lecture

Workshop on

XML and Web Service Security

How to secure XML documents, messages and sessions

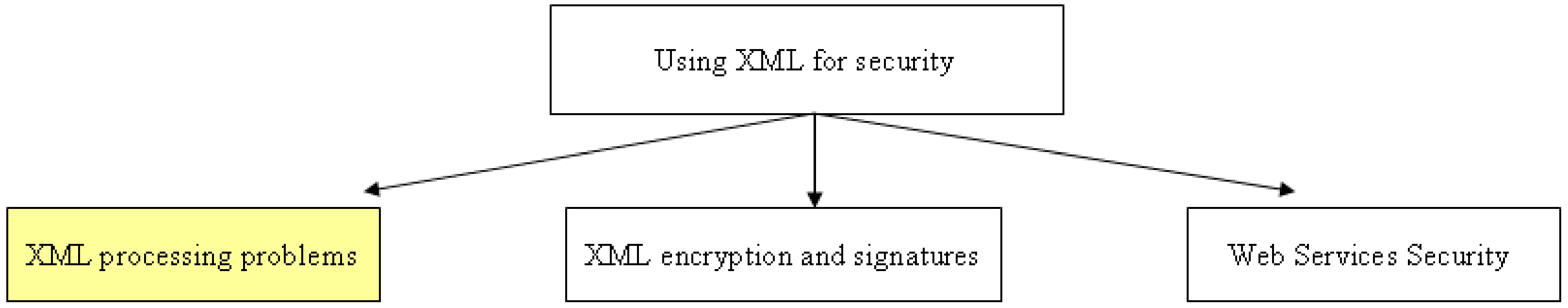
Walter Kriha

Goals

1. Attacks on XML tools and processes
2. Use of digital signatures and encryption with XML documents.
3. Canonical XML (like DER/BER in asn.1)
4. New business models for web services and what kind of security they need: message based vs. channel based security
5. The web services security stack (secure messages, trust, federation, policy)

Web Services use XML documents or fragments for communication. They rely on security mechanisms developed for XML documents. And they rely heavily on XML mechanisms like namespaces. The first part of the talk introduces those XML concepts.

Overview



- Logical vs. physical validity
- Are XSL scripts code?
- Can entities be used to steal information?
- DOS attacks using entities

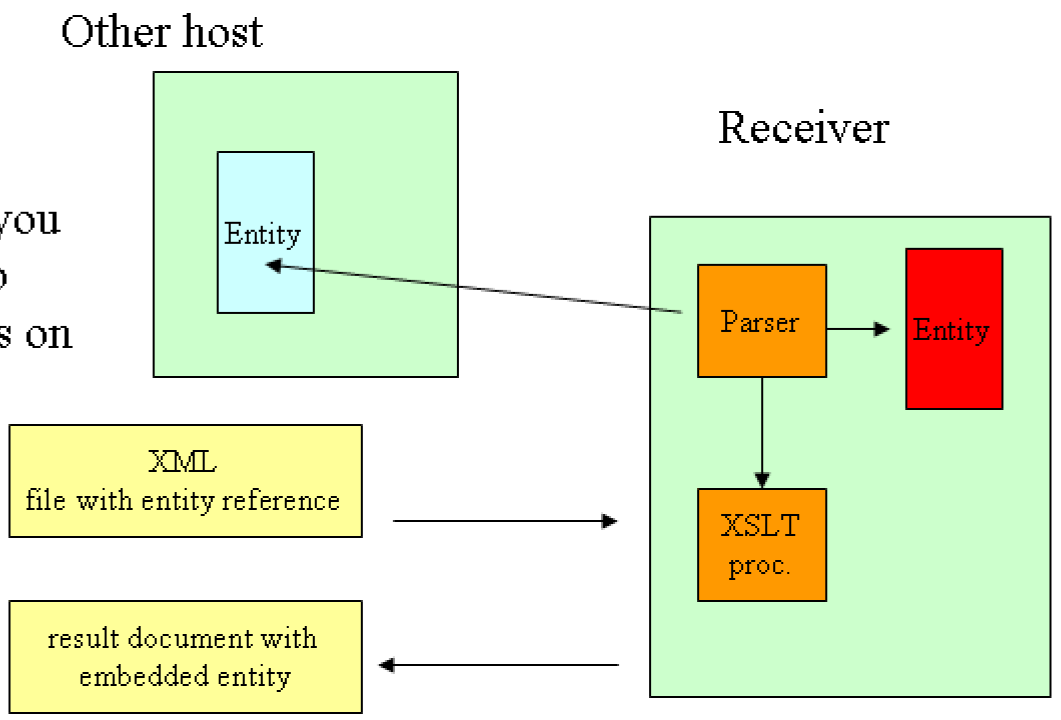
- Create canonical XML documents
- Sign XML documents or fragments
- Encrypt XML documents or fragments
- Namespaces

- Secure requests through intermediates
- Implementation independent security
- Channel based vs. document based security

The XML tools and processing steps can create a security problem by themselves. Don‘t get fooled by the „XML is only text“ assumption! Web Services are expressed with text but have the power of an RPC system like CORBA or DCOM.

Malicious documents?

If you offer a rendering service you might be abused to create artificial hits on some host.


Other host
Entity
Receiver
Parser
Entity
XSLT proc.
XML file with entity reference
result document with embedded entity

Does your XML processing system check the URIs of entity references BEFORE accessing them?

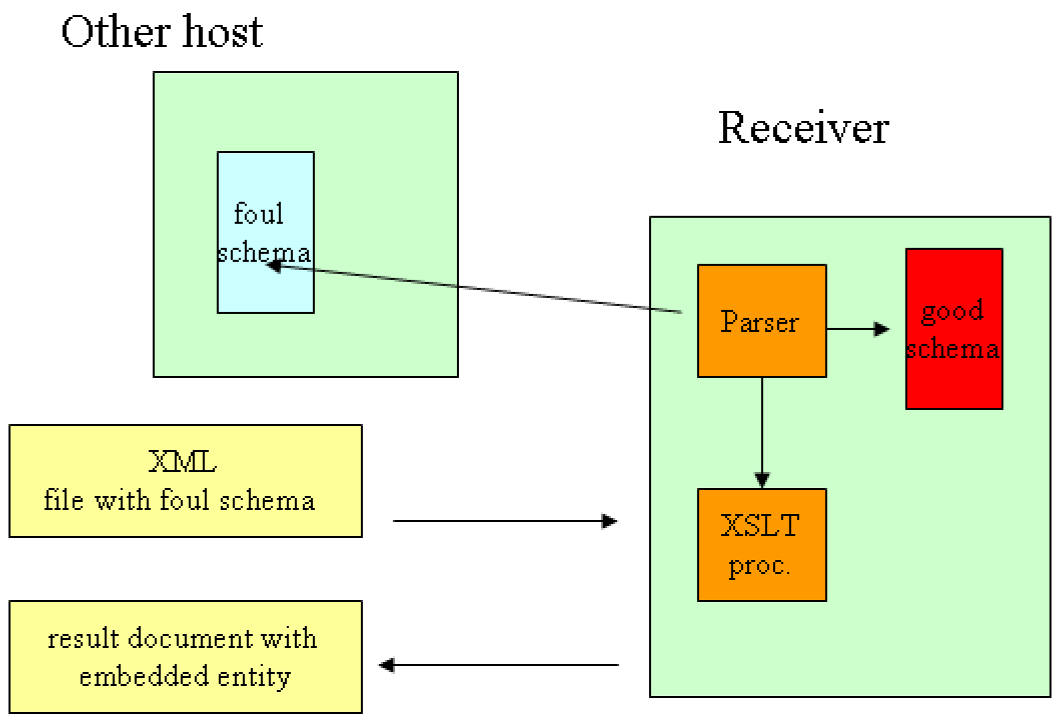
XML has some mechanisms that pose security problems by themselves – e.g. entities which are referenced automatically by a parser and which could be used to create denial-of-service attacks through the construction of a large number of those references. Or worse: those references could point anywhere on the target server and might pull secret information from such a server. Those problems are NOT the main focus of this lecture but they remind us on common vulnerabilities. Both examples have been taken from the XML-DEV mailing list (Miles Sabin, R.Tobin)

Extension Functions in XSLT

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transform version='1.0'>
<xsl:output method="html„ encoding="ISO-8859-1„ indent="no"/>
<!-- ============================================================ -->
<xsl:script language=„java“ implements-prefix=„sy“ src=„java:java.util.system“/>
<xsl:template match="*">
  <xsl:message>
    <xsl:text>No template matches </xsl:text>
    <xsl:value-of select=„sy:exec()"/>
    <xsl:text>.</xsl:text>
  </xsl:message>
```

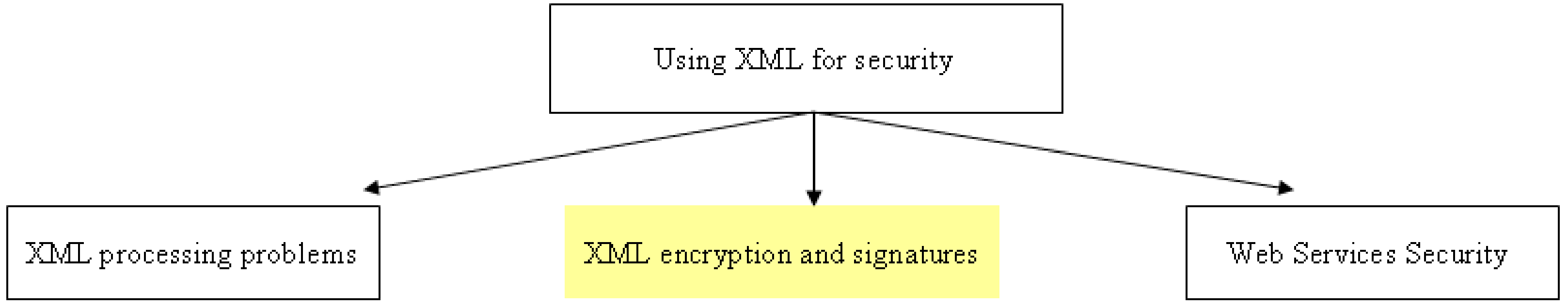
Calling extension functions from XSLT is easy. Several language bindings are supported (java, javascript etc.). What userid and rights is your XSLT processor using when you do server side processing of requests? (M.Kay, XSLT 2nd edition, page 568ff.)

Suppressing Validation


Other host
foul schema
Receiver
Parser
good schema
XML file with foul schema
XSLT proc.
result document with embedded entity

James Clark mentioned recently an especially evil way to work around validation: „Suppose an application is trying to use validation to protect itself from bad input. It carefully loads the schema cache with the namespaces it knows about, and calls validate(). Now the bad guy comes along and uses a root element from some other namespace and uses xsi:schemaLocation to point to his own schema that that has a declaration for that element and uses <xs:any namespace="##any„ processContents="skip"/>. Won't they just have almost completely undermined any protection that was supposed to come from validation?"

Overview



- Logical vs. physical validity
- Are XSL scripts code?
- Can entities be used to steal information?
- DOS attacks using entities

- Create canonical XML documents
- Sign XML documents or fragments
- Encrypt XML documents or fragments
- Namespaces

- Secure requests through intermediates
- Implementation independent security
- Channel based vs. document based security

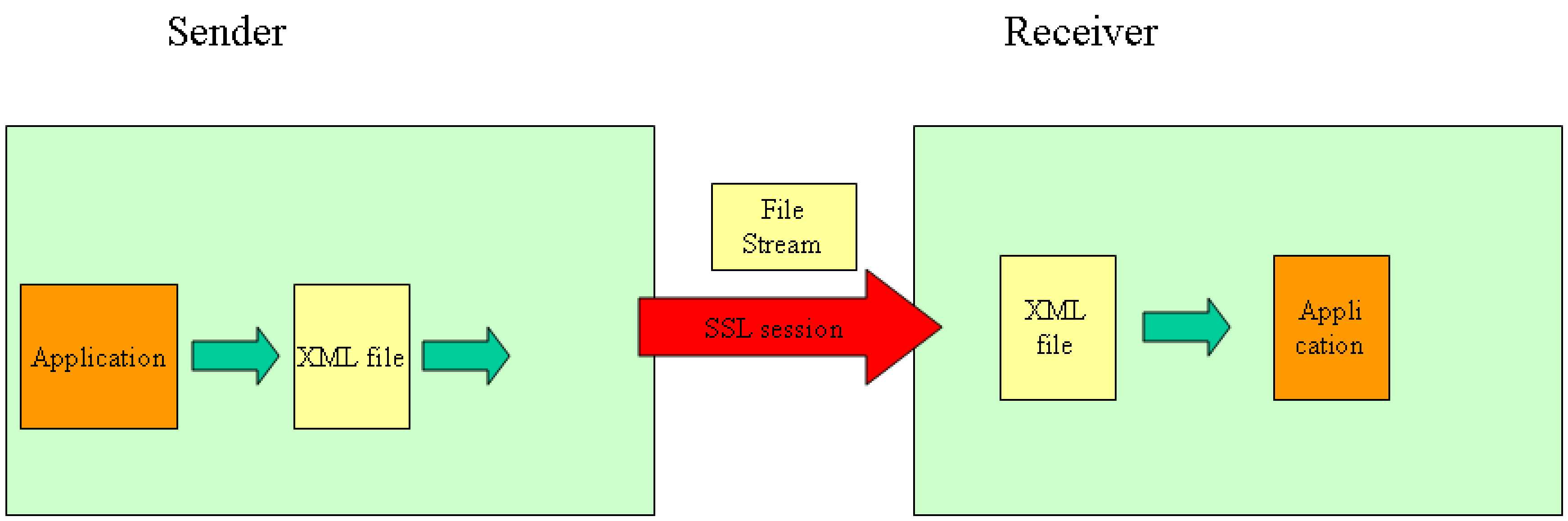
At the base of web services security are mechanisms which provide integrity and confidentiality for XML documents and fragments. XML-DSIG and XML-ENC are standards which allow digital signatures and encrypted areas to be embedded in XML documents

XML Standards related to Security

- XML Digital Signatures
- XML Encryption
- Canonical XML (C14N)
- related XML basic standards (XML Infoset, Namespaces etc.)

We will see how all these technologies are needed to solve the security problems caused by the new internet based, distributed and collaborative business model of web services. But first a look at XML processing of documents is in order.

Sending XML Securely (Today)



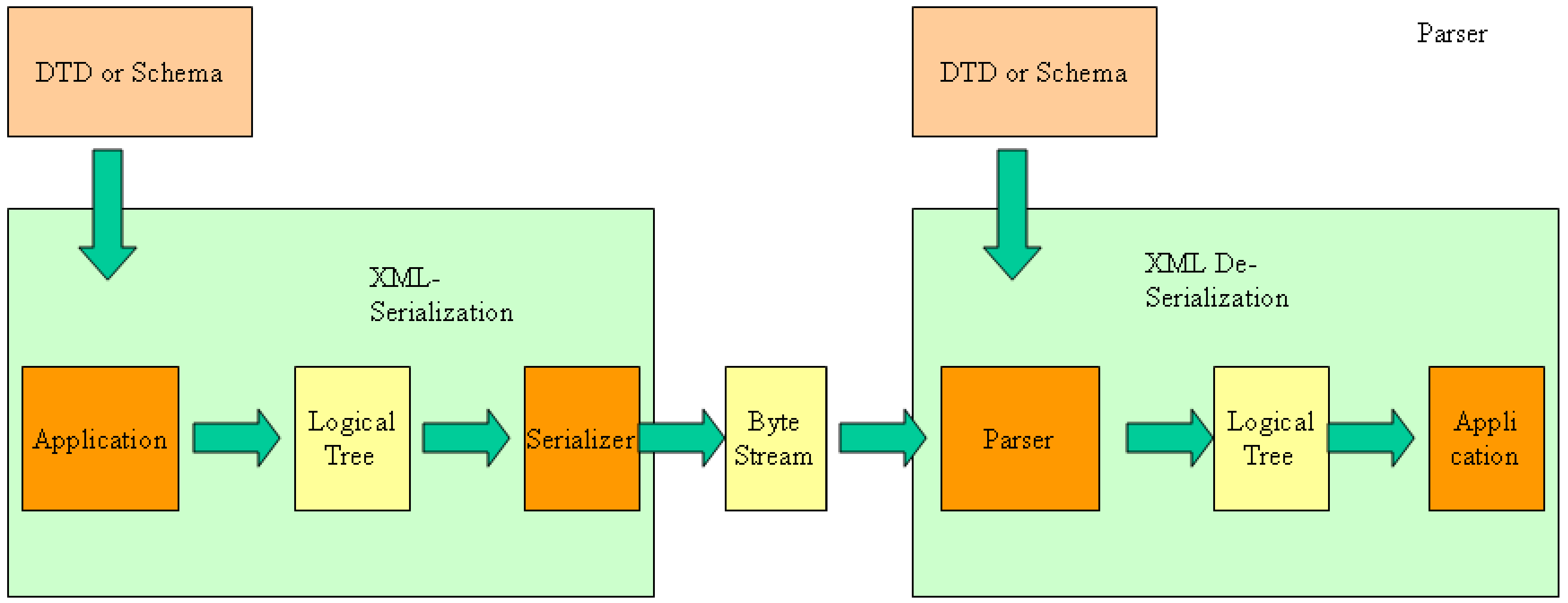
Today the easiest solution to send an XML file securely (with authentication, integrity and confidentiality provided by the transport-level protocol) is to use SSL/TLS. There are a number of disadvantages associated with this solution:

Problems with the SSL based solution

1. Security is provided by runtime code (SSL middleware etc.) NOT tied to the document itself. If the document is forwarded to another receiver its security is depending on the new security context. This poses a major problem for web services later.
2. The receiver does not have non-repudiation: no signature attached.
3. Worse yet: if signatures would be used, how would the signer know what the receiver is able to understand and process? How would we communicate the keys etc. used for it to our receiver?
4. Same problem with encryption.
5. Encryption of parts of the document is possible but there is no mechanism to create several signatures and encrypted blocks for multi-party document exchange.
6. The use of signatures is problematic because sender and receiver can treat the same logical XML content in a physically different way – thus voiding all signatures.

These problems are pretty much the same as for secure e-mail. They are caused by the same reason: using something that is SESSION oriented to transport single MESSAGES or DOCUMENTS. Eric Rescorla shows the problems with SSL when used for to secure e-mail. His „SMTP over TLS“ chapter sets the stage for most of the things in this lecture. Surprisingly Web Services seem to fall much more into the message/document model than the connection oriented model. Solutions for messages/documents are usually closer to the application (end-to-end argument in security). The latest security related proposals from the Web Services Industry seem to confirm this trend.

Sending and receiving XML documents



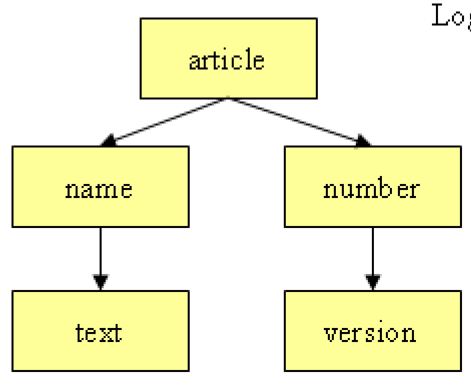
Both sender and receiver create or validate an xml instance using a schema or DTD which controls the LOGICAL content of the xml file. Different physical content can result in the SAME logical content. Unfortunately signatures e.g. work on the PHYSICAL content of an XML instance. Since serializers and parsers have considerable freedom with respect to physical content this means that a signature created over physical representation 1 (sender) may not fit to the physical representation 2 (receiver) re-created by the parser even though the logical content is the same: Signatures work on bit-level, not on XML element level (This is comparable to the C++ concept of „const“ methods which guarantee BITWISE constness of an object: you cannot even cash something in a const method)

Logical vs. Physical Representation

DTD/Schema:

```
<!ELEMENT article (name, number)
<!ELEMENT name (#PCDATA)
<!ELEMENT number (#EMPTY)
<!ATTLIST article version CDATA
#rREQUIRED
```

Logical Rep.



Physical instance I

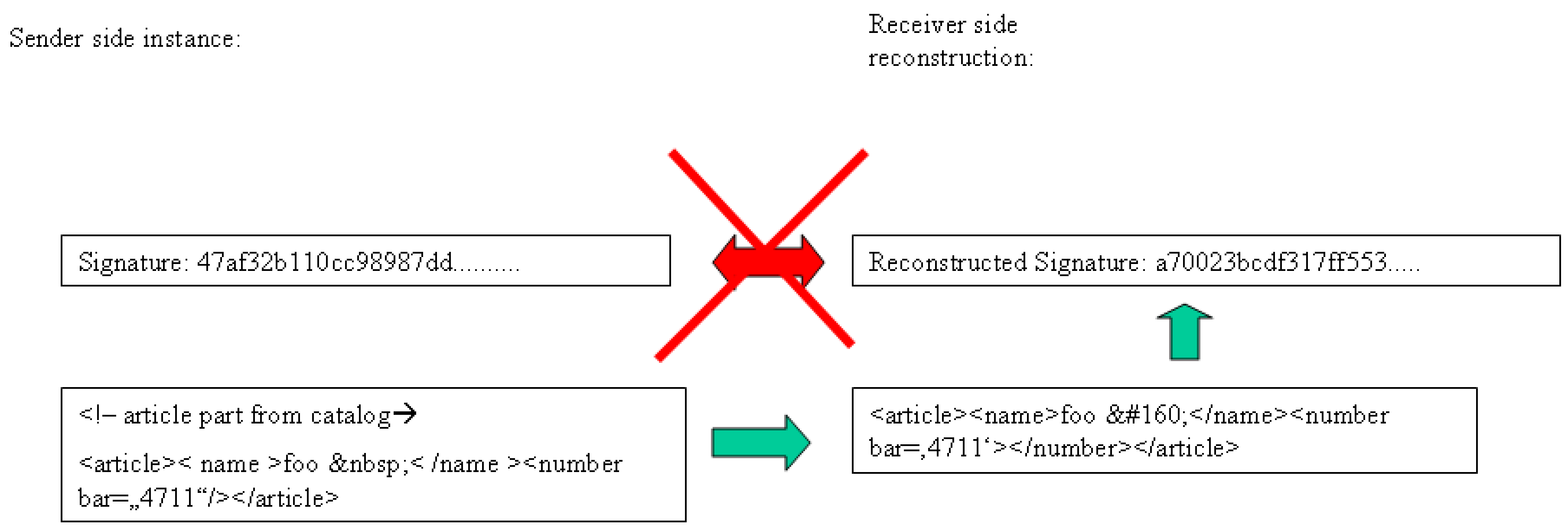
```
<!– article part from catalog→
<article>< name >foo &nbsp;< /name ><number
bar=„4711“/></article>
```

Physical instance II

```
<article><name>foo &#160;</name><number
bar=‚4711‘></number></article>
```

Watch the small differences in instances: whitespace in element names, character entities vs. character codes, special „empty“ syntax for number or not, whitespace in attributes, double quotes vs. single quotes etc. Please note: BOTH instances are a valid representation of the DTD or Schema because they both fit to the logical model above. For most applications the differences will not matter. But they will definitely matter if signatures over those representations are created. But XML itself has problems with it too as we will see.

Signatures over XML Instances



Signature: 47af32b110cc98987dd.........

```
<!– article part from catalog→
<article>< name >foo &nbsp;< /name ><number
bar=„4711“/></article>
```

Reconstructed Signature: a70023bcdf317ff553.....

```
<article><name>foo &#160;</name><number
bar=‚4711‘></number></article>
```

Once the signature is reconstructed on the receiver side it does not fit to the originally created signature – due to the differences in physical representation that serializer and parser used. It does not matter that the logical content is exactly the same.

Canonicalization of XML Instances

Sender side instance:

```
<!– article part from catalog→

<article>< name >foo &nbsp;< /name ><number
bar=„4711“/></article>
```

Canonicalized form:

```
<article><name>foo &nbsp;</name><number
bar=‚4711‘></number></article>
```

Canonical XML defines how a canonical instance needs to look like:

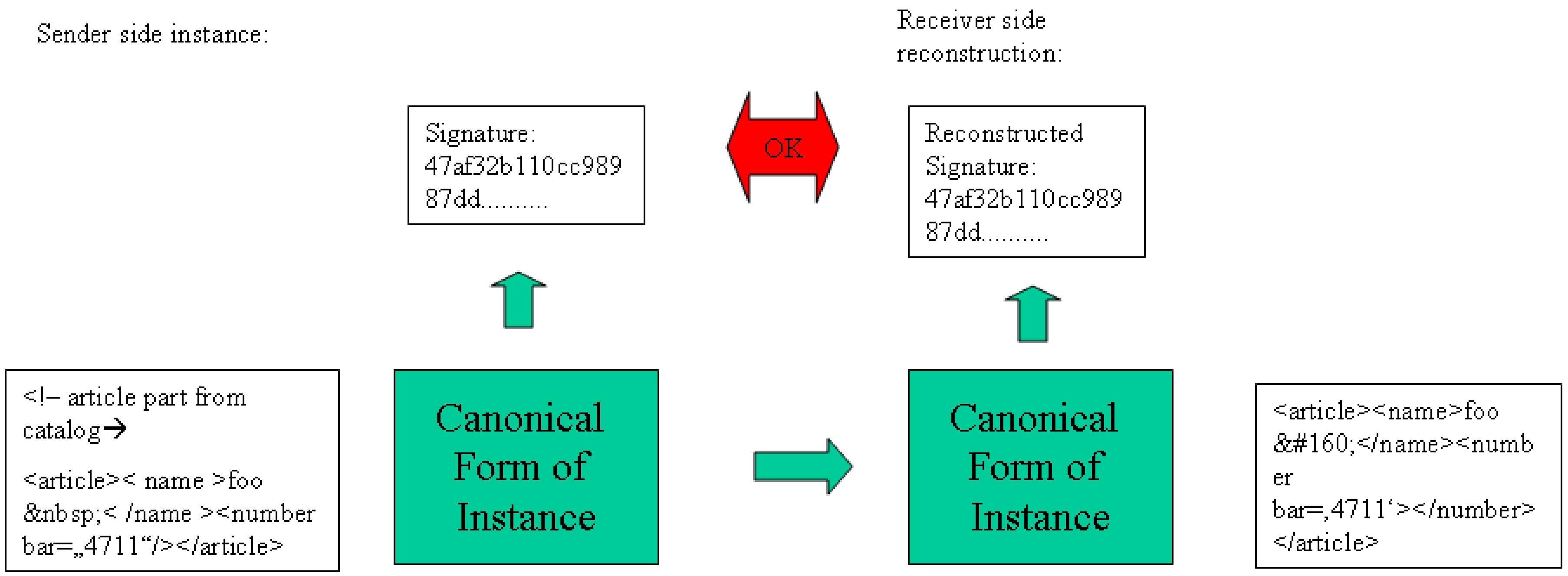
-UTF-8 encoding, line breaks normalized to #xA, attribute values normalized

-character references expanded, CDATA replaced with content, DTD and XML declaration removed, empty tags (<e/>) replaced with tag pairs (<e></e>)

-special characters replaced with character references, redundant namespaces removed, fixed attributes expanded, sorted according to defined order for attributes and namespaces

-(from Michael Kay, XSLT 2nd edition, pg. 71)

Signatures over canonical XML Instances

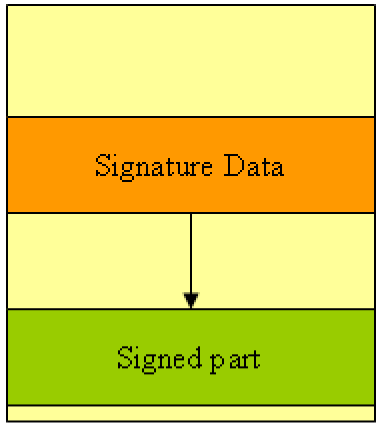


Sender side instance:
Receiver side reconstruction:
Signature: 47af32b110cc989 87dd.........
OK
Reconstructed Signature: 47af32b110cc989 87dd.........
<!– article part from catalog→
<article>< name >foo &nbsp;< /name ><number bar=„4711“/></article>
Canonical Form of Instance
Canonical Form of Instance
<article><name>foo &#160;</name><number bar=‚4711‘></number> </article>

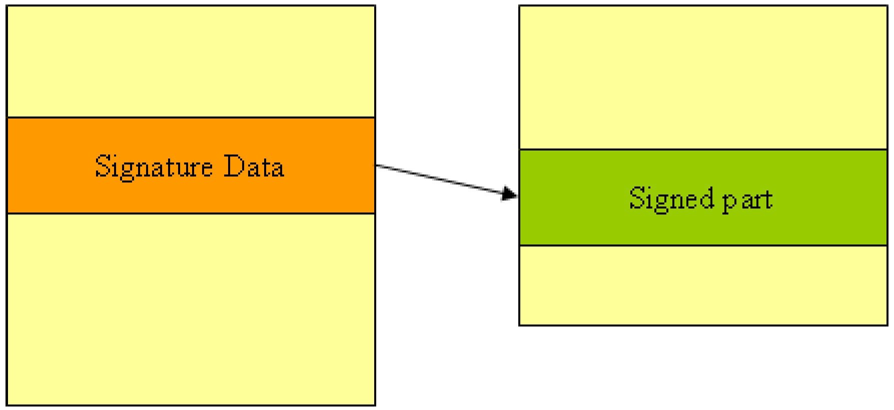
Signatures are constructed and compared based on the CANONICAL form of the instance. (see www.w3.org) for the C14N standard.

XML Signatures

XML instance with
ENVELOPING signature

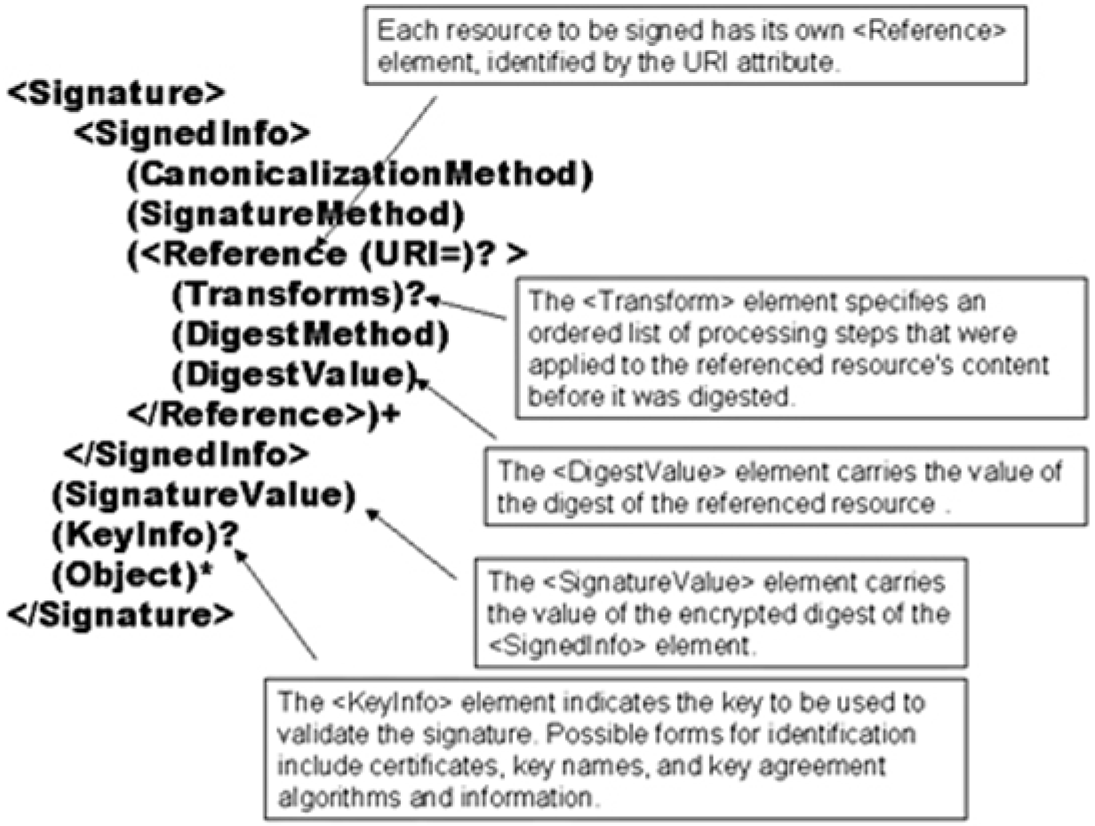


XML instance with
DETACHED signature



. XML DSIG (www.w3.org/Signature) defines a signature element which allows a signer to specify a signature over selected parts of the document. Web Services will use this feature e.g. to tie message content and message header together.

The XML DSIG „Signature“ Element



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

Each resource to be signed has its own <Reference> element, identified by the URI attribute.

The <Transform> element specifies an ordered list of processing steps that were applied to the referenced resource's content before it was digested.

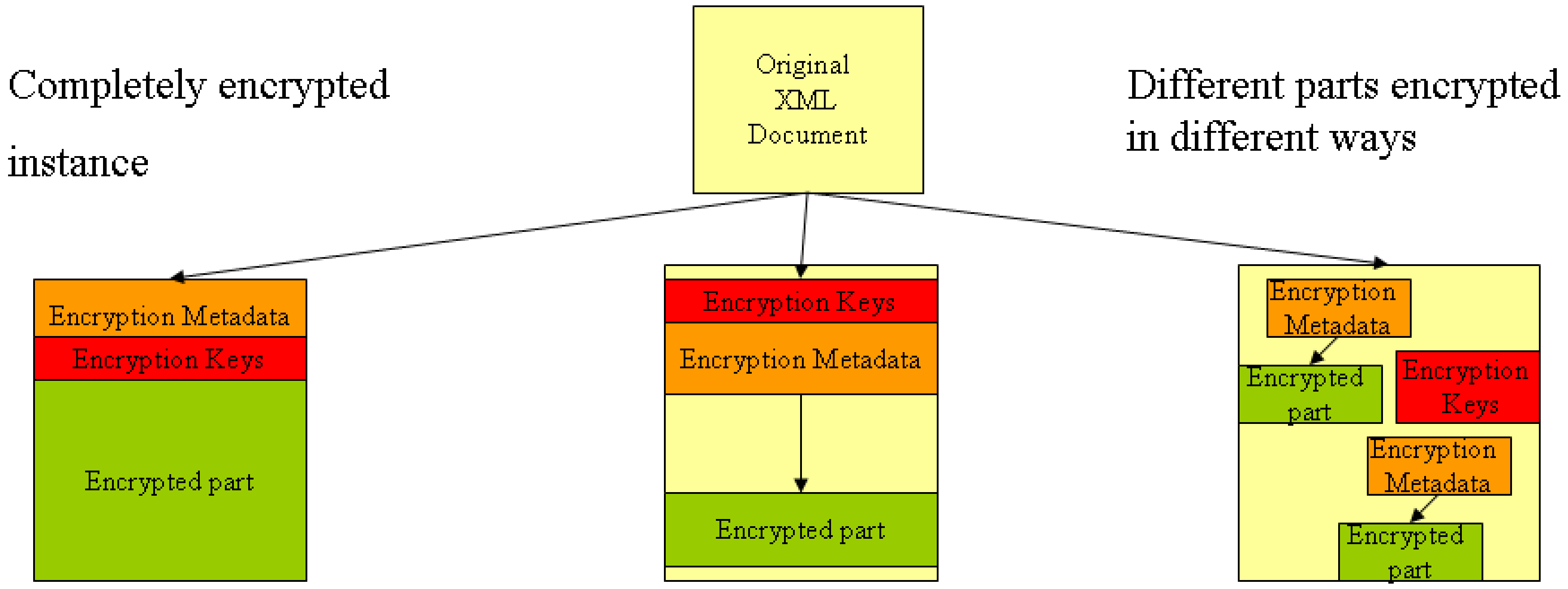
The <DigestValue> element carries the value of the digest of the referenced resource .

The <SignatureValue> element carries the value of the encrypted digest of the <SignedInfo> element.

The <KeyInfo> element indicates the key to be used to validate the signature. Possible forms for identification include certificates, key names, and key agreement algorithms and information.

From Ed Simon et.al, (see Resources). Note that „object“ will only be there if the signature is „enveloping“ otherwise the reference element will point with the URI to an out-of-document object. Transforms defines e.g. that the object has been canonicalized. Information that the receiver needs for verification is contained in the DigestMethod, SignatureValue and possibly also in the KeyInfo element (e.g. which public key was used to sign the disgest)

Encrypting XML documents



Especially in a multi-party communication system encryption is difficult to realize. The core problem is how to authorize and control the viewing of different parts by different parties. There is also the problem of known plain-text attacks if the tags are well known because the DTD is known. Web Services use this feature when passing messages via several intermediates.

The EncryptedData Element

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

Type = element or content

Algorithm used

key information element from XML DSIG

raw encrypted data (by value or reference)

additional info about generation of encrypted type

EncryptedData element which contains (via one of its children's content) or identifies (via a URI reference) the cipher data. When encrypting an XML element or element content the EncryptedData element replaces the element or conten (respectively) in the encrypted version of the XML document. (from XML Encryption spec. http://www.w3.org/Encryption/2001/Drafts/xmlenc-core/

Coding Example of XML Encryption

```
<?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>
```

In this example form the XML encryption specification only the CONTENT of the credit card information has been encrypted and is enclosed in the CipherValue element. The specification also defines rule about the relation between encryption and signatures, e.g. in which order they should be applied. When data is encrypted, any digest or signature over that data should be encrypted as well to avoid guessing attacks.

XML Namespaces

```
<schema xmlns='http://www.w3.org/2001/XMLSchema' version='1.0'
                xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
                xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'
                targetNamespace='http://www.w3.org/2001/04/xmlenc#'
                elementFormDefault='qualified'>
           <import namespace='http://www.w3.org/2000/09/xmldsig#'
                  schemaLocation='http://www.w3.org/TR/2002/REC-xmldsig-
core-20020212/xmldsig-core-schema.xsd'/>
```

← namespace used to denote a schema and how instance and schemas are related

```
http://www.w3.org/2001/04/xmlenc#tripledes-cbc
```

← namespace used to define different encryption algorithms

```
<ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
 <pay:PaymentInfo xmlns:pay='http://example.org/paymentv2'>
<dummy
xmlns:foo="http://example.org/foo"><pay:One><foo:One/></pay:One></dummy>
```

← namespace used within instances to avoid name clashes between elements of different schemas

Despite an ongoing discussion about their value, namespaces are increasingly used to denote all kinds of things. If you want to work with XML you will need to understand namespaces. Important: There is absolutely NO requirement that a namespace URI really points to a web resource. In most cases the URI is just used to make definitions unique (basically by using the DNS name system which already has unique names). Combine the URI with the tag name to get the fully qualified element name.

Frequently used Namespaces

The following namespaces are frequently used in Web Service Security:

Prefix	**Namespace**
S11	http://schemas.xmlsoap.org/soap/envelope/
S12	http://www.w3.org/2003/05/soap-envelope
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
wst	http://schemas.xmlsoap.org/ws/2004/04/trust
ds	http://www.w3.org/2000/09/xmldsig#
xenc	http://www.w3.org/2001/04/xmlenc#
wsp	http://schemas.xmlsoap.org/ws/2002/12/policy
wsa	http://schemas.xmlsoap.org/ws/2004/03/addressing
xs	http://www.w3.org/2001/XMLSchema

Please note that the prefix is not part of the namespace and will be replaced by the namespace URI. An element <foo> in namespace with prefix „ds“ will finally become: `http://www.w3.org/2000/09/xmldsig:foo`

Are Signatures and Encryption all that is needed?

Please note that we still have other unsolved problems. Our view right now was very static and document centric. In a more message oriented environment one has e.g. to solve the problem of security context negotiation

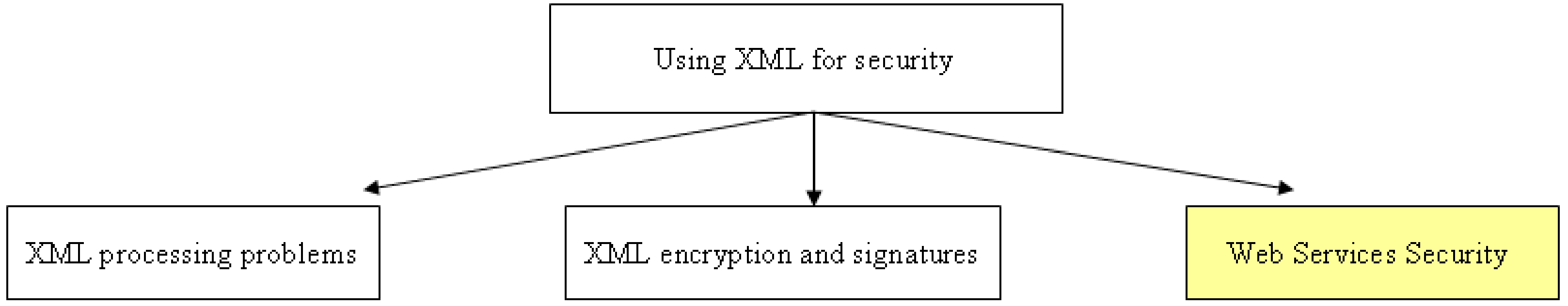
-what kind of security and encryption is required by the provider of a service?

-How does a potential requester know about those requirements?

-How do we establish initial trust?

Web Services intend to do business transactions over the Web and therefore need to have an answer for those problems as well.

Overview



- Logical vs. physical validity
- Are XSL scripts code?
- Can entities be used to steal information?
- DOS attacks using entities

- Create canonical XML documents
- Sign XML documents or fragments
- Encrypt XML documents or fragments
- Namespaces

- Secure requests through intermediates
- Implementation independent security
- Channel based vs. document based security

Be prepared for a whole lot of new terminology as Web Services security tries to convert existing security technology into new business models for the internet!

Goals

1. Demonstrate the Web Services business model and its implications for security models: secure delegation over intermediaries, secure messaging
2. Develop a concept of secure messaging
3. Have a look at Single Sign-On realized with the new security
4. Investigate interoperability of security mechanisms

We will derive the requirements for new security models by looking at the business model for web services. This model asks for interoperable security – quite different to the closed security domains within companies.

Web Services Security Standards and Technologies

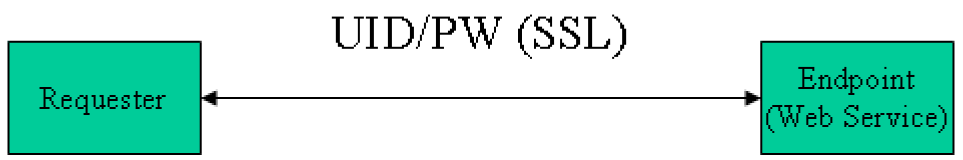
- SOAP, WSDL, UDDI: Message Envelope, Interfaces Definition and Registry
- WS-Security: Secure Messaging Definitions
- WS-Trust: How to get Security Tokens (issuing, validation etc.)
- WS-Federation (How to make security interoperable between trust domains)
- WS-Policy (How to express security requirements)
- Secure AssociationsMarkup Language (a language to express security related statements)
- WS-Reli (Rights management)
- WS-Util (Helper elements)
- WS-Authorization (express access rights)

WS-Security is the foundation of Web Services Security. We will take a close look at this specification and WS-Trust. Soap and WSDL are basic Web Services standards for messaging. If you are not familiar with them, you can find an introduction here: http://www.kriha.de/krihaorg/docs/lectures/distributedsystems/webservices/webservices.html. For a list of all Web Services standards go to: www.webservicessummit.com) Please NOTE: few of those standards are available in implementations. E.g. Web Services Enhancement 2.0 from Microsoft covers only basic Web Services security features.

The Web Services Business Model

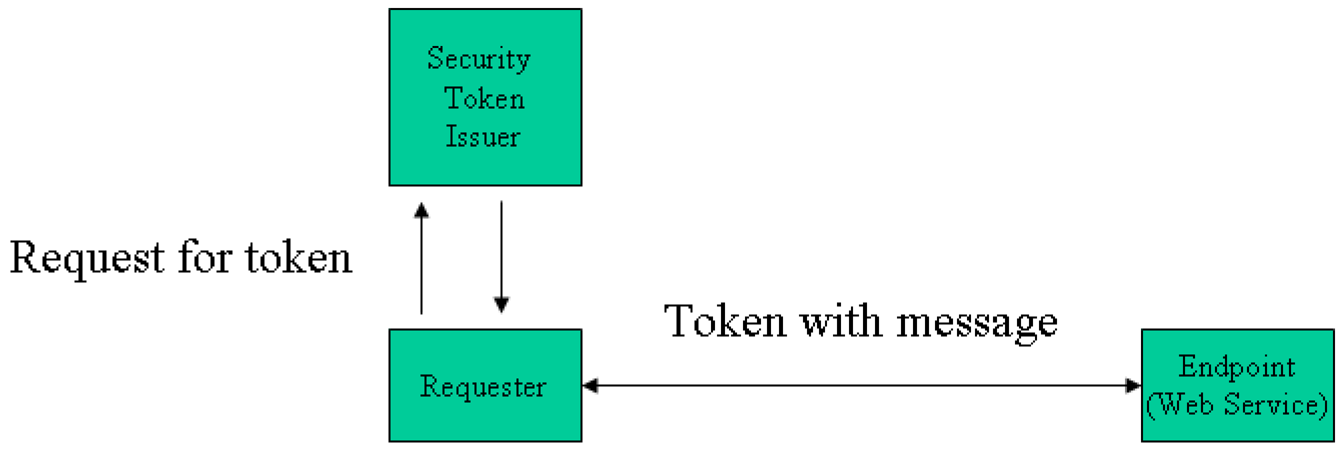
„loosely-coupled, language-neutral, platform independent way of linking applications within organizations, across enterprises, and across the Internet“ (from the roadmap to web services security, see Resources)

Scenarios (1): Direct Trust



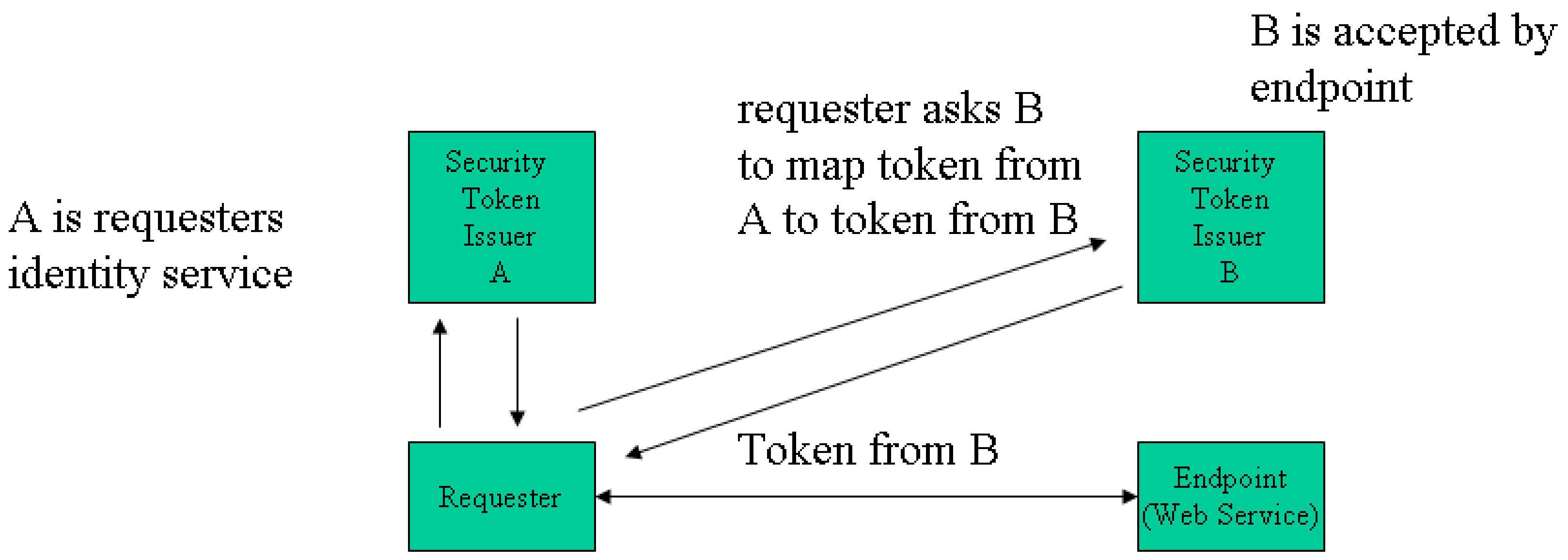
This is what is possible today. It suffers from a number of problems as we will see. The endpoint needs to implement security mechanisms as well as all administrative information to either allow the request or reject it. Business logic and security functions are both at the endpoint

Scenarios (2): Issued Security Token



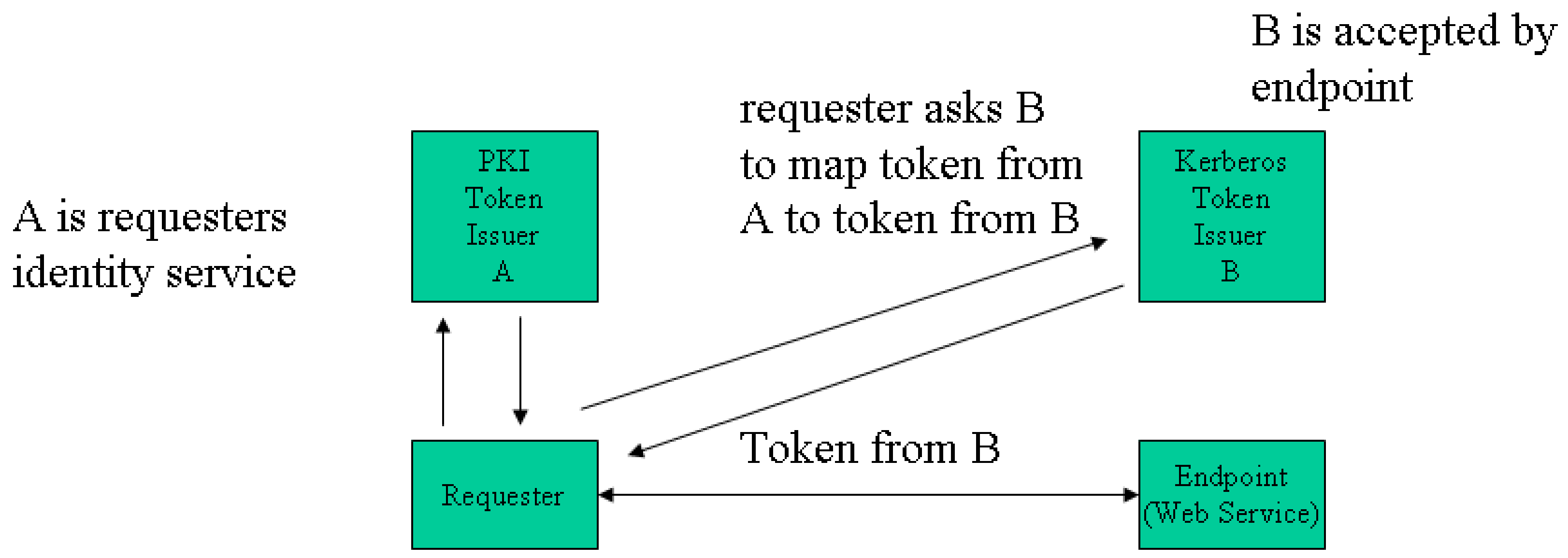
This shows authentication by a trusted third part. The endpoint trusts the security token issuer. Advantage: the endpoint need not carry administrative and technical burdens of secure authentication. Changes in technology will not affect endpoint. But the even bigger advantage is that this scheme opens the possibility for FEDERATION of security contexts. Please note that the token could also contain claims about authorization. In that case the endpoint would even accept authorization decisions of a security token issuer. A concept that might be used for Single Sign-on as well. Continuation tokens might be issued by the endpoint itself.

Scenarios (3): Federation


B is accepted by endpoint
requester asks B to map token from A to token from B
A is requesters identity service
Security Token Issuer A
Security Token Issuer B
Token from B
Requester
Endpoint (Web Service)

The endpoint won‘t accept a token from identity service A. But the endpoint trusts issuer B. So the requester asks B to map her token from A to one that will be accepted by the endpoint. This implies that issuer B trusts issuer A. How would endpoint declare that it accepts such mappings to happen? An alternative would be if requestor sends her A token directly to endpoint which forwards it to issuer B (which he trusts) and B tries to map the token to one of his own. The token issuers would probably exchange certificates from each other.

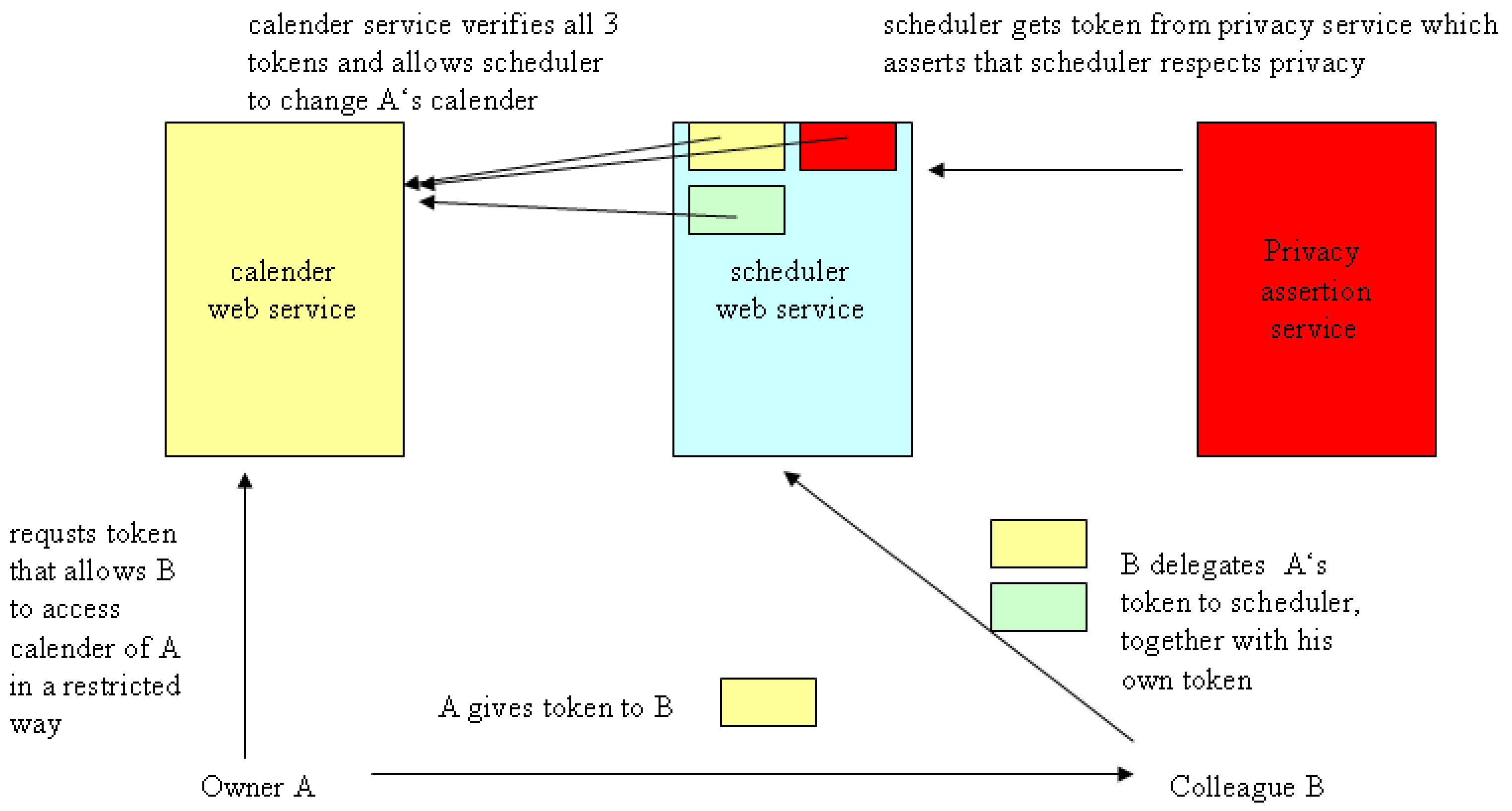
Scenarios (4): Crossing security domains



B is accepted by endpoint
requester asks B to map token from A to token from B
A is requesters identity service
PKI Token Issuer A
Kerberos Token Issuer B
Token from B
Requester
Endpoint (Web Service)

The endpoint will only accept a kerberos token from identity service B. So the requester asks B to map her certificate token from A to a kerberos symmetric key token accepted by the endpoint. This would also work between two kerberos token issuers or a kerberos key distribution center (KDC) and a PKI based issuer. In the case of two kerberos KDCs there are different levels of mutual trust possible.

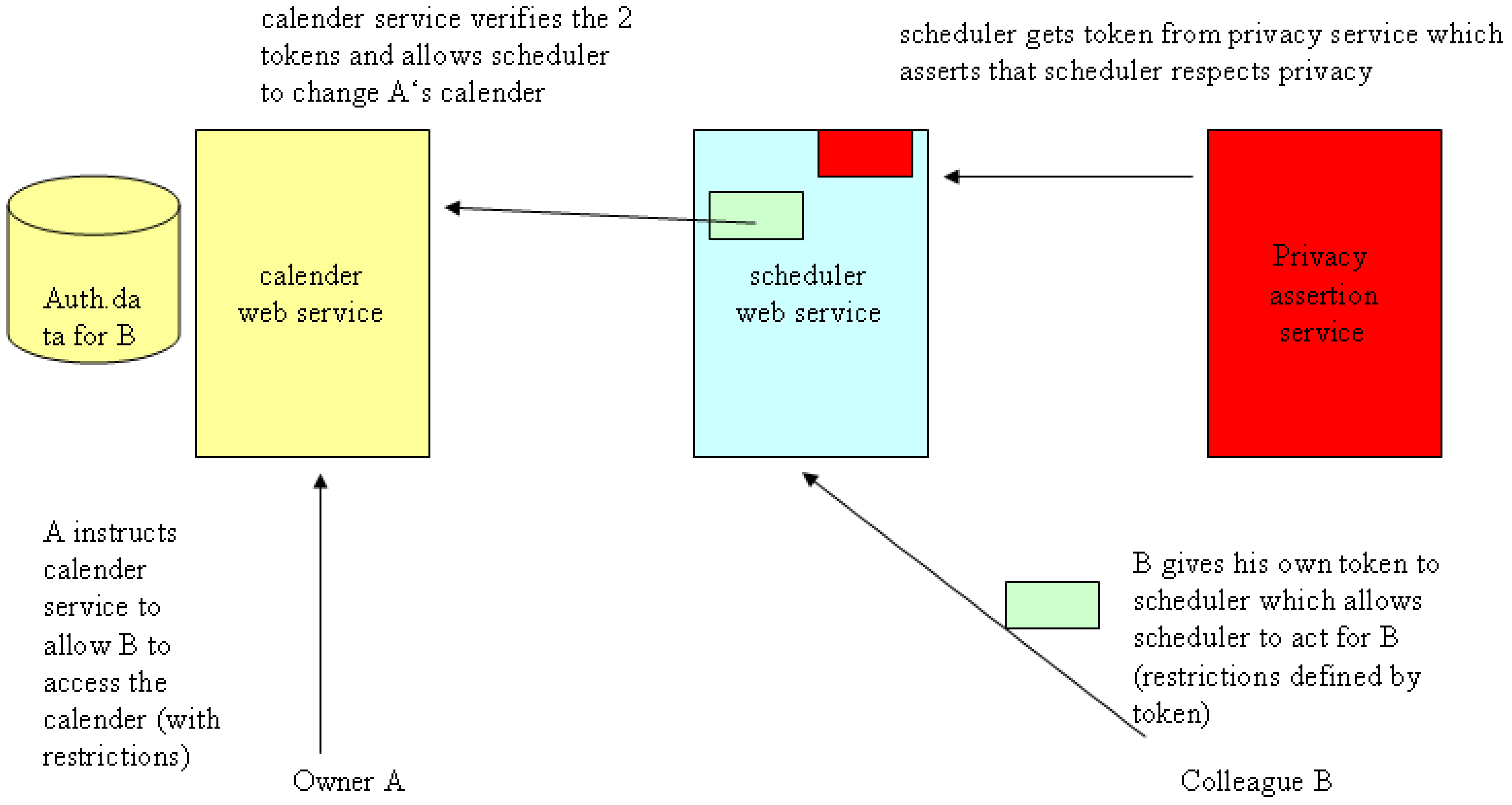
Scenarios (5): Token based delegated authorization



calender service verifies all 3 tokens and allows scheduler to change A's calender
scheduler gets token from privacy service which asserts that scheduler respects privacy
calender web service
scheduler web service
Privacy assertion service
requsts token that allows B to access calender of A in a restricted way
A gives token to B
B delegates A's token to scheduler, together with his own token
Owner A
Colleague B

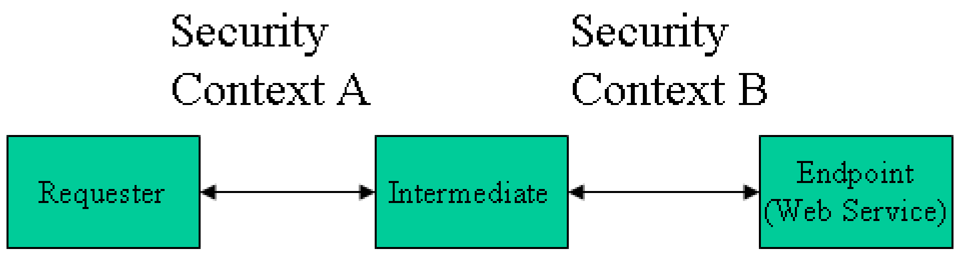
This is an example from the roadmap for ws-security. Please note that it depends on expiration data in the tokens how often A needs to re-issue an access token for B. If B needs to access the calender frequently it might be better to use endpoint access control to restrict and control B's access. See next page. Just extending the expiration dates causes problems with revocation.

Scenarios (6): Endpoint based authorization

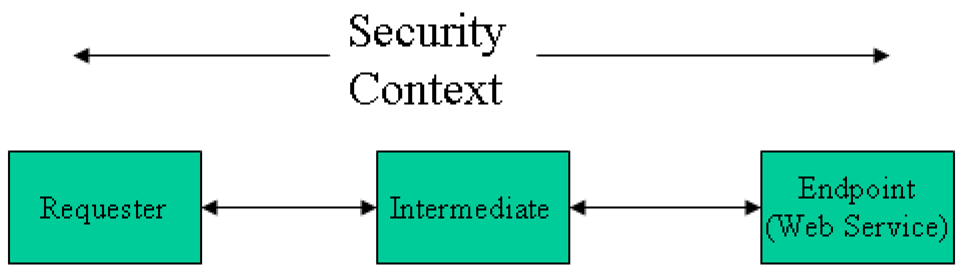

calender service verifies the 2 tokens and allows scheduler to change A's calender
scheduler gets token from privacy service which asserts that scheduler respects privacy
Auth.data for B
calender web service
scheduler web service
Privacy assertion service
A instructs calender service to allow B to access the calender (with restrictions)
Owner A
B gives his own token to scheduler which allows scheduler to act for B (restrictions defined by token)
Colleague B

Now B no longer needs a new token for every access. B still needs to identify himself and scheduler must provide proof of privacy. A can now revoke B's access right any time.

Problems (1): Intermediates



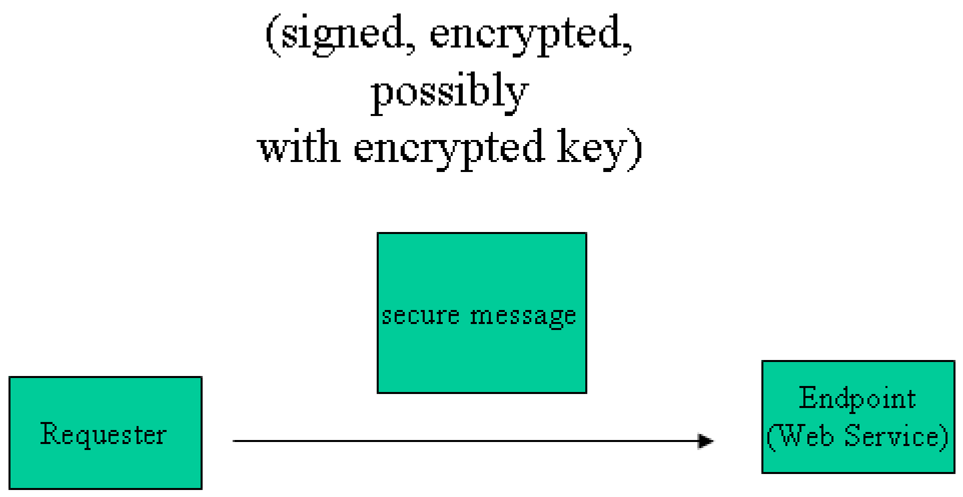
Transport level security does not allow end-to-end security because a new security context is established between each pair of communicators.



Application level security (e.g. secure messages) does allow end-to-end security because the end-point can validate the original claims made by the requester

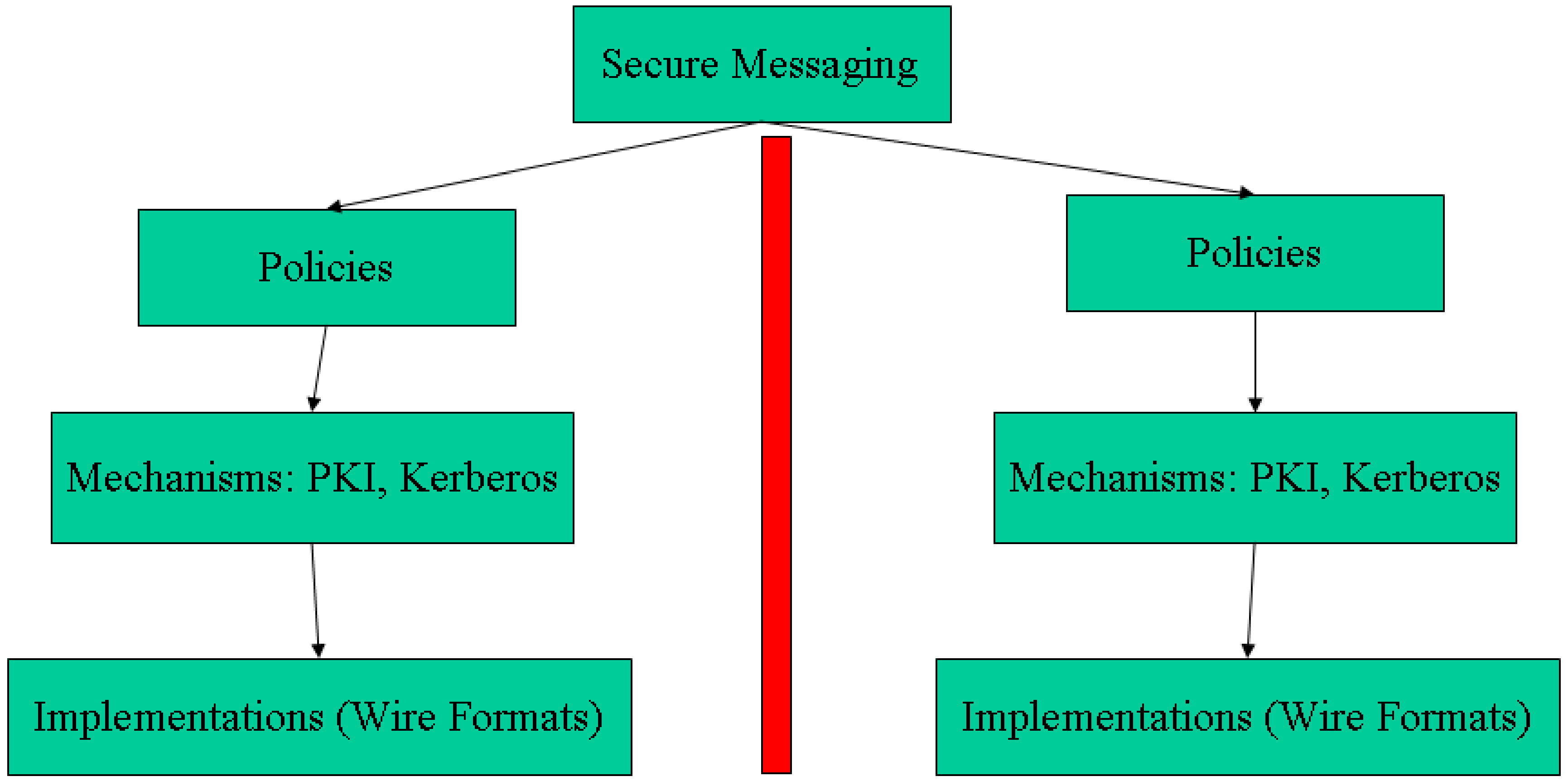
In a multi-party system secure messages are of much greater importance than transport level security. Example: Business process composition in supply-chain management. The composition can be done by an intermediate.

Problems (2): message authentication



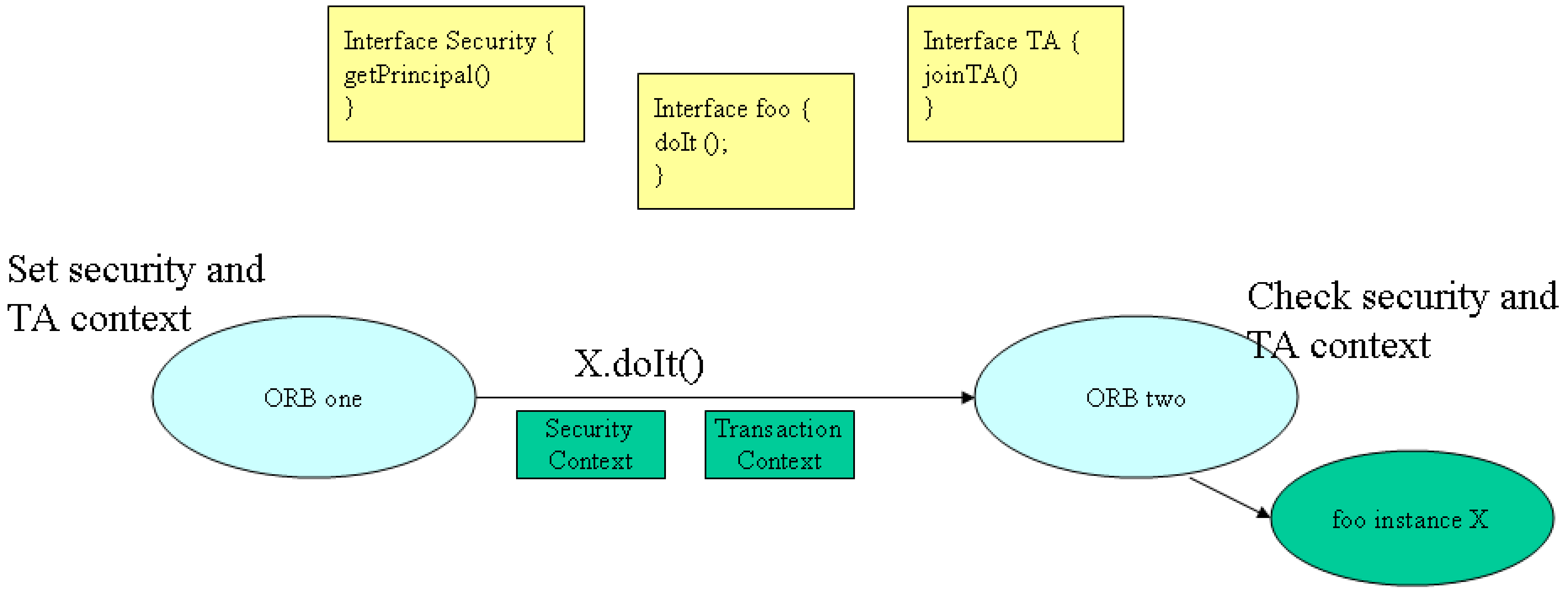
A secure message alone does NOT provide sender authentication. It takes some means to prohibit replay attacks. Digital signatures only are not enough. Secure messages also need to provide support for different key mechanisms, e.g. shared secret key (not contained in message) or random generated symmetric key, encrypted with receivers public key and part of the message. A key reference contained in the message may also be OK.

Problems (3):Interoperability Problems


Secure Messaging
Policies
Policies
Mechanisms: PKI, Kerberos
Mechanisms: PKI, Kerberos
Implementations (Wire Formats)
Implementations (Wire Formats)

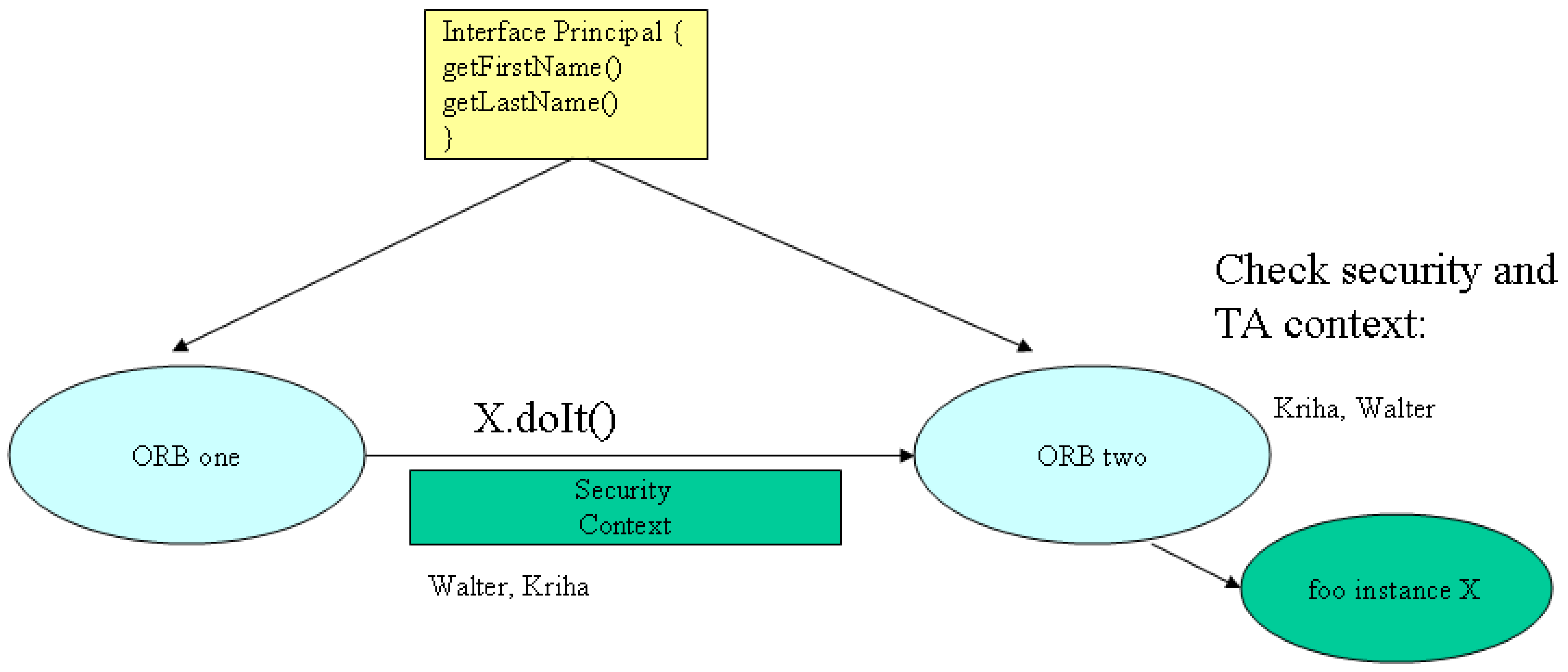
Interoperability requires mutual understanding on SEVERAL layers: Policies must be known in advance and fit mutually. A common mechanism must exist (or a transformation service) and last but not least if security information travels across domain borders (e.g. companies) it must be transported in a format that allows reconstruction on the other side without loss of security context. And last but not least a language must exist that allows properties and attributes to be expressed in a standard way.

Traditional interoperability issues with distributed systems



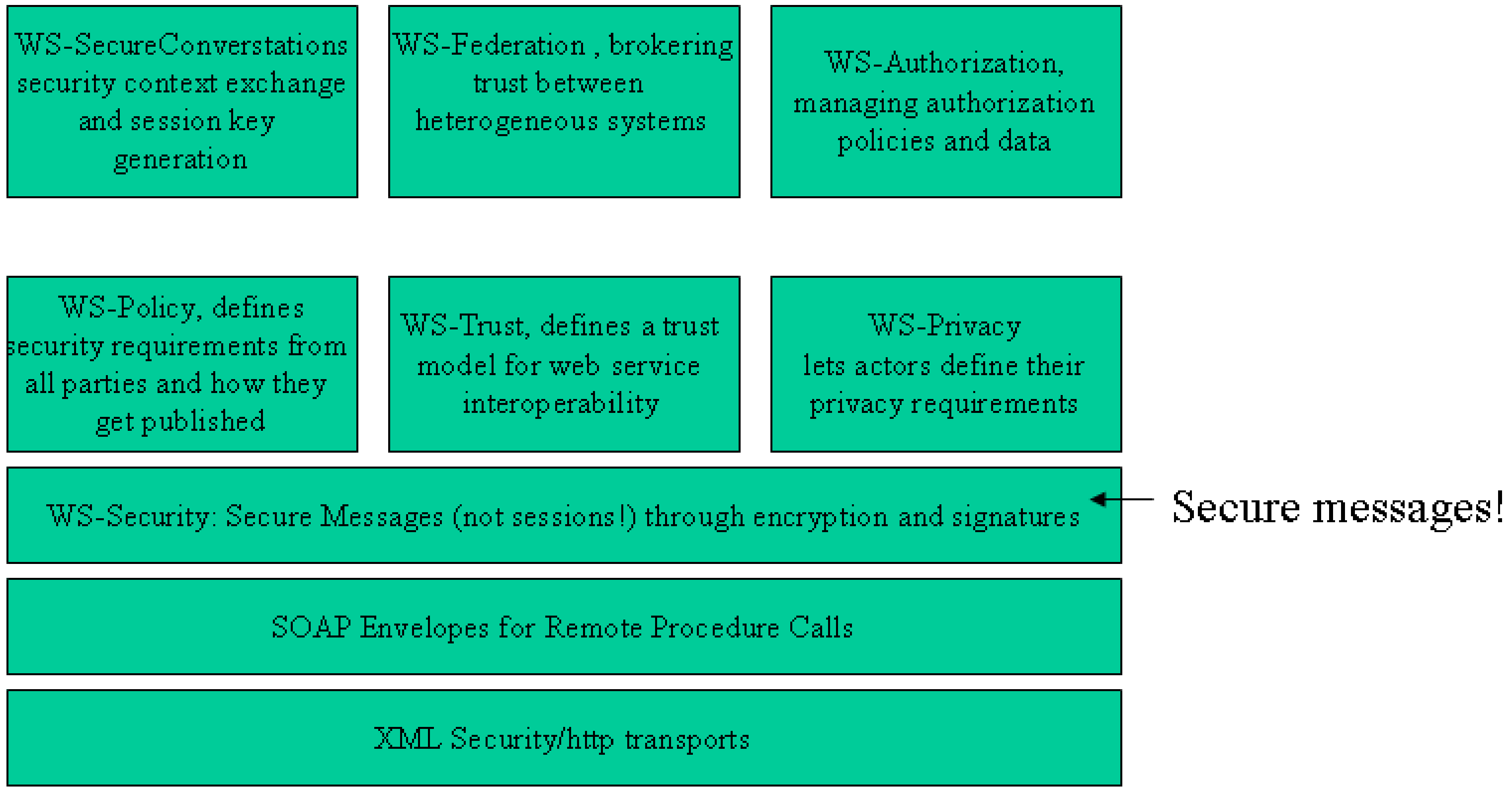
Associated with a remote message call are security and transaction contexts which need to flow to the receiving ORB, e.g. to enable access control. As long as we use only one middleware vendor (e.g. CORBA or EJB vendor) there should be little problem: the security and TA information is encoded binary but both sides use the exact same binary format. No problem. Once we try to couple products from different vendors we will learn the meaning of „Implementation dependent“ and „wire format“. Many times interoperability breaks down because a specification has only defined interfaces and not implementation. This is OK as long we do not cross domains (vendor domains or business domains).

Why Interfaces alone don‘t work across domains



The problem is that even though both ORBs understand the interface they disagree on how the information of the implementation (firstname, lastname) has to be serialized and transmitted. ORB two cannot use ORB ones principal information. Internet protocols have reckognized this and from early on defined WIRE FORMATS for interoperability. The Secure Association Markup Language (SAML) tries to do the same for web services. Now you should understand why it is possible to deploy an EJB in two EJB container but not necessarily that these containers can talk to each other!

Web Services Security Architecture


WS-SecureConverstations security context exchange and session key generation
WS-Federation , brokering trust between heterogeneous systems
WS-Authorization, managing authorization policies and data
WS-Policy, defines security requirements from all parties and how they get published
WS-Trust, defines a trust model for web service interoperability
WS-Privacy lets actors define their privacy requirements
WS-Security: Secure Messages (not sessions!) through encryption and signatures
Secure messages!
SOAP Envelopes for Remote Procedure Calls
XML Security/http transports

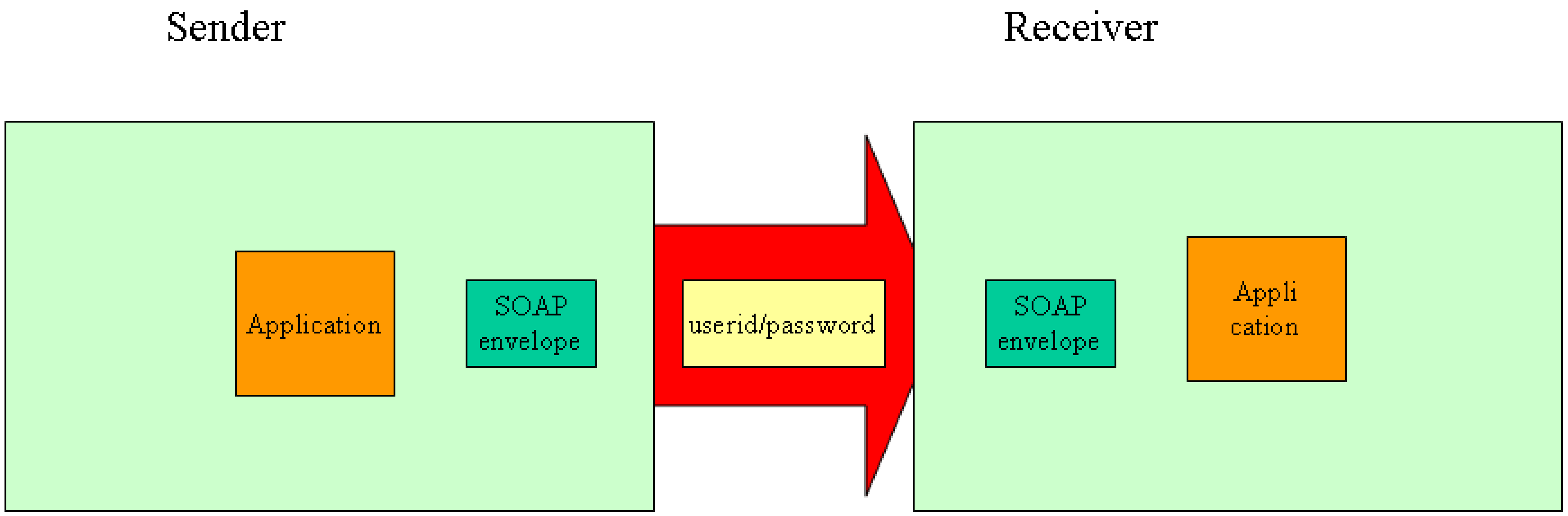
Currently only ws-security has been implemented. The other building blocks are both very ambitious and important for the business model of web services. But they have to solve tough problems like trust federation between domains or security context exchange which goes down to the implementation level as we will see. Just think about the problems to provide a seamless security level across PKI and Kerberos infrastructures.

WS-Security

1. Creates a secure messaging platform with support for message integrity (digital signatures), message confidentiality (encryption) and transport of binary security tokens

WS-Security defines the basic security infrastructure needed to achieve secure (single) messaging. The other parts of Web Services Security will build on this platform. WS-Trust e.g. needs to define how security tokens are acquired and validated.

Using a service on the web (Today)



Today the easiest solution to using a service on the web is to perform some kind of login to establish trust and run the whole session over SSL/TLS.

Problems using a transport-level protocol

There are a number of disadvantages associated with the SSL/TLS solution:

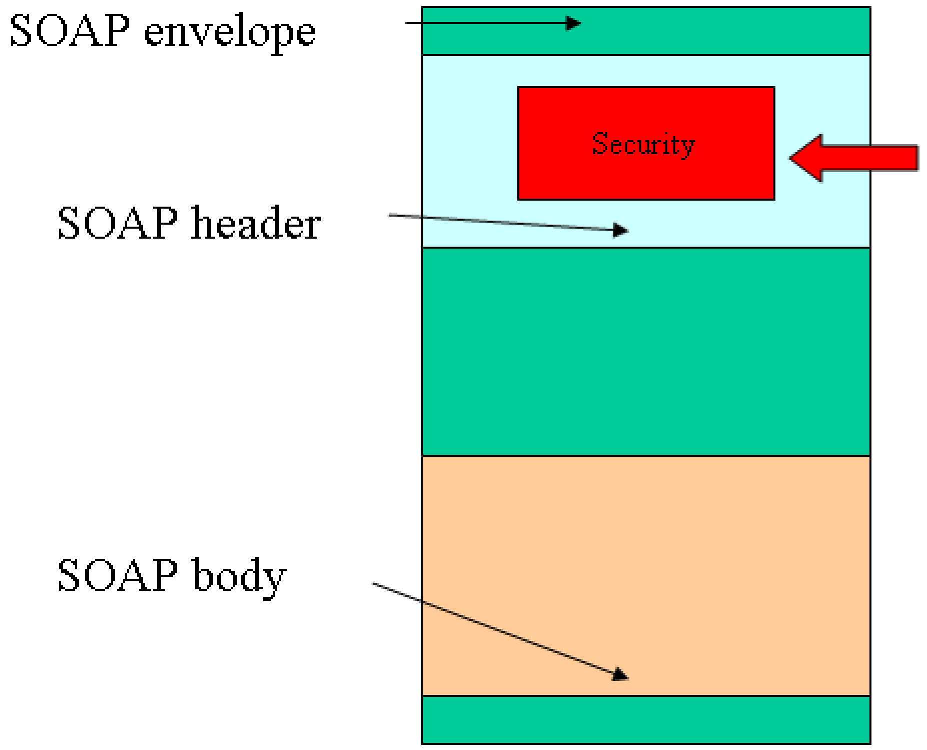
-How is trust established between both applications? A userid/password combination does not only expose the secret but also needs a PRE-ESTABLISHED trust relation between communicating partners.

-How does the sender know that the receiver will not abuse the password?

-How would delegation of the request to different endpoints be handled?

Some of these problems are the result of the web services business model that changes the whole environment in which security has to play: interoperable web services, defined trust relations, delegation of requests to other partners, business processes leaving the company border and cross into other domains via internet.

The SOAP Security Element



<s:Security s:actor=„somebody“ ...>

<s:BinarySecurityToken ValueType=„s:X509v3“
Id= „X509Token“
EncodingType=„s:Base64Binary“>
KLJefj234345lkjafdlök3eq45rkjkl4354545235
</s:BinarySecurityToken>

<xenc:EncryptedKey>.........

<ds:Signature>........

</s:Security>

The Security Element is part of the SOAP header and contains key definition elements from XML encryption and signature elements from XML digital signature. A SOAP specifc element is BinarySecurityToken which contains e.g an X509 certificate encoded in base64. Or UserNameToken which contains user ID and optional encrypted password information. Security elements can be specific for/from certain actors.

UserName Token

```
<Security>

<UsernameToken Id="...">

<Username>...</Username>

<Password
type="...">...</Password>

</UsernameToken>

....
```

This element is part of the security element. WS-Security recommends that it should only be used with a secure transport (channel). Other tokens are of type BinarySecurityToken (e.g. Certificates or Kerberos tickets) and will typically use some form of <encrypted...> element.

Encrypted Keys in WS-Security

```
<wsse:Security>
  <xenc:ReferenceList>
  <xenc:DataReference URI=„#foo“/>
  </xenc:ReferenceList>
<wsse:Security>
.....
<s:Body>
<xenc:EncryptedData Id=„foo“>
 <ds:KeyInfo>
 <ds:KeyName>CN=Walter Kriha, C=DE</ds:KeyName>
 </ds:KeyInfo>
 <xenc:CipherData>
 <xenc:CipherValue>a5e349cddb1243....</xenc:CipherValue>
<xenc:CipherData>
</xenc:EncryptedData></s:Body>
```

```
<wsse:Security>   <xenc:EncryptedKey>
<ds:KeyInfo>......
<xenc:CipherData>
 <xenc:CipherValue>78ef34abc3412....</xenc:CipherValue>
<xenc:CipherData>
  <xenc:ReferenceList>
  <xenc:DataReference URI=„#foo“/>
  </xenc:ReferenceList> <wsse:Security>
.....
<s:Body> <xenc:EncryptedData Id=„foo“>
 <ds:KeyInfo>
 <ds:KeyName>CN=Walter Kriha, C=DE</ds:KeyName>
 </ds:KeyInfo>
 <xenc:CipherData>
 <xenc:CipherValue>a5e349cddb1243....</xenc:CipherValue>
<xenc:CipherData>
</xenc:EncryptedData></s:Body>
```

The message on the left side assumes a shared symmetric key between receiver and sender. Therefore no key is embedded or referenced. Only the key names are associated with the encrypted parts. The right side embeds an encrypted key in the message - probably encrypted using the receivers public key. The key points to the encrypted part.

Mechanisms for single message verification

Document content:

aldflökja jföklj

aöldjfölkasdjföl

ökladjfölasjdf

Digital Signature:

ab348fe89765......

Document-Key validation: Has the document changed after having been signed with the authors private key?. This is done by creating an independent new hashvalue for the document and comparing it with the one in the digital signature.

Key-Identity validation: Was it the key of author X that was used to sign the document? The answer here comes from the authors certificate, e.g. signed by Thawte CA.

Document-Key check for liveness: Was the key revoked? In this case a request against the CAs certificate revocation list would show wheter the key used to sign the document is still valid.

Verifying a signed document includes the three steps from above. (The SAML document on security and privacy describes these steps nicely, cs-sstc-sec-consider-01). We still do not know WHO SENT the document. Author and sender may be different. And we don‘t know for sure WHEN the document was signed.

Mechanisms for single message authentication

-timestamp: Not recommended because of time base attacks (see Tanenbaum or Dollimore, Distributed Systems)

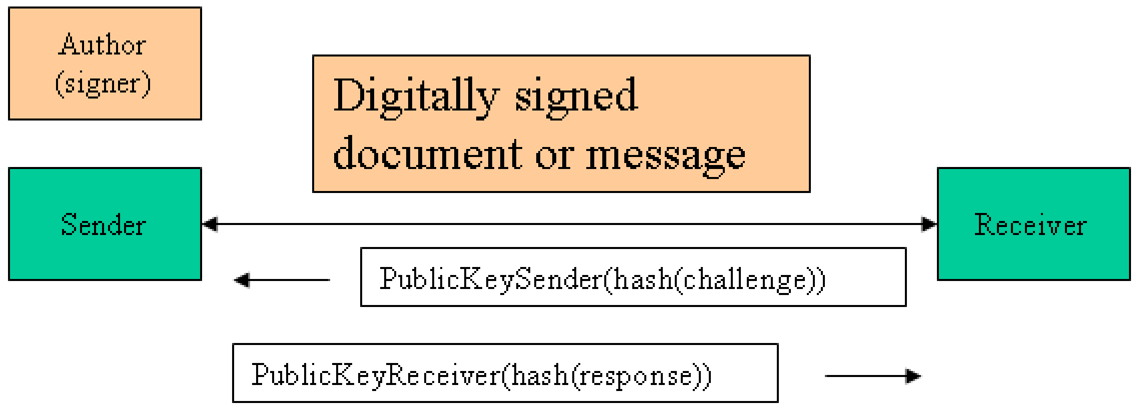
- sequence numbers with random start: OK

- Expirations: Causes the problem of revocation. The longer the expiration date the longer a stolen message can be abused for replay attacks. Same is true for permission tokens.

- Message Correlation

Please not that these mechanisms are NOT enough if the sender identity is in question. In this case the sender needs to prove the possession of the key – e.g. by answering an additional challenge with the proper response generated by her key. But this turns our single message authentication into client authentication as well and requires more messages.

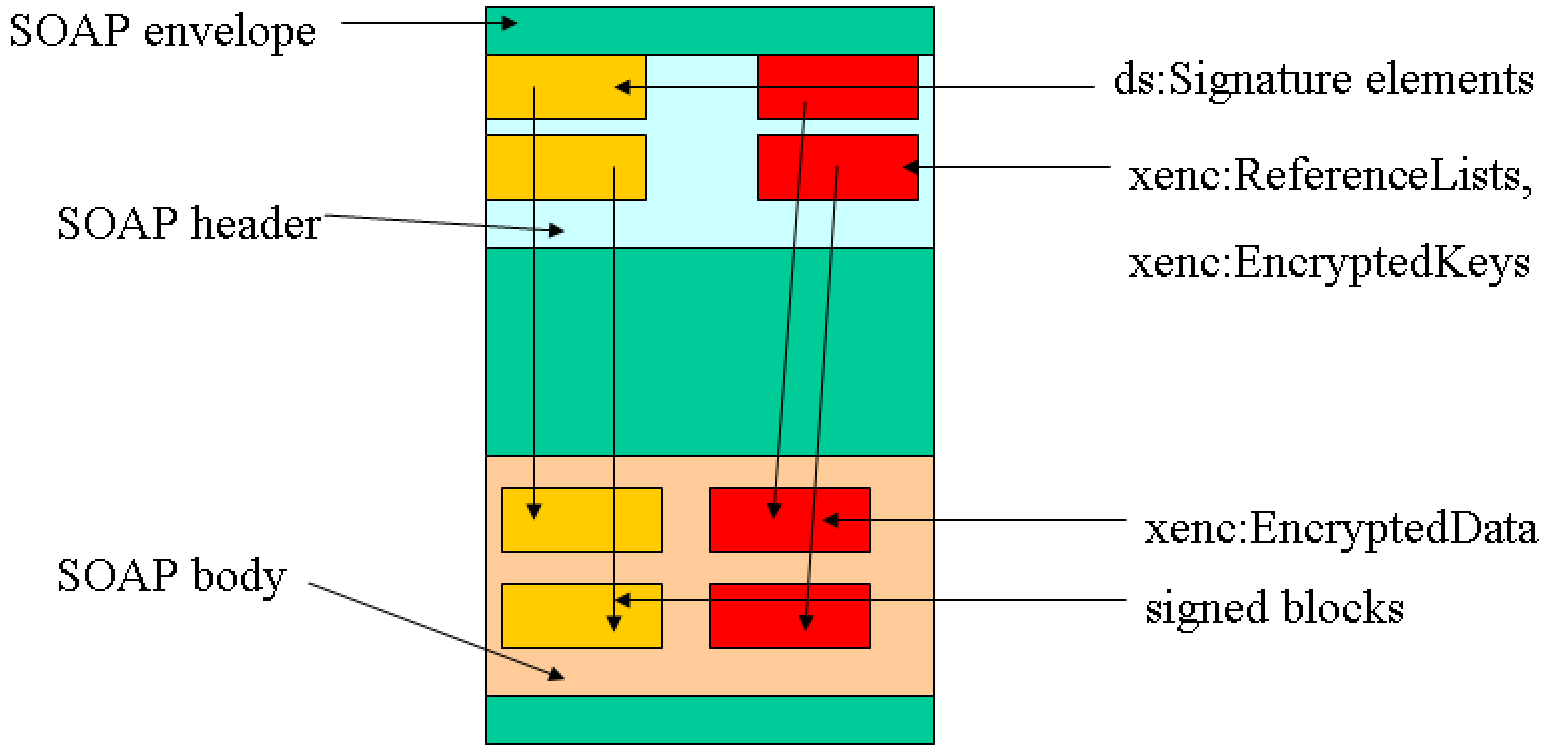
Mechanisms for sender authentication


Author
(signer)
Digitally signed document or message
Sender
Receiver
PublicKeySender(hash(challenge))
PublicKeyReceiver(hash(response))

It is important to realize that sender and author could be different. This can happen both with legal transactions through legal middleman or through replay or man-in-the-middle attacks. Secure messages are susceptible exactly for those two attack forms. The solution could be an validation of sender identity through extra channels (telephone: „did you send me that?“) or by forcing the sender to use SSL with client authentication. In this case the sender is FORCED to use her key to prove in realtime who she is (or better: that the sender has the secret key of a certain person).

If the sender can generate the proper response, the receiver can assume her identity as sender and possible author. Leaving this authentication step opens up the possibility of replay attacks or man in the middle attacks. We will cover this in our Single-Sign-On session.

Using XML DSIG and XML XENC in SOAP



To make DSIG and XENC compatible with SOAP ws-security defines a number of rules, most of them having to do with the fact that Web Services are explicitely designed for use with intermediates. Those intermediates can add signatures or encryption to the SOAP envelop, e.g. to create a chain of trust. The rule here is that new signatures or encryption information is always PREPENDED to already existing information. No encryption of envelope, header or body tag is allowed. Signatures need to respect the right of intermediates to change the envelope or some header information. Again, these restrictions are the results of SOAP processing by intermediates.

New requirements for XML based messages

Please note that we still have other unsolved problems. We can now guarantee integrity and confidentiality of Web Service messages but we still have a number of open issues:

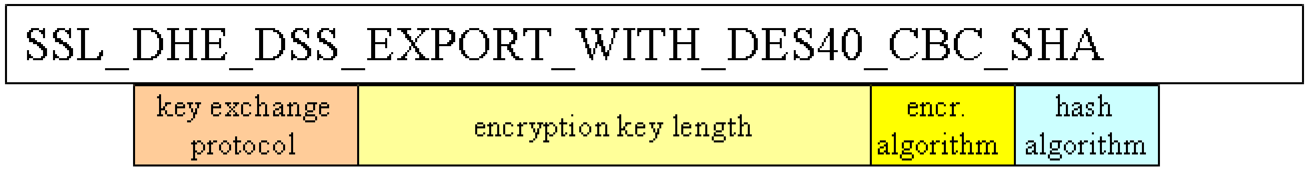
-what kind of security and encryption is required by the provider of a service? (policies)

-How do potential requester know about those requirements? (registries, negotiation)

-How do we establish initial trust? (Security token issuing, validation)

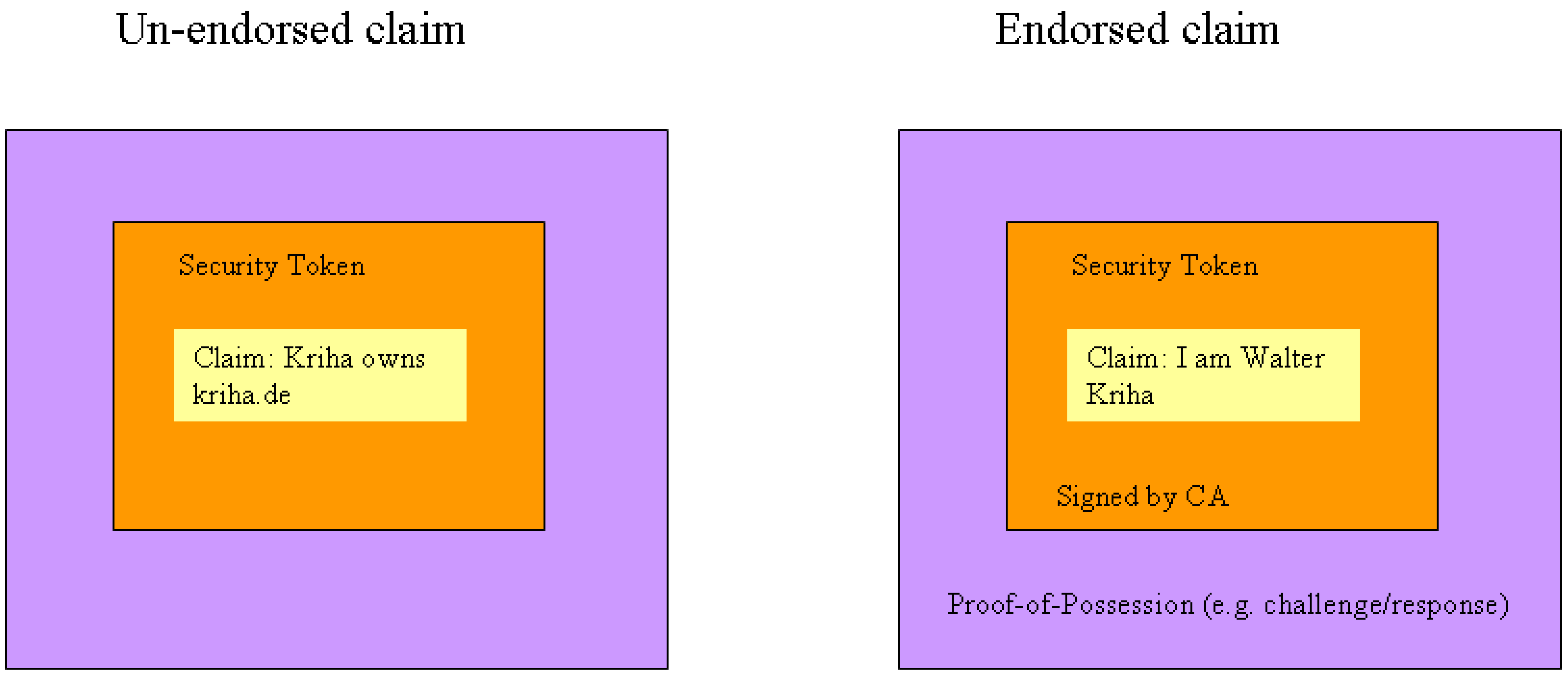
- How do we federate trust across domains?

Example Cypher-Specification



With secure messages sender and receiver cannot agree dynamically on a certain cipher specification. To achieve successful decryption the receiver should be able to advertise her encryption/decryption capabilities so that a sender can chose somethink apropriate for both. For WebServices the service description file WSDL would be such a place

Web Services: Claims, Policies and Security Tokens



A claim is a statement about something. Only authorities can endorse claims. If a receiver trusts an authority then the claim is valid. An endorsed claim like the one about sender identity needs an additional proof of possession process (claim validation) to validate the claim during an authentication process

Claims (Assertions, Statements)

Assertion Element: meta data about the assertion itself (issuer, signature etc.)

Conditions: valid until, valid not before etc.

Advice: additional information provided by issuer

Subject Statement: Information about the subject of the security token, how it can be identified, name etc.

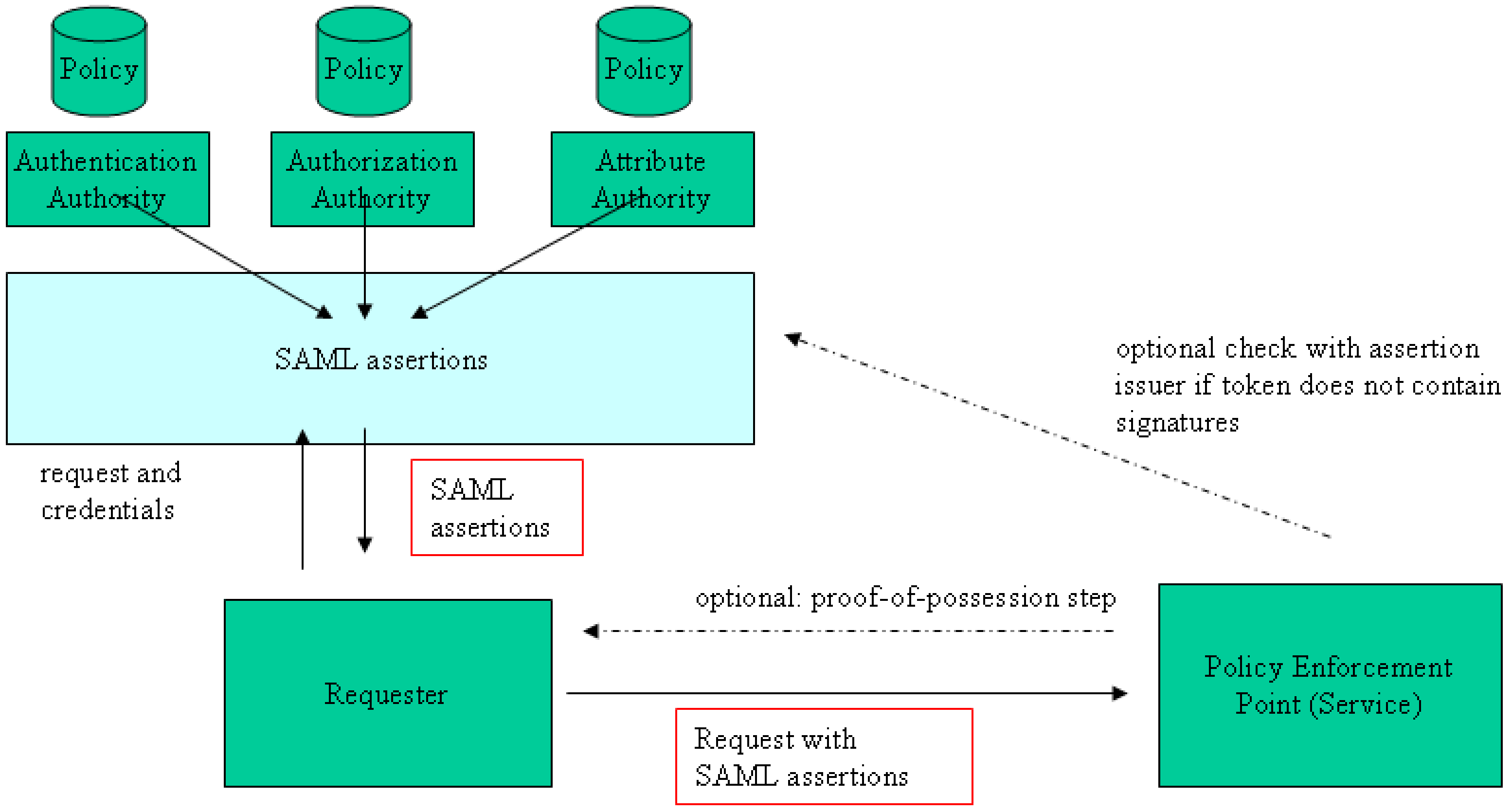
Authentication Statement: issuer has identified the subject at a certain time. Authentication method, location etc.

Authorization Decision Statement: a request by the subjects has been either granted or denied by the issuer. Contains requested resource, action and the evidence that has been provided by the subject (requester)

Attribute Statement: issuer confirms that the subject owns the attributes mentioned in the token.

Examples from the SAML assertion language.

Secure Association Markup Language (SAML)


Policy
Policy
Policy
Authentication Authority
Authorization Authority
Attribute Authority
SAML assertions
optional check with assertion issuer if token does not contain signatures
request and credentials
SAML assertions
optional: proof-of-possession step
Requester
Policy Enforcement Point (Service)
Request with SAML assertions

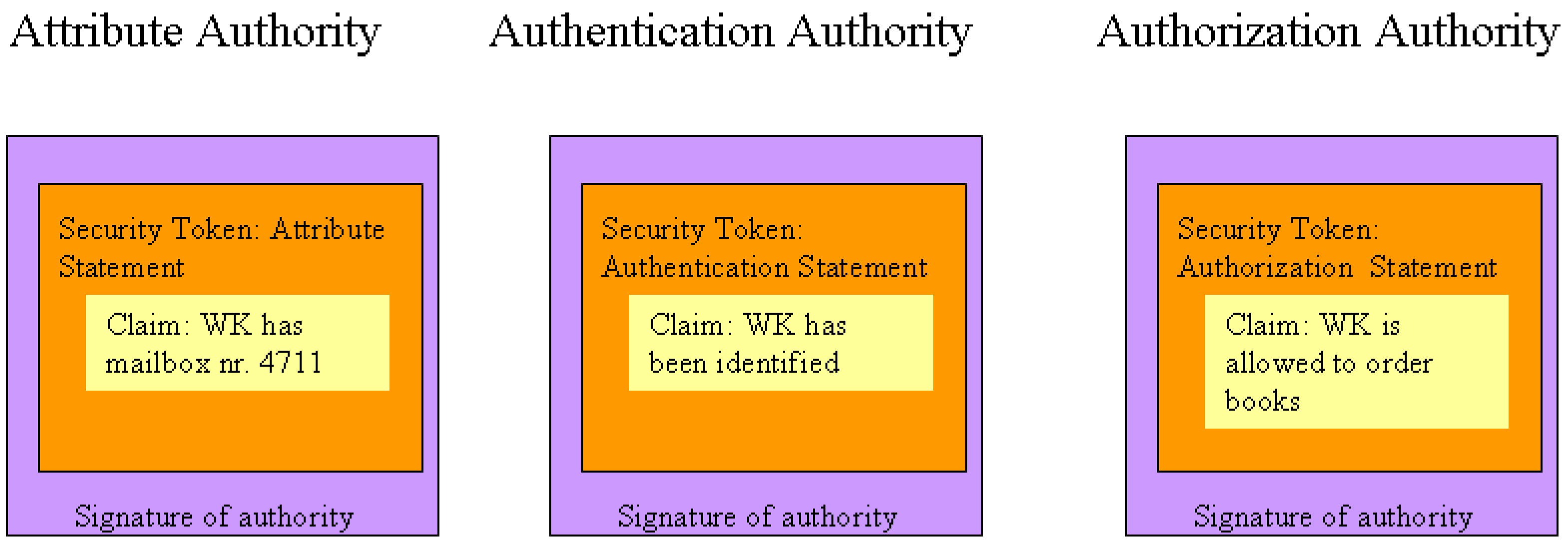
SAML allows to EXTERNALIZE all policies and mechanisms with respect to authentication, authorization and attribute assertion. The access control point needs to check only the assertions but does not have to implement all these mechanisms. On top of this, SAML makes all these statements interchangeable between different services because the format of the assertions is fixed.

Terminology

SAML	WS-Security
Assertion	Claim
Credential	endorsed (signed) security token
Authority	Security Token Issuer

WS-Security introduced a couple of new security terms. The intention was to create a conceptual framework wide enough to cover very different security policies and mechanisms. Unfortunately the terms defined by the Secure Association Markup Language are again different and need to be mapped. For the terms in SAML see: Assertions and Protocol for the OASIS Security Assertion Markup Language, 31 May 2002.

Authorities (Security Token Issuer)


Attribute Authority
Authentication Authority
Authorization Authority
Security Token: Attribute Statement
Claim: WK has mailbox nr. 4711
Signature of authority
Security Token: Authentication Statement
Claim: WK has been identified
Signature of authority
Security Token: Authorization Statement
Claim: WK is allowed to order books
Signature of authority

Any authority can assert claims and package them into a security token. Several claims can be contained in one security token. Web Services generalize that principle into the concept of Security Token Issuer: Web Services which can offer signed security tokens which contain claims. This is the major building block for federation and Single Sign-on in Web Services. Note that authorization tokens externalize previously internal authorization decisions.

Security Tokens

Pros	-Allow excellent delegation of rights -can contain authorization information as well -can be validated without interaction to security token issuer (signatures) -Are perfect support for single message authentication -Are perfect in a loosely-coupled environment when no trust has been established between receiver and endpoint yet.
Cons	-Need to be protected against replay attacks -can be stolen and used -suffer from expiration date and revocation problems -are not sufficient to prove sender identity

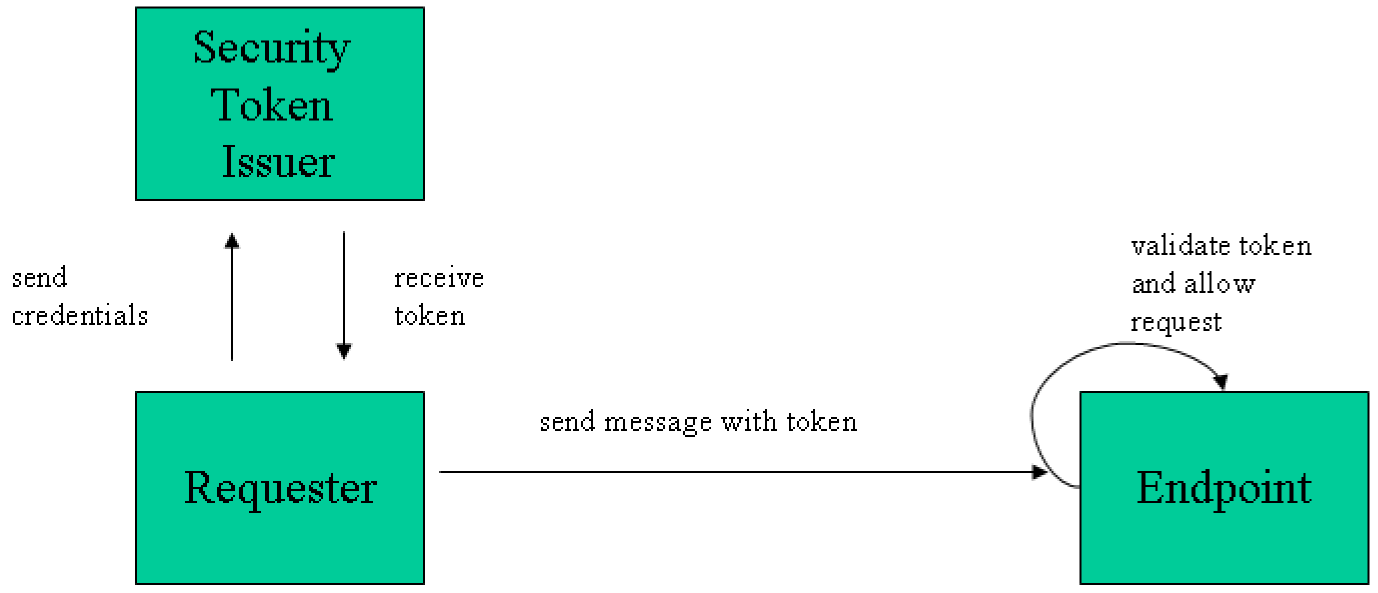
Security tokens are collections of claims. A Signed Security Token is a token that is cryptographically signed by an authority (e.g. X.509 certificate)

WS-Trust

1. Defines how security tokens are requested and obtained from security token issuers.
2. Allow trust engines to verify and broker tokens
3. Differentiates transport level trust mechanisms vs. secure messaging level trust mechanisms

Security Token Issuers (or services) are designed as Web Services themselves. Therefore WS-Trust defines not only XML elements for token request/response and token qualities but also the WSDL messages needed to perform the requests.

Security Token Issuer



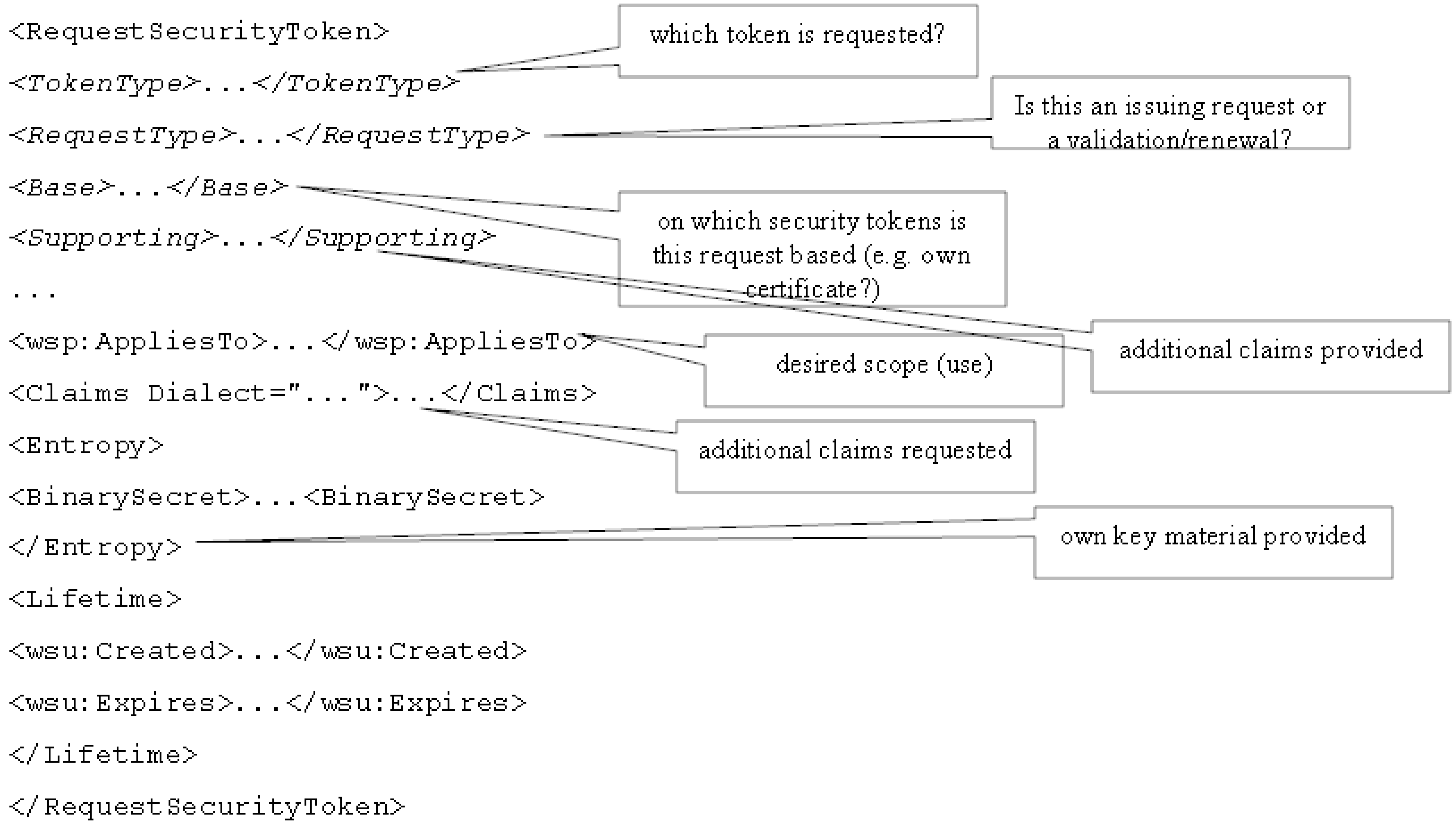
A requester knows from the WSDL description of the web service at endpoint that a certain security token is required to access the service. It contacts the security token issuer who can create such tokens and supplies claims/proof-of-possession tokens. If they validate, the service creates a security token which the requester will use to access the Web Service at endpoint. There are considerable benefits for requester (does give her password only to one actor e.g.) or endpoint (does not need to know about the mechanisms of authentication and authorization used at the security token issuer.

WS-Trust Scenarios

1. Requester wants a token with an embedded session key encrypted for a third party. This allows the third party to use a service on behalf of the requester but does not require the third party to know the requesters key shared with the service.
2. Requester want a token created that ties some attribute to her public key. Her private key serves as a proof-of-possession key later.
3. Web Service endpoint wants a token issuer service to validate a certain token
4. Many more key exchange protocols, renewal requests etc.
5. A token receiver wants to perform an additional challenge/response check for token liveness (freshness)

Usually token requester will be able to treat the returned token as an opaque entity without the need for interpretation. Tokens can be returned either in binary encoding (assuming a secure channel) or in encrypted form.

Security Token Request Element



```
<RequestSecurityToken>
<TokenType>...</TokenType>
<RequestType>...</RequestType>
<Base>...</Base>
<Supporting>...</Supporting>
...
<wsp:AppliesTo>...</wsp:AppliesTo>
<Claims Dialect="...">...</Claims>
<Entropy>
<BinarySecret>...<BinarySecret>
</Entropy>
<Lifetime>
<wsu:Created>...</wsu:Created>
<wsu:Expires>...</wsu:Expires>
</Lifetime>
</RequestSecurityToken>
```

which token is requested?
Is this an issuing request or a validation/renewal?
on which security tokens is this request based (e.g. own certificate?)
additional claims provided
desired scope (use)
additional claims requested
own key material provided

Note: if key material is transported as a „binarySecret“ a confidential channel is required! This seems to be a critical point in WS-Trust which could cause security problems due to programmer errors. More elements are available for delegation, forwarding and to specifiy specific token properties (encryption types etc.)

Security Token Response

```
<RequestSecurityTokenResponse>
<TokenType>...</TokenType>
<RequestedSecurityToken>...</RequestedSecurityToken>
...
<wsp:AppliesTo>...</wsp:AppliesTo>
<RequestedTokenReference>...</RequestedTokenReference>
<RequestedProofToken>...</RequestedProofToken>
<Entropy>
<BinarySecret>...<BinarySecret>
</Entropy>
<Lifetime>...</Lifetime>
</RequestSecurityTokenResponse>
```

which token is provided?

The token provided

desired scope (use)

optional proof of possession token

key material from the token issuer

Note: both, request and response elements are part of the SOAP body element, not the header element which contains digital signature and encryption information for the secure message itself. But security token elements can have references to the security elements in the SOAP header.

WSDL Sample for WS-Trust Requests

```
<wsdl:message name="RequestSecurityTokenMsg">
<wsdl:part name="request" element="wst:RequestSecurityToken" />
</wsdl:message>
<wsdl:message name="RequestSecurityTokenResponseMsg">
<wsdl:part name="response„ element="wst:RequestSecurityTokenResponse" />
</wsdl:message>  .....


<wsdl:portType name="WSSecurityRequestor">
<wsdl:operation name="SecurityTokenResponse">
<wsdl:input message="tns:RequestSecurityTokenResponseMsg"/>
</wsdl:operation>
<wsdl:operation name="Challenge">
<wsdl:input message="tns:RequestSecurityTokenResponseMsg"/>
<wsdl:output message="tns:RequestSecurityTokenResponseMsg"/>
</wsdl:operation>
</wsdl:portType>
```

WSDL creates a hierachy of portType which contain operations, operations contain messages and messages define xml elements which are exchanged.. As shown here this does not include binding information yet which will bind a logical service at a physical location (protocol, address etc.). By keeping logical service descriptions separate, differnent physical bindings can be offered and dynamically negotiated by the client.

WS-Policy

1. The Web Services Policy Framework, or WS-Policy, is a specification that allows a Web service to have a set of rules that must be met, or *consumed*.
2. The goal of WS-Policy: to specify policy information that Web service consumers must adhere to.

From: Tyler Anderson, Understanding Web Services specifications, Part 5: WS-Policy

WS-Policy Terminology

Policy: A list of policy alternatives.
Policy alternative: Contains policy assertions. In normal form, a policy contains a list of policy alternatives specified in wsp:All tags, meaning that a client following the policy can choose to follow either of the available policy alternatives. It can choose which one, but it must adhere to all policies with an alternative. You will see more about how this works later in this section.
Policy assertion: Represents a requirement or capability. For example, a policy assertion could require that a certain type of encryption be used in encrypting transmitted data. In normal form, policy assertions are listed within a policy alternative.
Policy assertion type: A class of policy assertions. For example, <sec:SecurityToken> is an example of a policy assertion type.
Policy expression: The XML representation of a policy, as shown in Listing 1, in normal or compact form.
Policy subject: An entity or object to which a policy can be applied.
Policy scope: The set of objects to which a policy can be applied.
Policy attachment: The way policies are associated with one or more policy scopes.

From: Tyler Anderson, Understanding Web Services specifications, Part 5: WS-Policy

WS-Policy Assertion Types

SignedParts: Specifies which parts of a message, such as header, body, and so on, must be signed.
SignedElements: Specifies which elements within a message must be signed.
EncryptedParts: Same as SignedParts, except encryption is required.
EncryptedElements: Same as SignedElements, except encryption is required.
Token assertions: Several Token assertions are listed in the WS-SecurityPolicy document that define what type of token to include in messages.
Properties: Several properties can be set also, such as encryption algorithms, or a timestamp requirement.

From: Tyler Anderson, Understanding Web Services specifications, Part 5: WS-Policy

WS-Policy Example

```
<wsp:Policy
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:sp="http://schemas.xmlsoap.org/ws/2002/12/secext"
>
...
</sp:SecurityToken>
<sp:UsernameToken
/>
<sp:SignedParts /> <!– all of document must be signed -->
<sp:EncryptedParts> <sp:Body/></sp:EncryptedParts> <!– only body  of document must be encrypted -->

<sp:TransportBinding> <!– messages must have timestamp -->

<sp:IncludeTimeStamp
/>
</sp:TransportBinding>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
```

From: Tyler Anderson, Understanding Web Services specifications, Part 5: WS-Policy

Attacks against WS-Security

XML Level	signatures over plain text, encrypted plain text. Guessing attacks
Message level	Signature misses security tokens used. Token replacement attacks
Key Exchange level (ws-trust)	Same keys used for authentication and message encryption allows key attacks.
Secure Message level	Replay attacks. Message (or message parts) copied and sent again by attacker
Secure Channel confusion	Message uses unencrypted binary key material (password etc.) under the assumption of a secure transport/channel. Channel not secure.
Sender/author confusion	Application or trust engine forgets to check liveness of token. Author is not sender.
Certificate level	Trust engine does not verify validity of certificates

Here we assume that traditional attacks against confidentiality and integrity are prevented by proper use of signatures and encryption. Man-in-the-middle attacks are prevented through mutual authentication and random numbers are generated properly. Applications and users are aware of identity mix-up problems with certificates. For a list of threats and challenges see WS-I Security Scenarios (resources).

A Management View on Web Service Security

„This framework and syntax by itself *does not provide any guarantee of* security.

It is not feasible to provide a comprehensive list of security considerations for such an extensible set of mechanisms. A complete security analysis MUST be conducted on specific solutions based on this specification. „ (from WS-I Security Scenarios)

The second part above sounds almost apologetically. It promises nothing good for implementations using web service security mechanisms. Each one will have to evaluated (signed-off) independently.

A Management View cont‘d

1. WS Security implementations will be complicated and error prone.
2. Therefore only few packages will be available (e.g. WSE from Microsoft)
3. These packages will take a long time to provide complete coverage of WS-XX standards.
4. Security weaknesses contained in those packages will be exploited and affect large numbers of hosts.

After all: business as usual...

Resources (1)

- Murdoch Mactaggart, Enabling XML security – an introduction to XML encryption and XML signature. If you are too lazy to read the original specs from w3c, at least read these 6 pages. Excellent introduction and easy to read too. Shows you with pieces of xml how to sign or encrypt parts of xml documents or messages. Not SOAP related. http://www-106.ibm.com/developerworks/xml/library/s-xmlsec.html/index.html
- An Introduction to XML Digital Signatures, By Ed Simon, Paul Madsen, Carlisle Adams http://www.xml.com/lpt/a/2001/08/08/xmldsig.html . Good and short. Shows the <signature> element and children of it clearly.
- www.w3.org/Signature, www.w3.org/Encryption . Find the latest specifications here.
- Michael Kay, XSLT 2nd edition for a real good introduction to XSLT and extensions.
- Uche Ogbuji, Use XML namespaces with care. Some excellent info on how to use namespaces. Starts with basic principles and explains how namespaces work. Short and good. from developerworks.
- **Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More**
  by S. Weerawarana et.al,

Resources (2)

- Steve DeRose, David Durand, Making Hypermedia work. A good introduction to HyTime, the SGML based hypermedia architecture. If you want to understand what computer science really is about: Naming, addressing, linking, get this book.
- Eliot Kimber, Practical HyTime. Eliot sent this out as a draft but never finished it. VERY good. Explains the concept of an „enabling architecture“ by giving us the logical structures necessary for naming, addressing and linking. If you want to get into Topic maps etc., get these books first. I learnt more from these HyTime books than I did from reading most other computer science literature.
- Paul Prescott on Groves, Property Sets etc. Paul wrote a number of very good articles about the concept of Property Sets. I always wondered how e.g. LDAP models are somehow related to property sets and nodes???

Resources (4)

- If you are not familiar with Web Services yet: go through the slides of my Web Services lecture for distributed systems: http://www.kriha.de/krihaorg/docs/lectures/distributedsystems/webservices/webservices.html
- Eric Rescorla, SSL and TLS, Designing and building secure systems. Chapter 10: SMTP over TLS. Makes the principles of „object based“ security clear. Makes you understand the end-to-end problems of security in a multi-hop environment and shows some solutions. Interestingly, there is NO real solution to secure e-mail over session-oriented SSL/TLS. Best to read before getting into webservices.
- IBM Websphere 4.0 Advanced Edition Security. Chapter 9, Securing Web Services. Get it from www.redbooks.ibm.com . Shows you how web services can be secured in todays J2EE environment.
- M.Hondo, N.Nagaratnam, A.Nadalin, Securing Web Services, IBM Systems Journal Vol.41, No.2, 2002. A very good introduction into the problems of the vision of webservices as interoperable B2B mechanisms true. Shows interoperability problems related to security. Shows explicit vs. implicit security, implementation differences etc. Excellent glossary. A few pages well worth to read. Also covers web services in J2EE environments.

Resources (5)

- Web Services Security (SOAP Message Security), March, 2004, OASIS Standard 200401. Extends SOAP with signatures, encryption and binary security tokens. The basis for most Web Services Security. Download from www.oasis-open.org
- Security in a Web Services World: A proposed architecture and roadmap, IBM and Microsoft joint paper, April 7, 2002, V1. Introduces the web services philosophy with respect to security. Discusses all future standards and has business show cases to demonstrate how it will work. A must read.
- Security Assertion Markup Language (SAML). An OASIS headed specification to achieve interoperable security with XML. Allows single-sign-on. Unclear relation to the new web services specs published by IBM and MS. Very good chapter on security and privacy considerations.
- SAML advances single sign-on prospects, www.fawcette.com. Shows how a SSO can be made with SAML.

Resources (6)

- Securing Web Services with Single Sign-On, Systinet. Nice article showing how a security token issuer service could be used to implement a SSO solution. But leaves the details out (how to secure the token etc.)
- Webservices portals: http://www.webservicessummit.com (contains WS-I security paper etc.)
- WS-I-Security Scenarios, Feb. 14, 2004