommodate building a wide variety of security models using a variety of cryptographic technologies.
  
- A general-purpose mechanism for associating security tokens with messages.
  - However, no specific type of security token is required by WS-Security.
    - Designed to be extensible (*e.g.* support multiple security token formats) to accommodate a variety of authentication and authorization mechanisms.
    - How to include opaque encrypted keys as a sample of different binary token types
  
- WS-Security describes how to encode binary security tokens and attach them to SOAP messages.
  - Specifically, the WS-Security profile specifications describe how to encode
    - Username Tokens,
    - X.509 Tokens,
    - SAML Tokens,
    - REL Tokens and
    - Kerberos Tokens
  
- WS-Security also includes extensibility mechanisms that can be used to further describe the credentials that are included with a message.

# WS-Security Specification

- WS-Security describes enhancements to SOAP messaging to provide quality of protection through
  - message integrity,
  - message confidentiality, and
  - single message authentication.
  
- Describes a way to place a security token in a SOAP Header
  - Tokens can be carried in the message (“pushed”) or
  - Tokens can be referenced in the message and subsequently the receiver can “pull” the token from a token service
  
- Provides a means to protect a message by encrypting and/or digitally signing a body, a header, an attachment, or any combination of them (or parts of them)
  - **Message integrity**
    - leverages XML Signature and security tokens
    - ensures that messages
      - ✓ have originated from the appropriate sender
      - ✓ were not modified in transit.
  - **Message confidentiality**
    - leverages XML Encryption and security tokens to
      - ✓ keeps portions of a SOAP message confidential security tokens to keep portions of a SOAP message confidential.
  
- WS-Security **does not** specify the format of the signature or encryption.
  - It specifies how one would embed the security information laid out by other specifications within a SOAP message.

# Inside a WS-Security Header

```
<SOAP:Envelope xmlns:SOAP="...">  
  <SOAP:Header>  
    <wsse:Security SOAP:role="..." SOAP:mustUnderstand="...">  
      ...  
    </wsse:Security>  
  </SOAP:Header>  
<SOAP:Body Id="MsgBody">  
  <!-- SOAP Body data -->  
</SOAP:Body>  
</SOAP:Envelope>
```





# WS-Security Namespaces

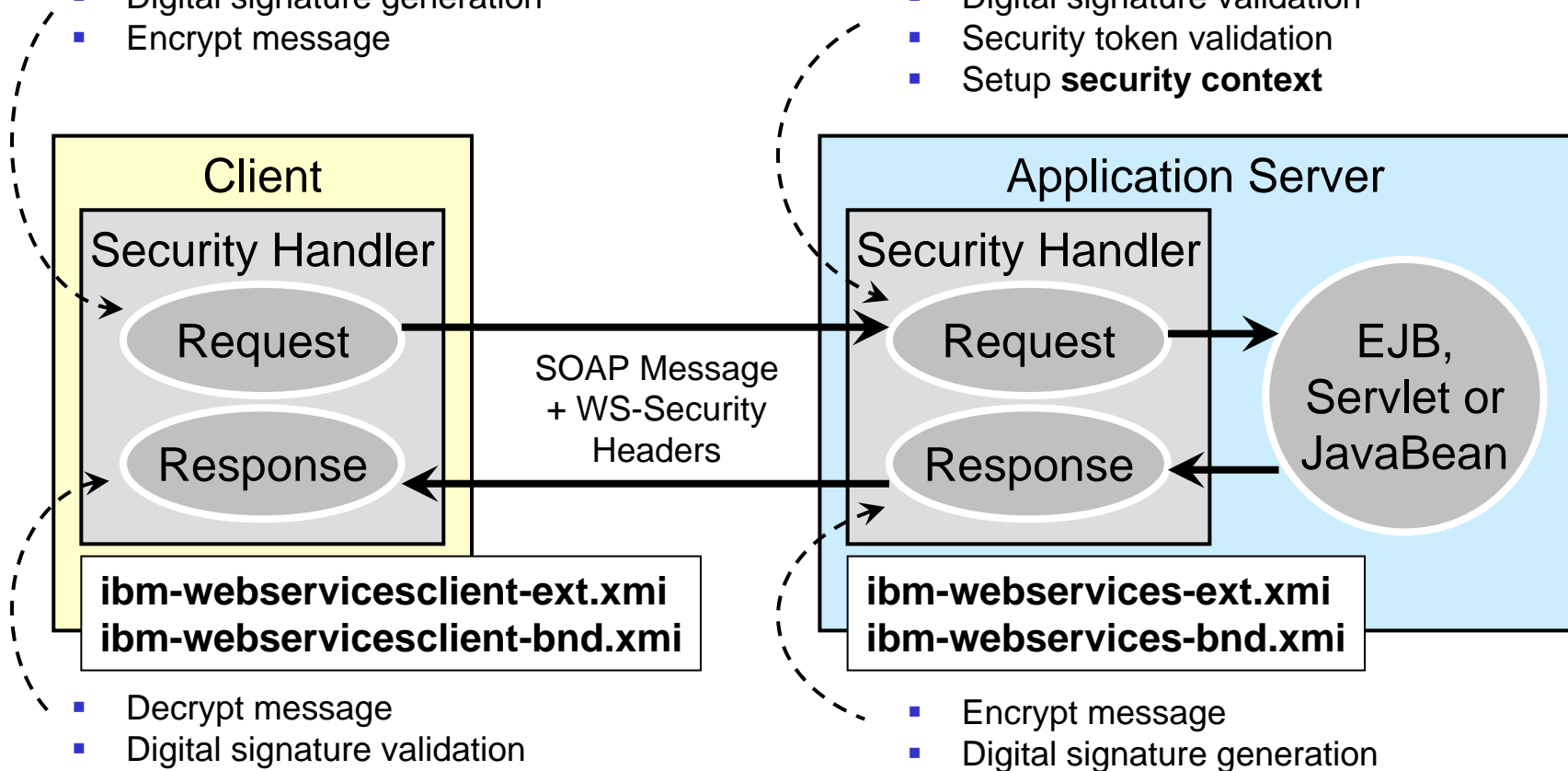
---

| Prefix | NameSpace   | Some Related Elements  |
|--------|---|--|
| wsse   | <a href="http://schemas.xmlsoap.org/ws/2002/07/secext">http://schemas.xmlsoap.org/ws/2002/07/secext</a>   | <Security>, <UsernameToken>, <Username>, <Password>, <Nonce> |
| wsu    | <a href="http://schemas.xmlsoap.org/ws/2002/07/utility">http://schemas.xmlsoap.org/ws/2002/07/utility</a> | <Timestamp>, <Created>, <Expires>                            |
| ds     | <a href="http://www.w3.org/2000/09/xml#dsig">http://www.w3.org/2000/09/xml#dsig</a>                       | <Signature>, <SignedInfo>, <SignatureValue>, <KeyInfo>       |
| xenc   | <a href="http://www.w3.org/2001/04/xml#enc">http://www.w3.org/2001/04/xml#enc</a>                         | <EncryptedData>, <CipherData>                                |

# WS-Security implementation

- Security token generation
- Digital signature generation
- Encrypt message

- Decrypt message
- Digital signature validation
- Security token validation
- Setup **security context**



- Decrypt message
- Digital signature validation

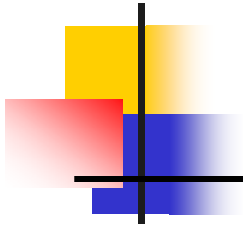
- Encrypt message
- Digital signature generation

# Message Security Characteristics and J2EE

---

## ■ Authorization

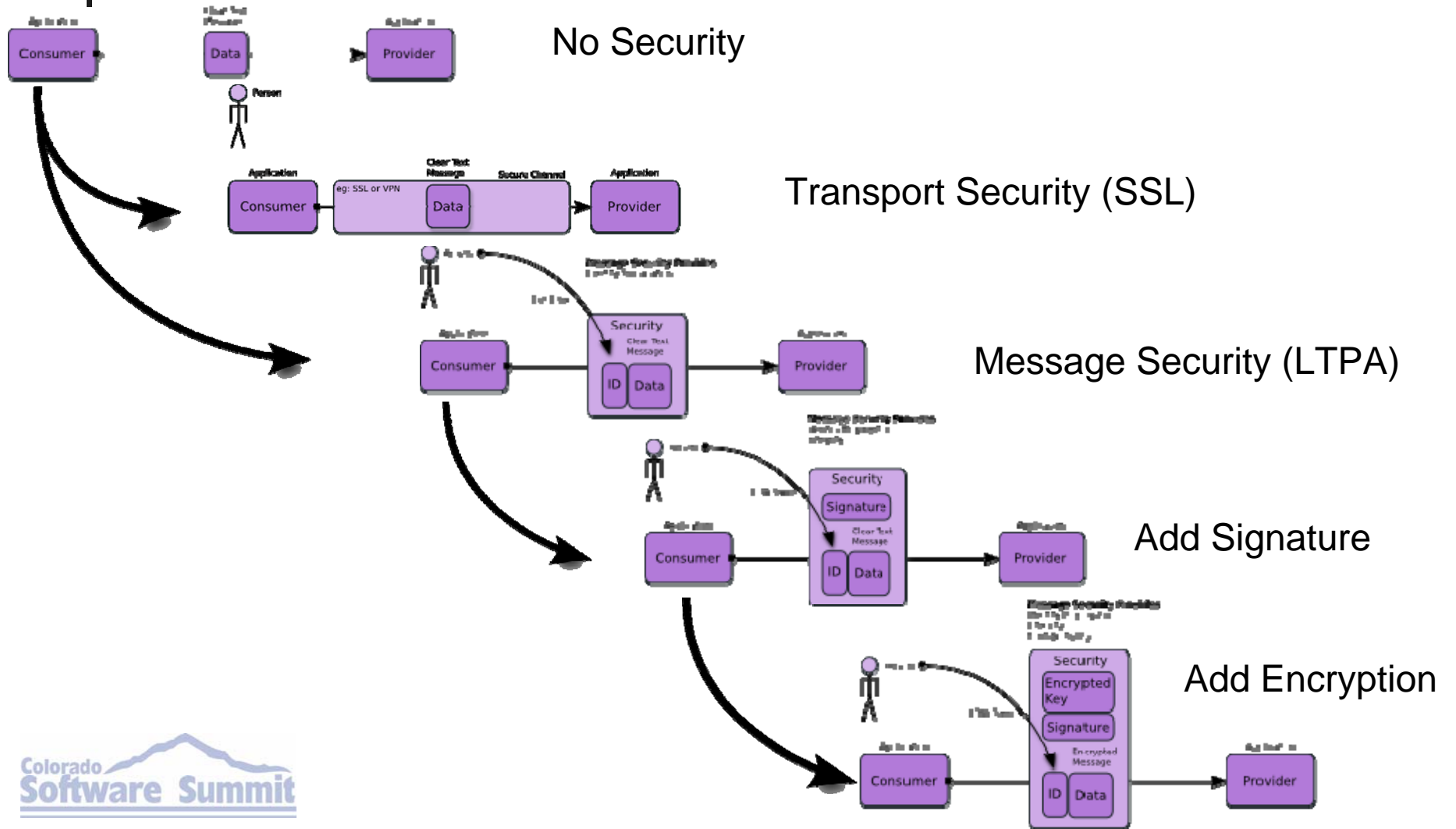
- Normal J2EE authorization
- Web Service is implemented as Servlet Java Bean
  - Authorization is done by the Web Container
  - Coarse grain protected URL
  - Using the identity from HTTP(s) transport, not identity from WSS
- Web Service is implemented as stateless EJB
  - Authorization could be done by the EJB Container
  - J2EE method level protection
  - Could use identity from WSS or transport
- Custom authorization done by J2EE container through JAAS interface



# Examples



# Example Scenario Overview





# WS-Security

---

- WS-Security defines SOAP extensions to implement
  - client authentication,
  - message integrity and
  - message confidentiality on the message level.



# WS Security Authentication

---

- **Authentication solves questions as**

- "Who is the caller?"
- "How does he prove his identity?"

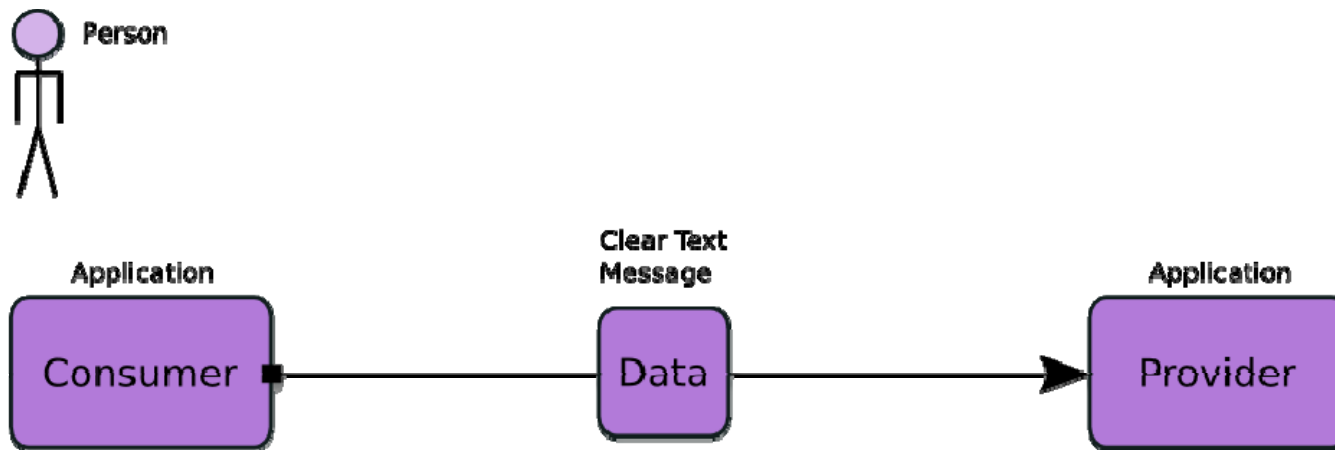
If these questions can be answered, it's the recipient's task to clarify the caller is to be trusted.

- **Authentication specifically prevents:**

- Masquerade attacks:
  - Users must prove their identity, so it is more difficult to masquerade as another.
- Replay attacks:
  - When using timestamps, it is difficult to reuse stolen authentication information.
- Identity interception:
  - When exchanges are additionally encrypted, intercepted identities are useless.

Note that authentication is only half the part of the security task. Once you know who the user is, you have to determine which resources the user is allowed to access. That's what authorization is for.

# Example: No Security





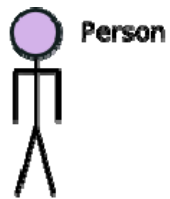
# Example: No Security

---

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <soapenv:Header>
    <wsa:To>
      http://localhost:9081/WeatherServiceProvider/services/WeatherJavaBean
    </wsa:To>
    <wsa:Action>getTemperatures</wsa:Action>
    <wsa:MessageID>
      uuid:205CE752-0112-4000-E000-03C4A9FE52B8
    </wsa:MessageID>
  </soapenv:Header>
  <soapenv:Body>
    <p821:getTemperatures xmlns:p821="http://bean.itso">
      <startDate>2007-04-23T04:00:00.000Z</startDate>
      <days>3</days>
    </p821:getTemperatures>
  </soapenv:Body>
</soapenv:Envelope>
```



# Example: Transport Security (SSL)



# Example: Transport Security (SSL)

---

- More difficult to capture
- Designed to prevent “man in the middle” attacks
- If we did capture it,
  - It would be after decrypting it, so it would look the same as the previous example

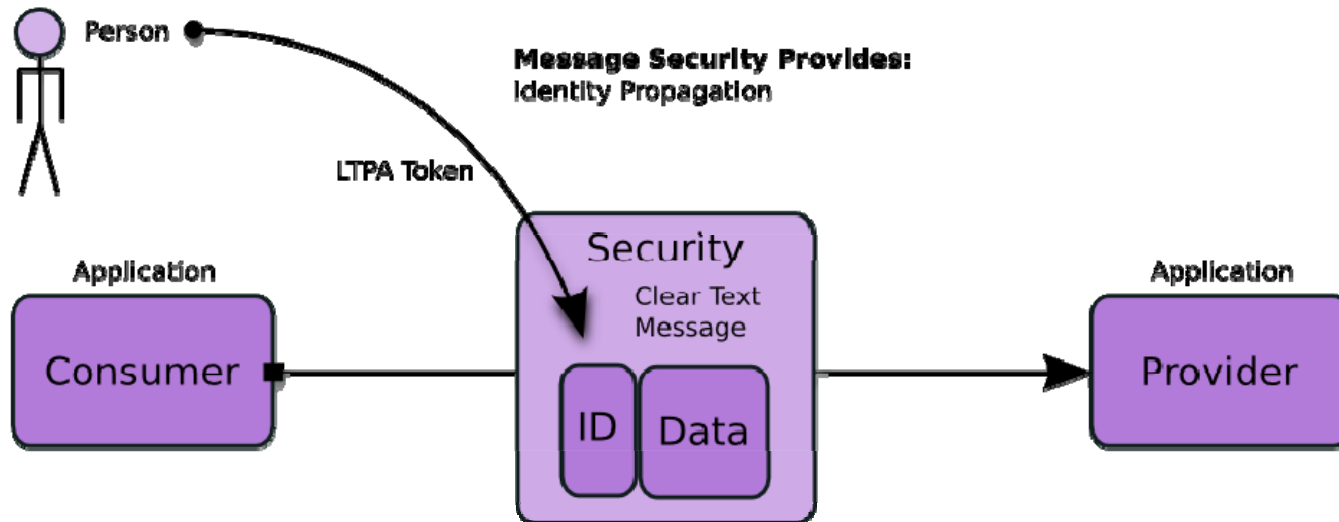


# Transport Security Characteristics

---

- Authentication
  - HTTP Basic authentication
  - Client side certificates
  - Authenticates peer, but not data origin
- Integrity
- Confidentiality
- Fast

# Example: Message Security (Security Token)





# Security Tokens

---

- Most service providers see the importance to know who is talking to them, and whether to allow the message sender access to their services.
  
- WS-Security allows us to use any security token we like to use.
  - Explicitly defined are three different options:
    - username/password authentication
      - ✓ in case of custom authentication
    - binary authentication
      - ✓ tokens in the form of Kerberos tickets or X.509 certificates.
    - custom binary security tokens can be applied

# Security Tokens – UserName /Password

---

- The first option is to rely on custom authentication using username and password validation only.
- WSS defines an element called <UsernameToken> which provides support of this purpose.
  - <UsernameToken> among other things comprise the following sub-elements:
    - /Username
      - username associated with this token
    - /Password
      - password for the username associated with this token
    - /Password/@Type
      - type of the password provided; 2 pre-defined types:
        - ✓ PasswordText
          - clear text password
        - ✓ PasswordDigest
          - digest of the password, base64-encoded SHA1 hash value of the UTF8-encoded password
    - /Nonce
      - ✓ nonce for the token
    - /Created
      - ✓ date and time of token creation



## Security Tokens – UserName /Password Scenarios

---

The <username> token's usage is dependent on the way the <password> token is used.

- Simplest – Username only

```
<UsernameToken>  
  <Username>MyName</Username>  
</UsernameToken>
```

- Note

- Alone would be quite insecure.
- Without any prove of identity, it's only useful in situations where some other authentication mechanism like SSL is used, with the user name only being a basic identification means.

# Security Tokens – UserName /Password Scenarios

---

- Conveying the `<password>` element as a part of the `<UsernameToken>` element,
  - The first approach to (admittedly weak) secure authentication would be done, since an identity could now be proven.

```
<UsernameToken>  
  <Username>MyName</Username>  
  <Password Type="PasswordText">MyPass  
  </Password>  
</UsernameToken>
```

- The Type attribute indicates that the password is given as clear text.
  - Thus, if someone intercepts the message he can easily figure out the password and authenticate himself.



## Security Tokens – UserName /Password Scenarios

---

- To prevent this, the password should be sent as a hashed value, making it impossible for an intermediary to see the real password.
  - The hashing is done following the [SHA-1](#) algorithm, then transmitting the hashed password base64-encoded.

```
<UsernameToken>  
  <Username>MyName</Username>  
  <Password Type="PasswordDigest">  
    fm6SuM0RpI|hBQFgmESjdim/yj0=  
  </Password>  
</UsernameToken>
```



# WS-Security

---

- <!-- Revised UsernameToken -->

```

<wsse:UsernameToken
  xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
  wsu:Id="SecurityToken-58564463-5bdc-4a6b-a7fb-94a0d7357a20">
  <wsse:Username>Joe</wsse:Username>
  <wsse:Password Type="wsse:PasswordDigest">
    gpBDXjx79eutcXdtIUlIcrSiRs=
  </wsse:Password>
  <wsse:Nonce>
    h52sl9pKV0BVRPUoIQc7Cg==
  </wsse:Nonce>
  <wsu:Created>
    2002-11-04T19:16:50Z
  </wsu:Created>
</wsse:UsernameToken>

```

Password\_Digest =

Base64(SHA-1(Nonce + Created + Password))

# Example – WS-Security authentication

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
        wsswssecurity-secext-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>
          mickey
        </wsse:Username>
        <wsse:Password Type=
          "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wssusername-
            token-profile-1.0#PasswordText">
          password
        </wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <q0:getBookInventory xmlns:q0=http://north.library.ibm.com/>
  </soapenv:Body>
</soapenv:Envelope>

```





# Deployment descriptor snippet

---

```
<securityRequestGeneratorBindingConfig xmi:id="SecurityRequestGeneratorBindingConfig_1176825888703">
  <tokenGenerator xmi:id="TokenGenerator_1177433308202" name="weatherLTPA"
    classname="com.ibm.wsspi.wssecurity.token.LTPATokenGenerator">
    <valueType xmi:id="ValueType_1177433308202" localName="LTPA"
      uri="http://www.ibm.com/websphere/appserver/tokentype/5.0.2"
      name="LTPA Token"/>
    <callbackHandler xmi:id="CallbackHandler_1177433308202"
      classname="com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler">
      <basicAuth xmi:id="BasicAuth_1177433308202"/>
    </callbackHandler>
    <partReference xmi:id="PartReference_1177433308202" part="weatherLTPA"/>
  </tokenGenerator>
</securityRequestGeneratorBindingConfig>
```

```
<securityRequestGeneratorServiceConfig
  xmi:id="SecurityRequestGeneratorServiceConfig_1176825580296">
  <securityToken xmi:id="SecurityToken_1177433308249" name="weatherLTPA"
    uri="http://www.ibm.com/websphere/appserver/tokentype/5.0.2"
    localName="LTPA"/>
</securityRequestGeneratorServiceConfig>
<securityResponseConsumerServiceConfig xmi:id="SecurityResponseConsumerServiceConfig_1177433308249"/>
```

# Example: Message Security (Security Token)

<soapenv:Envelope

```

xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
<soapenv:Header>

```

```

<wsse:Security soapenv:mustUnderstand="1"
  xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
  <wsse:BinarySecurityToken ValueType="wsst:LTPA"
    xmlns:wsst="http://www.ibm.com/websphere/appserver/tokentype/5.0.2">
    SZq0bum+DS5rBzF91Gc554JB9xmrdy9cQHPWhNwveqABVpjhZRbLJvSjFW/ApVb9nGaesbH9cp5mrG5h4pxtVyZq
  </wsse:BinarySecurityToken>
</wsse:Security>

```

```

<wsa:To>
  http://localhost:9081/WeatherServiceProvider/services/WeatherJavaBean
</wsa:To>
<wsa:Action>getTemperatures</wsa:Action>
<wsa:MessageID>
  uuid:20752EF5-0112-4000-E000-0CDCA9FE52B8
</wsa:MessageID>

```

```

</soapenv:Header>
<soapenv:Body>
  <p821:getTemperatures xmlns:p821="http://bean.itso">
    <startDate>2007-04-23T04:00:00.000Z</startDate>
    <days>4</days>
  </p821:getTemperatures>

```

</soapenv:Envelope>



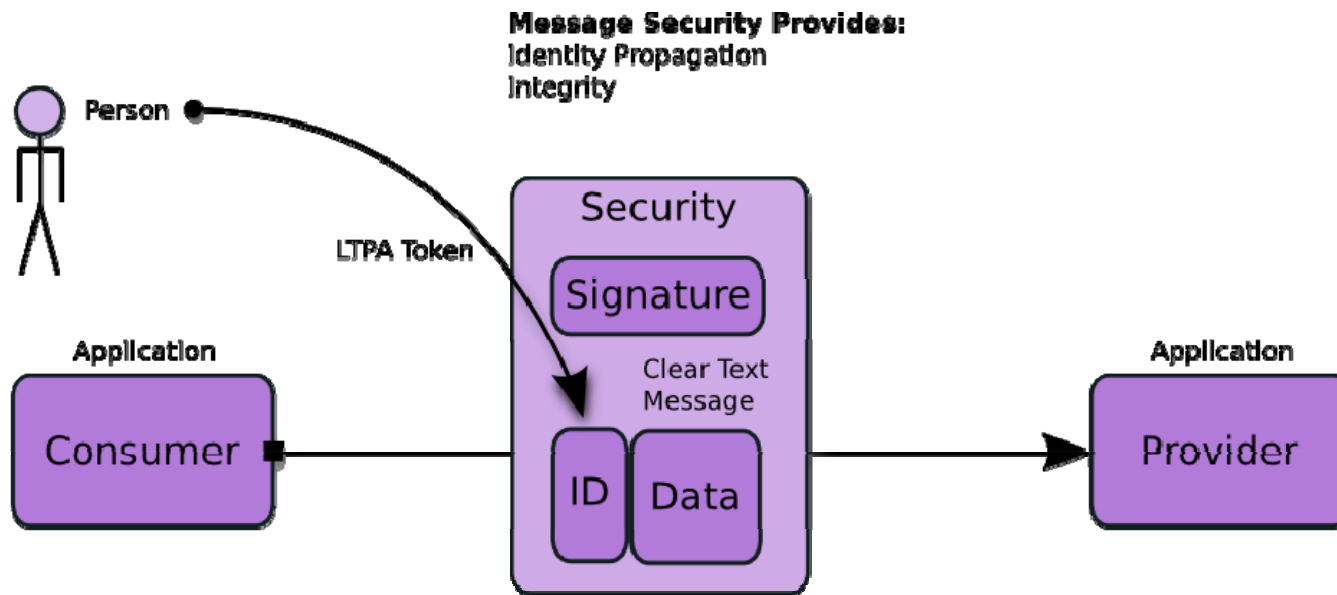
```

<wsse:BinarySecurityToken ValueType="wsst:LTPA"
  xmlns:wsst="http://www.ibm.com/websphere/appserver/tokentype/5.0.2">
  SZq0bum+DS5rBzF91Gc554JB9xmrdy9cQHPWhNwveqABVpjhZRbLJvSjFW
  /ApVb9nGaesbH9cp5mrG5h4pxtVyZq
  ...
</wsse:BinarySecurityToken>

```



# Example: Message Integrity





# WS-Security Integrity

---

- Message integrity ensures the recipient that the data he receives has not been altered during transit.
- WS-Security tries to ensure integrity using the XML Signature specification
  - Defines a methodology for cryptographically signing XML.
  - The signatures are defined using a <Signature> element and accompanying sub-elements as part of a security header.
- The signature itself is computed based on the SOAP message content and a security token.
- The message receiver can check the validity of the message using an according decoding algorithm.

# Example: Message Security (Add Signature)

```

- <soapenv:Envelope>
  - <soapenv:Header>
    - <wsse:Security soapenv:mustUnderstand="1">
      + <wsse:BinarySecurityToken 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/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509"
        wsu:Id="x509bst_2"></wsse:BinarySecurityToken>
      + <ds:Signature></ds:Signature>
      + <wsse:BinarySecurityToken ValueType="wsst:LTPA"></wsse:BinarySecurityToken>
    </wsse:Security>
  + <wsa:To></wsa:To>
  <wsa:Action>getTemperatures</wsa:Action>
  <wsa:MessageID> uuid:209E951A-0112-4000-E000-0BF8A9FE52B8 </wsa:MessageID>
</soapenv:Header>
- <soapenv:Body wsu:Id="wssecurity_signature_id_1">
  - <p821:getTemperatures>
    <startDate>2007-04-23T04:00:00.000Z</startDate>
    <days>6</days>
  </p821:getTemperatures>
</soapenv:Body>
</soapenv:Envelope>

```

```

<wsse:BinarySecurityToken ValueType="wsst:LTPA"
  xmlns:wsst="http://www.ibm.com/websphere/appserver/tokentype/5.0.2">
  SZq0bum+DS5rBzF91Gc554JB9xmr9cQHPWhNwveqABVpjhZRbLJvSjFW
  /ApVb9nGaesbH9cp5mrG5h4pxtVyZq
  ...
</wsse:BinarySecurityToken>

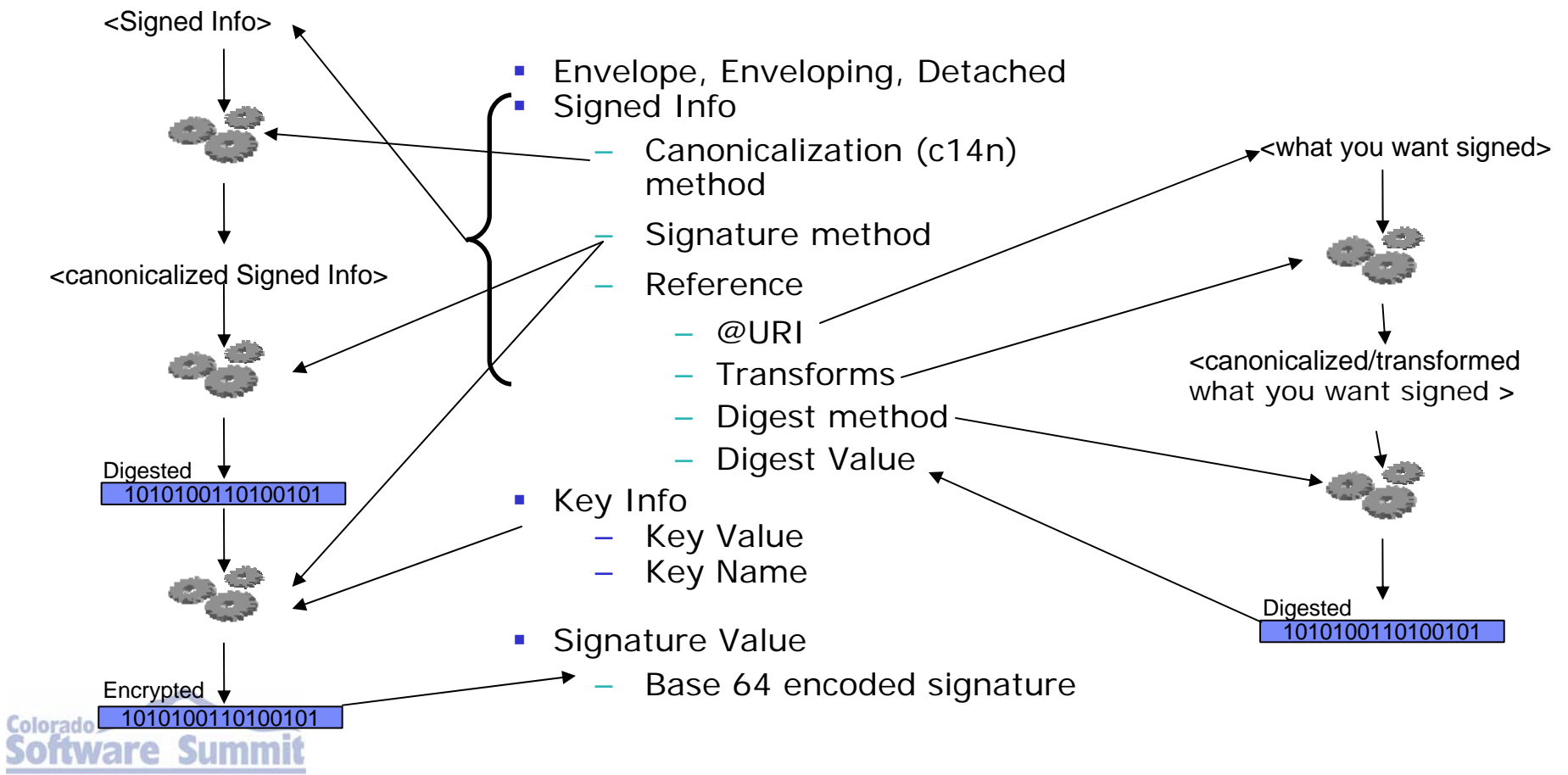
```



# Inside an XML Digital Signature

## The security token

## What you want signed



# Sample deployment descriptor snippets

```

<signingInfo xmi:id="SigningInfo_1176912461906" name="">
  <signatureMethod xmi:id="SignatureMethod_1177434173436"
    algorithm="http://www.w3.org/2000/09/xmlsig#rsa-sha1"/>
  <canonicalizationMethod xmi:id="CanonicalizationMethod_1177434173436"
    algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
  <partReference xmi:id="PartReference_1176912461906" part="Integrity_1854828174">
    <transform xmi:id="Transform_1176912461906"
      algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <digestMethod xmi:id="DigestMethod_1176912461906"
      algorithm="http://www.w3.org/2000/09/xmlsig#sha1"/>
  </partReference>
  <signingKeyInfo xmi:id="SigningKeyInfo_1177434173436"
    keyInfoRef="KeyInformation_1862692614" name=""/>
</signingInfo>

```

```

<integrity xmi:id="Integrity_1177434173452" name="Integrity_1854828174" order="1">
  <messageParts xmi:id="MessageParts_1177434173452"
    Dialect="http://www.ibm.com/websphere/webservices/wssecurity/dialect-was"
    keyword="body"/>
  <messageParts xmi:id="MessageParts_1177434173453"
    Dialect="http://www.ibm.com/websphere/webservices/wssecurity/dialect-was"
    keyword="securitytoken"/>
</integrity>

```

# Digital Signature

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

```
  <ds:SignedInfo>
```

```
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
```

```
      <ec:InclusiveNamespaces PrefixList="xsi xsd wsa soapenv soapenc wsse ds"
```

```
        xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" />
```

```
    </ds:CanonicalizationMethod>
```

```
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
```

```
    <ds:Reference URI="#wssecurity_signature_id_1">
```

```
      <ds:Transforms>
```

```
        <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
```

```
          <ec:InclusiveNamespaces PrefixList="xsi xsd wsa soapenv soapenc wsu p821"
```

```
            xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" />
```

```
          </ds:Transform>
```

```
        </ds:Transforms>
```

```
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
```

```
        <ds:DigestValue>fHGMxsl2VswZa0Zz8eX0pUieTrc=</ds:DigestValue>
```

```
      </ds:Reference>
```

```
    </ds:SignedInfo>
```

```
  <ds:SignatureValue>HeS15nuP2SVcNdVDRY471T1qNx63nu5a0etGp09+VW0Zrv6n1uO9zDj60apcm16sbZLlIm/HRtGzk1+y+31E6  
+W5D0btslzHpJ3F6dkF9MObz76e37+9X+y94q+zHFzmDqrUzqiCliJaiayW41nny+MsuZs0vjvRsbn4m/LeIM=
```

```
</ds:SignatureValue>
```

```
  <ds:KeyInfo>
```

```
    <wsse:SecurityTokenReference>
```

```
      <wsse:Reference URI="#x509bst_2" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-  
wss-x509-token-profile-1.0#X509" />
```

```
    </wsse:SecurityTokenReference>
```

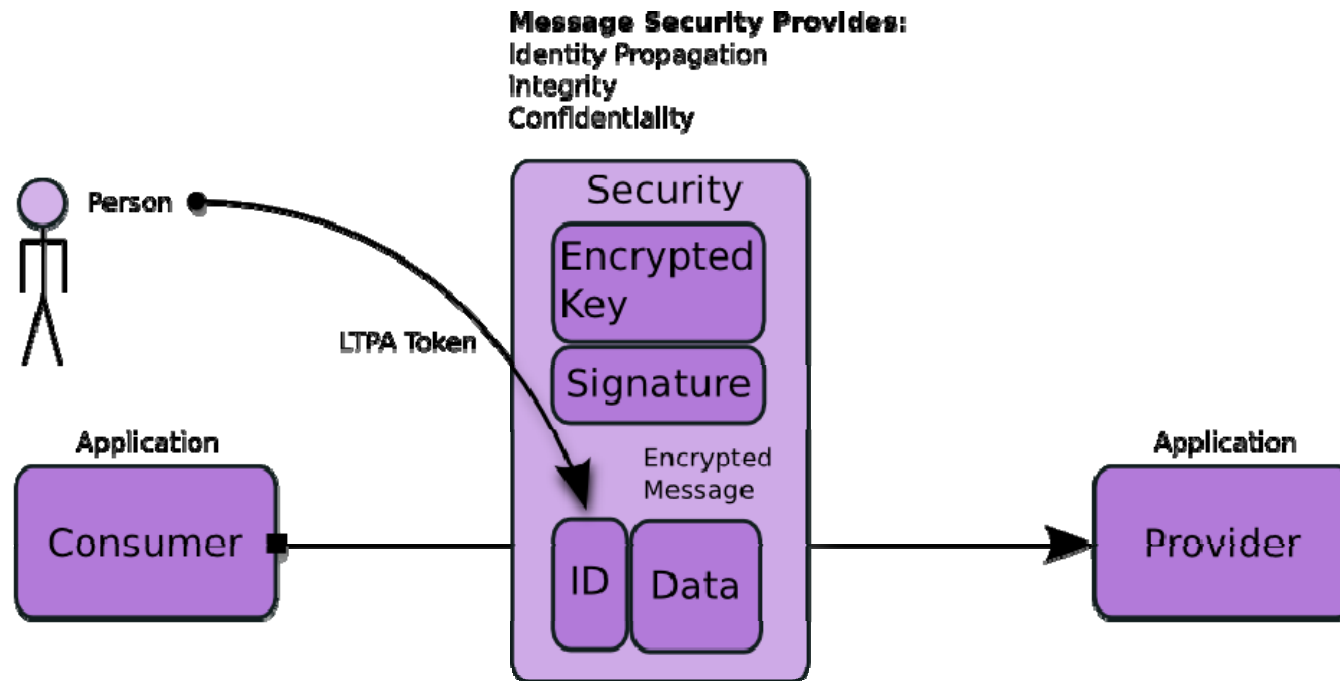
```
  </ds:KeyInfo>
```

```
</ds:Signature>
```

Signature

What we are signing

# Example: Message Confidentiality





# WS-Security Confidentiality

---

- Message confidentiality is to make the user sure that the data can't be read during transit, by means of message encryption.
- The XML Encryption specification is the basis to encrypt portions of the SOAP messages.
  - Any portions of SOAP messages, including headers, body blocks, and substructures, may be encrypted.
- The encryption is realized using either
  - symmetric keys shared by the sender and the receiver of the message or
  - a key carried in the message in an encrypted form.

# Inside XML Encryption

## Encrypted Key

Method (Algorithm)

Key Info

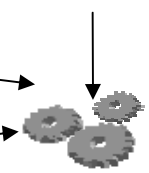
Token Reference

Cipher Data

Cipher Value

Reference List

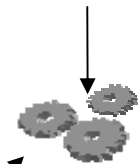
<Random symmetric Key>



Encrypted

1010100110100101

<what you want to encrypt>



Encrypted

1010100110100101

## Encrypted Data

Method (Algorithm)

Cipher Data

Cipher Value



# Deployment descriptor snippet

```
<securityRequestGeneratorServiceConfig xmi:id="SecurityRequestGeneratorServiceConfig_1176825580296">
  <integrity xmi:id="Integrity_1176912461937" name="Integrity_1854828174" order="1">
    ...
  </integrity>
  <confidentiality xmi:id="Confidentiality_1177434738905" name="Confidentiality_1253984958"
    order="2">
    <messageParts xmi:id="MessageParts_1177434738905"
      Dialect="http://www.ibm.com/websphere/webservices/wssecurity/dialect-was"
      keyword="bodycontent"/>
    </confidentiality>
    <securityToken xmi:id="SecurityToken_1176858688812" name="weatherLTPA"
      uri="http://www.ibm.com/websphere/appserver/token/5.0.2" localName="LTPA"/>
  </securityRequestGeneratorServiceConfig>

<encryptionInfo xmi:id="EncryptionInfo_1177370076046" name="">
  <encryptionMethod xmi:id="DataEncryptionMethod_1177370076046"
    algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc"/>
  <keyEncryptionMethod xmi:id="KeyEncryptionMethod_1177370076046"
    algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
  <encryptionKeyInfo xmi:id="EncryptionKeyInfo_1177370076046"
    keyInfoRef="KeyInformation_1268927394" name=""/>
  <partReference xmi:id="PartReference_1177370076046" part="Confidentiality_1253984958"/>
</encryptionInfo>
```

# Example: Message Security (Add Encryption)

```

- <soapenv:Envelope>
  - <soapenv:Header>
    - <wsse:Security soapenv:mustUnderstand="1">
      + <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary" ValueType="Binary" wsu:Id="x509bst_4"></wsse:BinarySecurityToken>
      + <EncryptedKey></EncryptedKey>
      + <ds:Signature></ds:Signature>
      + <wsse:BinarySecurityToken ValueType="wsst:LTPA"></wsse:BinarySecurityToken>
    </wsse:Security>
  + <wsa:To></wsa:To>
  <wsa:Action>getTemperatures</wsa:Action>
  <wsa:MessageID> uuid:20C73FCF-0112-4000-E000-0E64A9FE52B8 </wsa:MessageID>
</soapenv:Header>
- <soapenv:Body wsu:Id="wssecurity_signature_id_3">
  - <EncryptedData Id="wssecurity_encryption_id_5" Type="http://www.w3.org/2001/04/xmlenc#Content">
    <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
    - <CipherData>
      - <CipherValue>
        aicDZqQIIZv7E4IWozoo4rMAZYrUD3xsHDyBdS2ioK6EdXwioY6QMGMfO/c8eXYYVsuw+N+6K+bpbisXzQun8lLo28YKpRyH4iTY1JVZjTxQk
      </CipherValue>
    </CipherData>
  </EncryptedData>
</soapenv:Body>
</soapenv:Envelope>

```



# Encryption Key

---

```

<EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
  <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 ValueType=
        "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-
        profile-1.0#X509SubjectKeyIdentifier">RwsUhyym9TU=
      </wsse:KeyIdentifier>
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>
  <CipherData>
    <CipherValue>
      oYEVhA/LgND2hZ6bAappw4ucOlftz7Xv1Xsz8xSckedEBn+gfEmKWkauhWPSAzxjMugOaK+AoXG0986ObznjLN9J
      bLNzCSLEKynuT5EiA6hGIKaQAhhr/FE8txclI5kWe7DrYO+i3sN4nKYZJEKraJHqX/Q7e74ejN4YD0Ua10=
    </CipherValue>
  </CipherData>
  <ReferenceList>
    <DataReference URI="#wssecurity_encryption_id_5" />
  </ReferenceList>
</EncryptedKey>

```



# Message Security Characteristics

---

- Security information is contained in and travels with each SOAP message.
- Security information becomes transport-independent.
  - Identity/Authentication
  - Integrity
  - Confidentiality
  - Non-Repudiation
- Based on:
  - XML digital signature to provide integrity
  - XML encryption to provide confidentiality
  - Security tokens to provide identity/authentication



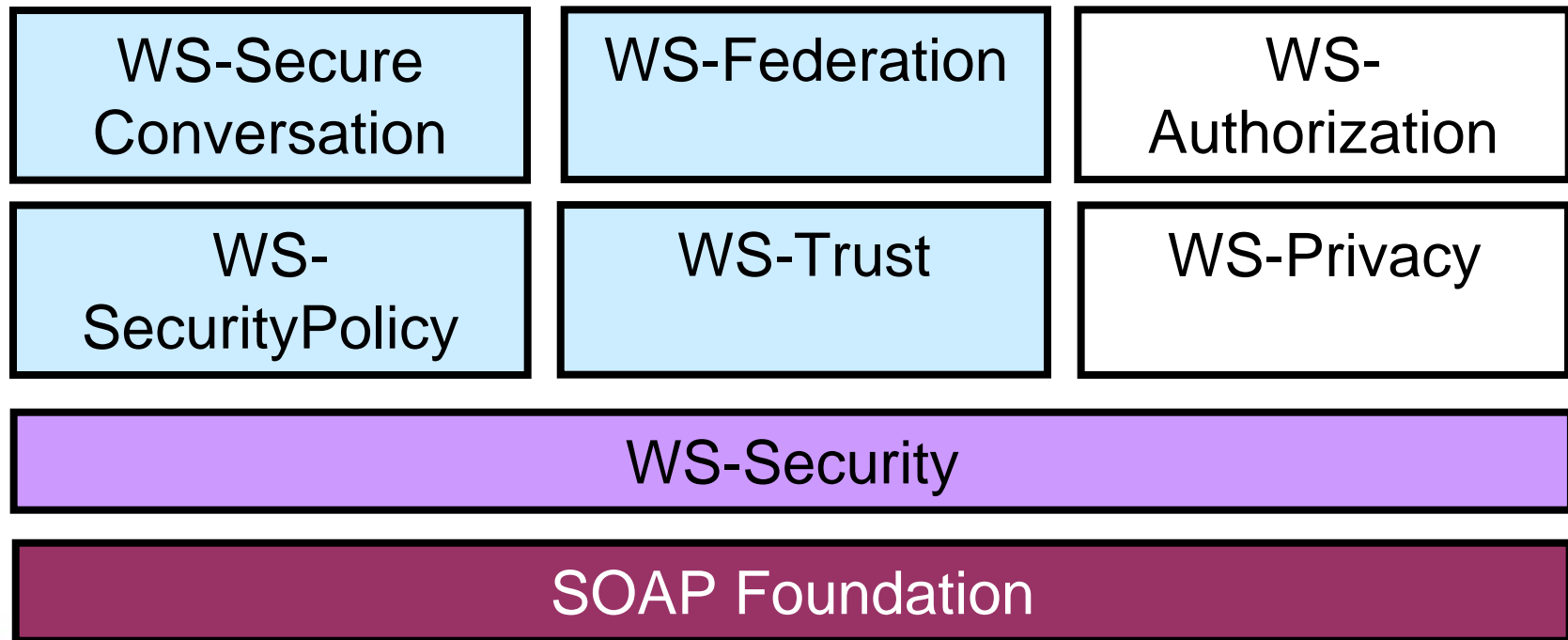
# WS Security Summary

---

- Lots to Understand:
  - There are a lot of technologies to understand when working in this space. Expect to take some time coming up to speed. Fortunately tooling helps (a little) to relieve the need to understand it thoroughly.
  
- Transport Security:
  - Consider using transport level security if you don't have the need for message based characteristics.
  
- Message Security:
  - Message based security as provided by WS-Security allows the advantages of maintaining security beyond the context of a connection (across intermediaries and when the message is persisted) and for content based routing.
  
- WS-Security is Trustworthy:
  - While WS-Security is new, it is built on security technologies that are very mature and proven. You should be comfortable implementing critical systems using WS-Security.

# The Web services security model

WS-Security only addresses a subset of the security services required - other security specifications rely on the WS Security foundation



# Web Services Security

## ■ WS-SecurityPolicy

- Defines general security policy assertions which apply to Web Services Security:
  - SOAP Message Security,
  - WS-Trust, and
  - WS-SecureConversation.

## ■ WS-Trust

- Defines extensions that build on WS-Security to both provide a framework for
  - requesting and issuing security tokens
  - brokering trust relationships

## ■ WS-SecureConversation

- Defines extensions that build on WS-Security and WS-Trust to provide secure communication across one or more messages.
  - Specifically, this specification defines mechanisms for establishing and sharing security contexts and for deriving keys from established security contexts (or any shared secret).



# WS SecurityPolicy Language

---

- *a.k.a.* WS SecurityPolicy
- Security Assertion Model
  - The goal to provide richer semantics for combinations of security constraints
    - what parts of a message are being secured
      - ✓ Protection Assertions
    - general aspects or pre-conditions of the security
      - ✓ Conditional Assertions
    - the security mechanism
      - ✓ Security Binding Assertions that is used to provide the security,
    - the token types and usage patterns
      - ✓ Supporting Token Assertions used to provide additional claims, and
    - token referencing and trust options
      - ✓ WSS and Trust Assertions

# Propagating security policy with WS-SecurityPolicy

---

- Defines a model and syntax to describe and communicate security policy assertions within the larger WS-Policy framework.
  - It covers assertions for
    - required security tokens,
    - message integrity,
    - message confidentiality,
    - message visibility to SOAP intermediaries,
    - constraints on the security header,
    - and the age of a message.
  - Assertions can be made for the security features provided by
    - WS-Security, WS-Trust and WS-SecureConversation.
  - Version 1.1 released in July 2005.
  - Contributors: Microsoft, IBM, Verisign, RSA Security.
- Designed to be flexible with respect to token types, cryptographic algorithms and mechanisms used, including using transport level security.



## Example – Security policy (1 of 4)

---

```
<wsp:Policy>
  <sp:SymmetricBinding>
    <wsp:Policy>
      <sp:ProtectionToken>
        <wsp:Policy>
          <sp:KerberosV5APREQToken
            sp:IncludeToken=".../IncludeToken/Once" />
          </wsp:Policy>
        </sp:ProtectionToken>
        <sp:SignBeforeEncrypting />
        <sp:EncryptSignature />
      </wsp:Policy>
    </sp:SymmetricBinding>
    <sp:SignedParts>
      <sp:Body/>
      <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
    </sp:SignedParts>
    <sp:EncryptedParts>
      <sp:Body/>
    </sp:EncryptedParts>
  </wsp:Policy>
```

## Example – Security policy (2 of 4)

1. Indicates that this is a policy statement and that all assertions contained by the `<wsp:Policy>` element are required to be satisfied.

```
<wsp:Policy> ... </wsp:Policy>
```

2. Specifies the kind of security binding in force.
  - In this case, messages will be symmetrically encrypted using XML Encryption from the WS-Security specification.
    - The same `<sp:EncryptionToken>` is used from initiator to recipient and back again.
    - If an `<sp:ProtectionToken>` assertion is specified, the specified token populates both token properties and is used as the basis for both encryption and digital signature in both directions.

```
<sp:SymmetricBinding> .... </sp:SymmetricBinding>
```

## Example – Security policy (3 of 4)

3. <wsp:Policy> contains assertions that qualify the behavior of the <sp:SymmetricBinding> assertion.
  - **<sp:ProtectionToken>**
    - is a ProtectionToken assertion.
      - ✓ The nested <wsp:Policy> element asserts the type of token to be used for the ProtectionToken.
      - ✓ In this case, a Kerberos V5 APREQ token is to be used by both parties in the message exchange for protection.
  - **<sp:SignBeforeEncrypting>**
    - states that signatures are generated over plaintext rather than ciphertext (you create a signature then encrypt it).
  - **<sp:EncryptSignature>**
    - indicates that the signature over the signed messages parts is required to be encrypted.

```
<wsp:Policy>
  <sp:ProtectionToken>
    <wsp:Policy>
      <sp:KerberosV5APREQToken
        sp:IncludeToken=".../IncludeToken/Once" />
    </wsp:Policy>
  </sp:ProtectionToken>
  <sp:SignBeforeEncrypting />
  <sp:EncryptSignature />
</wsp:Policy>
```



## Example – Security policy (4 of 4)

---

4. `<sp:SignedParts>` declares which message parts are to be signed by the primary signature.
- In this case the `<soap:Body>` element and any SOAP headers in the WS-Addressing namespace.

```
<sp:SignedParts>  
  <sp:Body/>  
  <sp:Header namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing" />  
</sp:SignedParts>
```

5. `<sp:EncryptedParts>` declares which message parts are to be encrypted
- In this case just the `<soap:Body>` element.

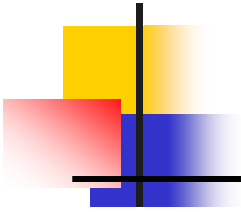
```
<sp:EncryptedParts>  
  <sp:Body/>  
</sp:EncryptedParts>
```

# Propagating trust relationships using WS-Trust

---

- Describes a framework for **trust models** that enables Web services to **securely interoperate**, by establishing and managing trust relationships.
  - Contributors:
    - IBM, BEA Systems, Microsoft, Layer 7 Technologies, Oblix, VeriSign, Actional, Computer Associates, OpenNetwork Technologies, Ping Identity, Reactivity, RSA Security.
  - Draft version updated February 2005.
  
- Defines a service model, the Security Token Service (STS).
  - Allows Web services to set up and agree on which security servers they trust and to rely on these servers.
  
- Defines a protocol for issuing, renewing, and validating security tokens:
  - Used by WS-Security to transfer the required security tokens in a way that ensures the integrity and confidentiality of these tokens.
  - Described by WS-SecurityPolicy.





# Questions?

And don't forget to do your evaluations... if you liked the session :>)

**And don't forget to do your evaluations... if you liked the session :>)**

[Software Summit](#)