T-110.5140 SOAP and UDDI

Tancred Lindholm, Sasu Tarkoma and Pekka Nikander

Aalto University

Lecture outline

- SOAP
  - Document style vs. RPC style SOAP
  - SOAP intermediaries
  - Data encoding in SOAP
- UDDI
  - White, Yellow and Green pages
  - UDDI API

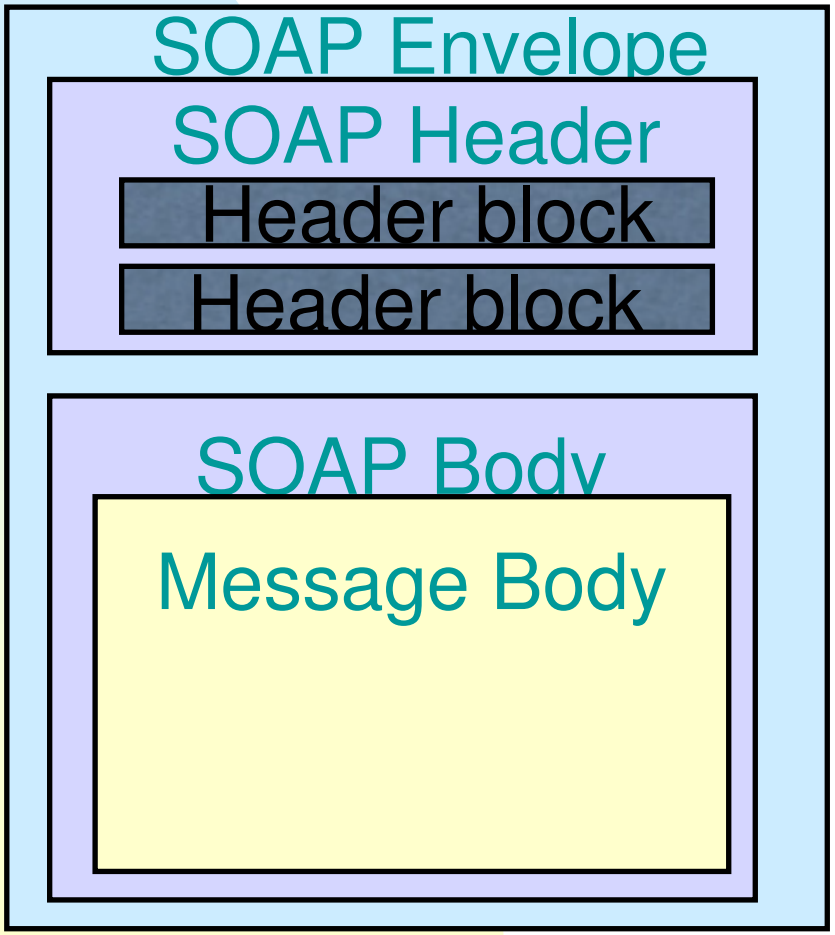
SOAP

- Fundamentally stateless one-way XML message exchange paradigm
  - More complex interactions may be implemented
- Exchange of structured and typed information
  - Using different mediums: HTTP, Email, ..
- Request-reply and one-way communication are supported
- Specified using XML Infoset
  - Note that XML Infoset is an abstract specification
  - On-the-wire representation does not have to be XML 1.0!
- Specifications
  - SOAP Version 1.2 Part 0: Primer
  - SOAP Version 1.2 Part 1: Messaging Framework
  - SOAP Version 1.2 Part 2: Adjuncts
  - SOAP Version 1.2 Specification Assertions and Test Collection

It is necessary to define:

- The type of information to be exchanged
- How to express the information as XML (according to the Infoset)
- How to send the information
- SOAP defines (adjuncts part) these using:
  - Data model
    - Application-defined data structures and values as a directed, edge-labeled graph
  - SOAP encoding
    - Rules for encoding instances of data from SOAP data model to XML
  - One-way and request-reply (RPC) msg exchange
  - Binding framework in order to support concrete messaging protocols and custom on-the-wire representation

SOAP Message Structure

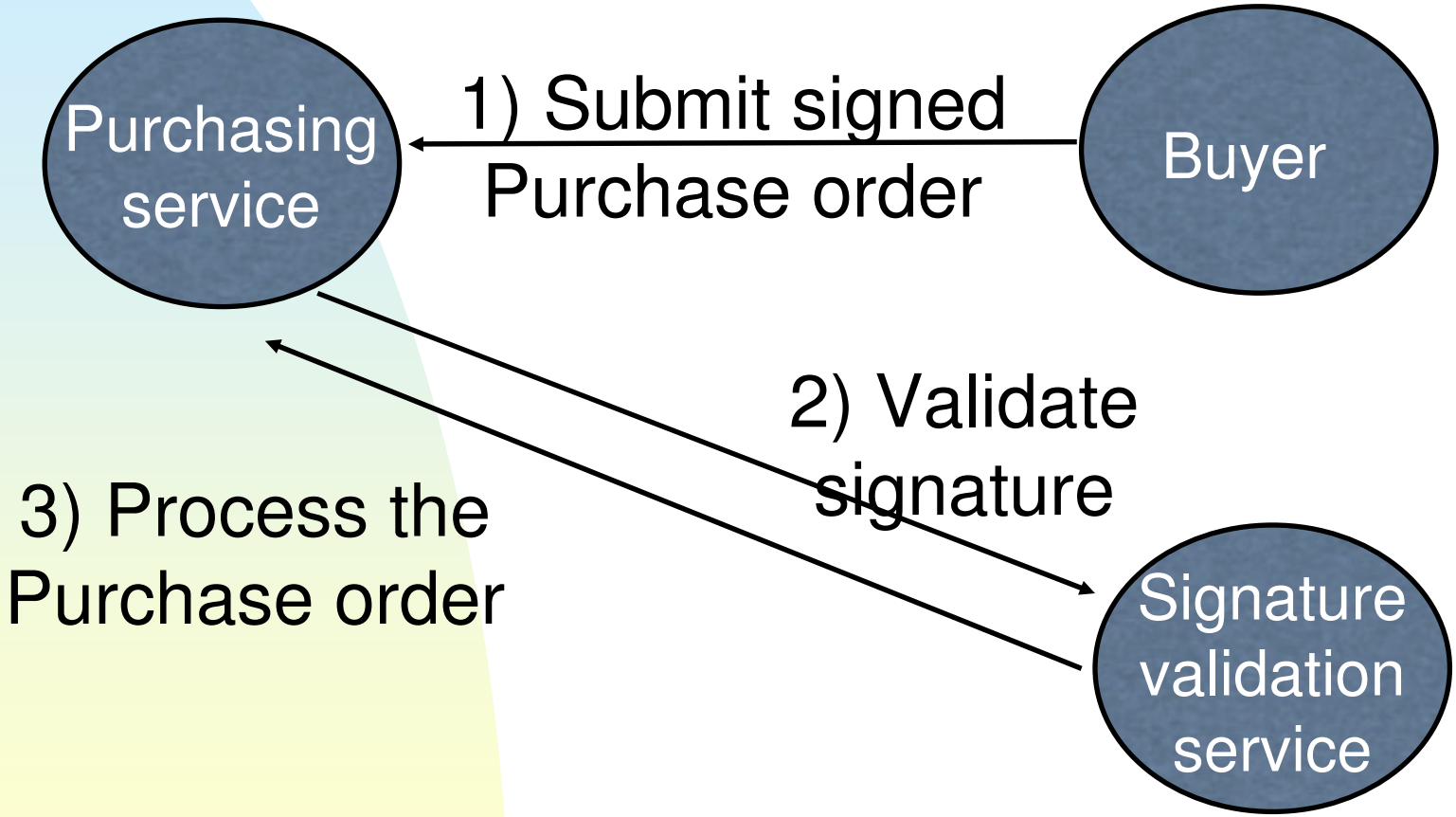


- Optional header contains blocks of information regarding how to process the message:
  - Routing and delivery settings
  - Authentication/authorization assertions
  - Transaction contexts
- Body is a mandatory element and contains the actual message to be delivered and processed (and fault information)

SOAP Message Exchange Model

- SOAP intermediary, or actor, sits between a service consumer and provider and adds value or functionality to the transaction
- The set of intermediaries that the message travels through is called the message path
- No standard way of expressing the message path

Intermediary example



Intermediaries

- SOAP Version 1.2 describes two intermediaries
- A Forwarding Intermediary
  - forwards SOAP messages
  - ”routing” block
  - May not modify content
- An Active Intermediary
  - Additional processing on an incoming SOAP message
  - May modify content in the message
  - encryption, blinding, new header block, timestamping, annotation, ..

The ”role” attribute

- Processing of header block and the body depend on the role(s) assumed by the SOAP node for the message
  - SOAP defines optional env:role attribute that
  - may be present in a header block (a URI)
  - identifies the role played by the intended target of the block
- A SOAP node is required to process the block if it assumes the role identified by the value of the URI
- Three standardized roles:
  - None
    - no SOAP node should process the block
  - Next
    - next node must process block
  - ultimateReceiver
    - implicit if role not specified
- It is up to the node to know its roles, not part of the specification

Header example

```
<m:reservation xmlns:m="http://travelcompany.example.org/reservation"
        env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
         env:mustUnderstand="true">
```

The next SOAP node must process this header block. The block is removed (even if not processed), but may be reinserted. Preserved when ”relay” is used.

SOAP RPC I

- SOAP may be used for both request-reply and one-shot messaging
- Ultimate SOAP receiver is the target of the RPC procedure
- RPC information is carried in the env:Body element and modelled as a struct / array
- Serialization according to the SOAP encoding rules
  - They are optional and there may be several encoding rules for data types

SOAP RPC II

- To make an RPC call the following information is needed:
  - The address of the target SOAP node (ultimate receiver)
  - The procedure name
  - The identities and values of any arguments, output parameters and return value
  - The message exchange pattern
  - Optional data carried in the header blocks
- Service and procedure specification is stored in a WSDL file

SOAP's Data Encoding

- Method of serializing the data intended for packaging in the SOAP message
- Rules outline how basic application data types are to be mapped and encoded into XML
- A simple type system that is a generalization of the common features found in type systems in programming languages, databases, etc.
- SOAP encoding and SOAP RPC representation are optional and not always useful
  - Encoding not needed if the data is already in XML
  - Yet another data model ...
  - RPC representation has some restrictions on data models and encodings for RPC calls

Encoding Styles

- SOAP RPC encoding (rpc/encoded)
  - <soap:Body> contains an element with the name of the method or remote procedure being invoked
  - This element in turn contains an element for each parameter of the procedure
  - SOAP stack handles the complexity
  - Section 5 of the SOAP 1.1 spec, mapping into XML 1.0
  - Developed before schema / WSDL
- SOAP RPC Representation literal encoding (rpc/literal)
  - Suitable for XML data
  - Schema for every parameter type but not for the whole body
- SOAP document-style literal (document/literal)
  - There are no SOAP formatting rules for what the <soap:Body> contains
  - The developer handles everything (using schemas)
  - Easier for the system, easy to validate body
  - Web Service Interoperability (WS-I): Only document/literal allowed!

Purchase order in document/literal-style SOAP

```
<s:Envelope
      xmlns:s=http://www.w3.org/2001/06/soap-envelope>
       <s:Header>
          <m:transaction xmlns:m="soap-transaction"
                  s:mustUnderstand="true">
             <transactionID>1234</transactionID>
          </m:transaction>
       </s:Header>
       <s:Body>
          <n:purchaseOrder xmlns:n="urn:OrderService">
             <from><person>Christopher Robin</person></from>
             <to><person>Pooh Bear</person></to>
             <order><quantity>1</quantity>
                <item>Pooh Stick</item></order>
          </n:purchaseOrder>
       </s:Body>
    </s:Envelope>
```

RPC/encoded-style SOAP Message

```
public Float getQuote(String symbol);

<s:Envelope
 xmlns:s=http://www.w3.org/2001/06/soap-envelope>
  <s:Header>
    <m:transaction xmlns:m="soap-transaction"
            s:mustUnderstand="true">
      <transactionID>1234</transactionID>
    </m:transaction>
  </s:Header>
  <s:Body>
    <n:getQuote xmlns:n="http://example/QuoteService.wsdl">
      <symbol xsi:type="xsd:string">IBM</symbol>
    </n:getQuote>
  </s:Body>
</s:Envelope>
```

SOAP RPC Response

```
<s:Envelope
      xmlns:s=http://www.w3.org/2001/06/soap-envelope>
      <s:Body>
        <n:getQuoteResponse
         xmnls:n="http://example/QuoteService.wsdl">
          <value xsi:type="xsd:float">
            98.06
          </value>
        </n:getQuoteResponse>
      </s:Body>
    </s:Envelope>
```

UDDI

- Universal Description Discovery and Integration
- A “meta service” for locating web services by enabling robust queries against rich metadata
- Distributed registry of businesses and their service descriptions implemented in a common XML format

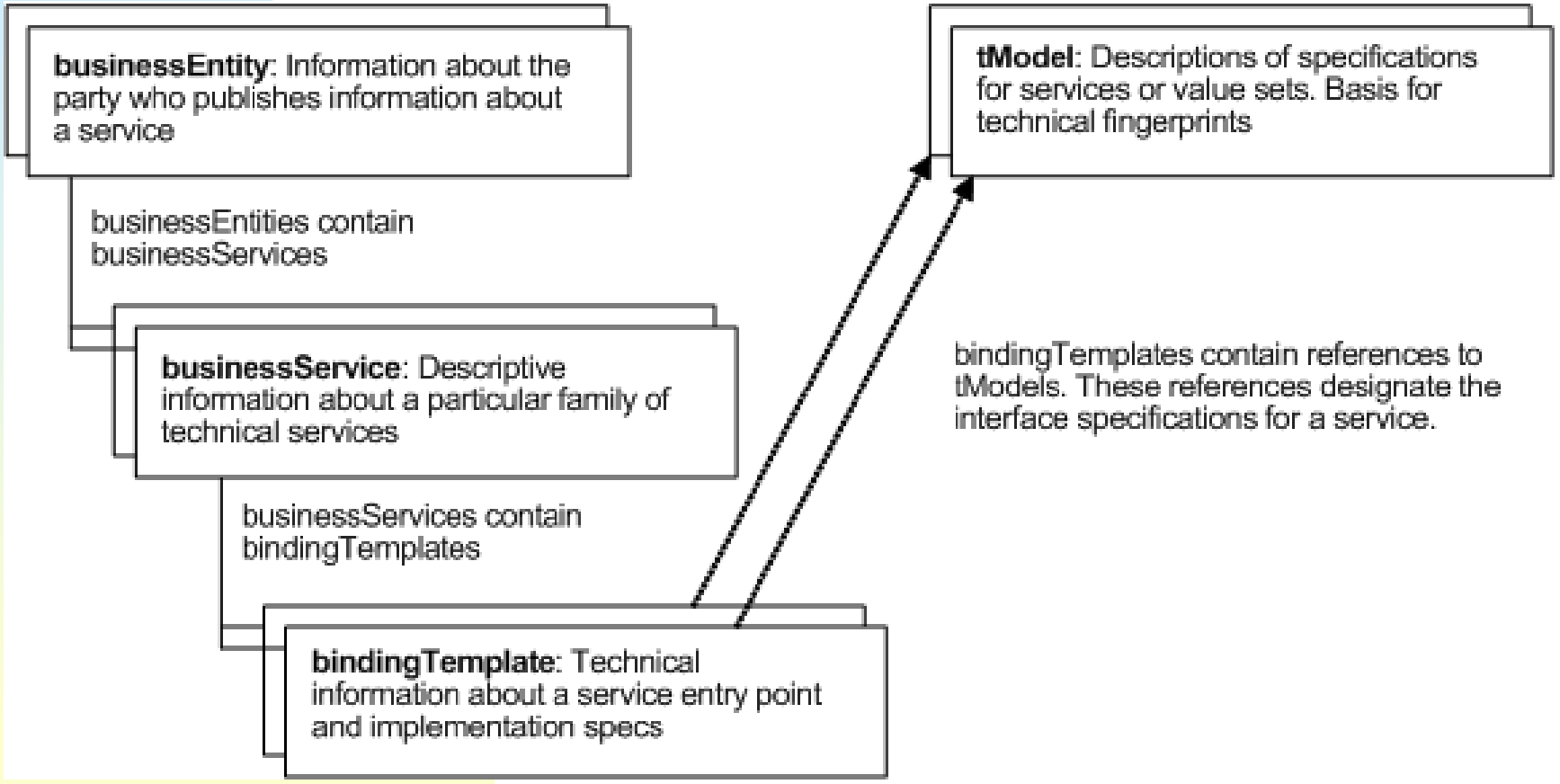
Web Service Challenges

- Who provides web services?
- How are they implemented?
- Where are they provided?
- What is their behavior?
- Is an application compatible?
- Searching and indexing do not work today
  - how to find the right services at the right time?
  - we need solutions designed for tools and apps

What is UDDI?

- Universal Description Discovery and Integration
- Industry-wide initiative supporting web services
- Specifications
  - Schemas for service description
  - Schemas for business (service implementers) description
  - Developed on industry standards
    - Applies equally to XML and non-XML web services
- With UDDI a programmer or a program can locate
  - Information about services exposed by a partner
  - **find compatible in-house services**
  - Find links to specifications of a Web service
  - Maintain technical compatibility by automatically configuring certain technical connections

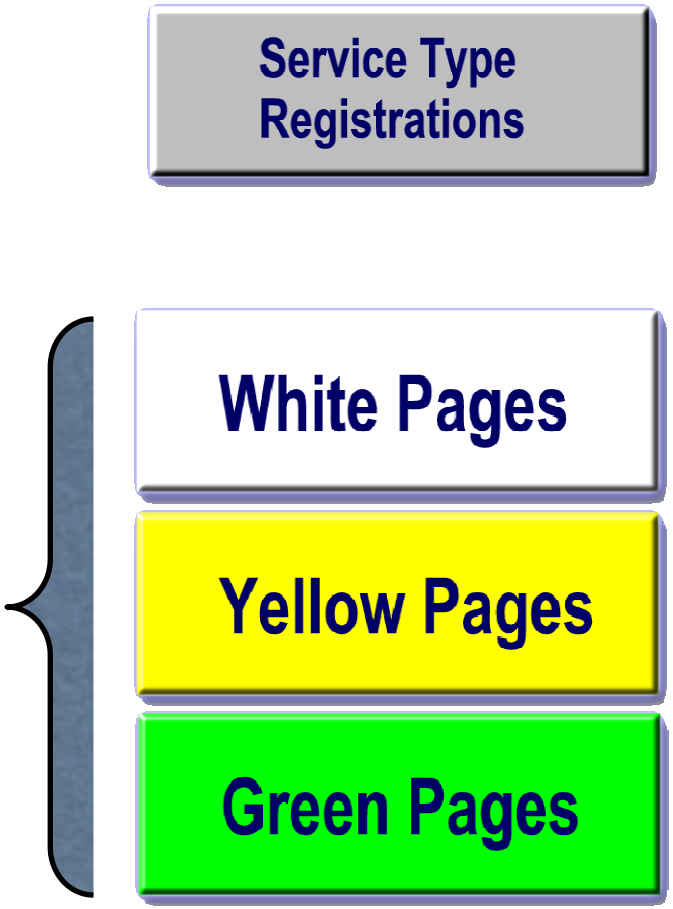
The four core types of data structures that are specified by the UDDI API Schema and their relationships are shown here



UDDI Registry Entries

Standards Bodies, Agencies, Programmers, Publishers register specifications for their Service Types

Service providers register precise information about themselves and their Web services



White pages

- Business name
- General business description
  - Any number of languages
- Contact info
  - Names, phone numbers, fax numbers, web sites, etc.
- Known identifiers
  - List of unique identifiers for a business

Yellow pages

- Business categories
  - Based on standard taxonomies
  - 3 base taxonomies in V1
- Taxonomies
  - Industry: NAICS (Industry codes - US Govt.)
  - Product/Services: UNSPSC (ECCMA)
  - Location: Geographical taxonomy (ISO 3166)
  - …easy extension in upcoming releases
- Realized using name-value pairs, any valid taxonomy identifier can be attached to the business white page

Green pages

- New set of information businesses use to describe how to “do e-commerce” with them
  - References to specifications for Web Services
- Business process (functional)
- Service specifications (technical)
  - Programming/platform/implementation agnostic
- Binding information (implementation)

UDDI tModels

- With tModels you attach an identifier to
  - Technical interfaces / standards
  - Arbitrary classification schemes
- Identifiers then used in service descr.

UDDI tModels Example

```
<tModel xmlns="urn:uddi-org:api" tModelKey="UUID:AAAAAAAA-">
 <description xml:lang="en">Check limit reporter</description>
 <overviewURL>http://schema.com/creditcheck.wsdl</overviewURL>
 <categoryBag>
  <keyedReference
   tModelKey="UUID:CD153257-086A-4237-B336-6BDCBDCC6635"
   keyName="Consumer credit gathering or reporting services"
   keyValue="84.14.16.01.00"/>
  <keyedReference
   tModelKey="UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4"
   keyName="types"
   keyValue="wsdlSpec"/>
 </categoryBag>
</tModel>
```

```
<businessService  businessKey="BB"  serviceKey="CC">
  <name>HPCU Credit Check</name>
  <bindingTemplates>
   <bindingTemplate serviceKey="CC" bindingKey="DD">
    <accessPoint URLType="https">
            https://hpcu.com/creditcheck</accessPoint>
    <tModelInstanceDetails>
     <tModelInstanceInfo tModelKey="UUID:AAAAAAAA"/>
    <tModelInstanceDetails>
   </bindingTemplate>
  </bindingTemplates
 </businessService>
```

Registry APIs

- Inquiry API
  - find_business, find_service, find_binding, find_tModel
  - get_businessDetail, get_serviceDetail, get_bindingDetail, get_tModelDetail
- Publisher's API
  - save_business, save_service, save_binding, save_tModel
  - delete_business, delete_service, delete_binding, delete_tModel
- Security
  - get_authToken, discard_authToken

T-110.5140 Network Application Frameworks

XML Security Basics

1.3.2010

Tancred Lindholm, Sasu Tarkoma, Pekka Nikander

Contents

- Basic XML security
- High-level view to WS security
- Standardization
- Summary

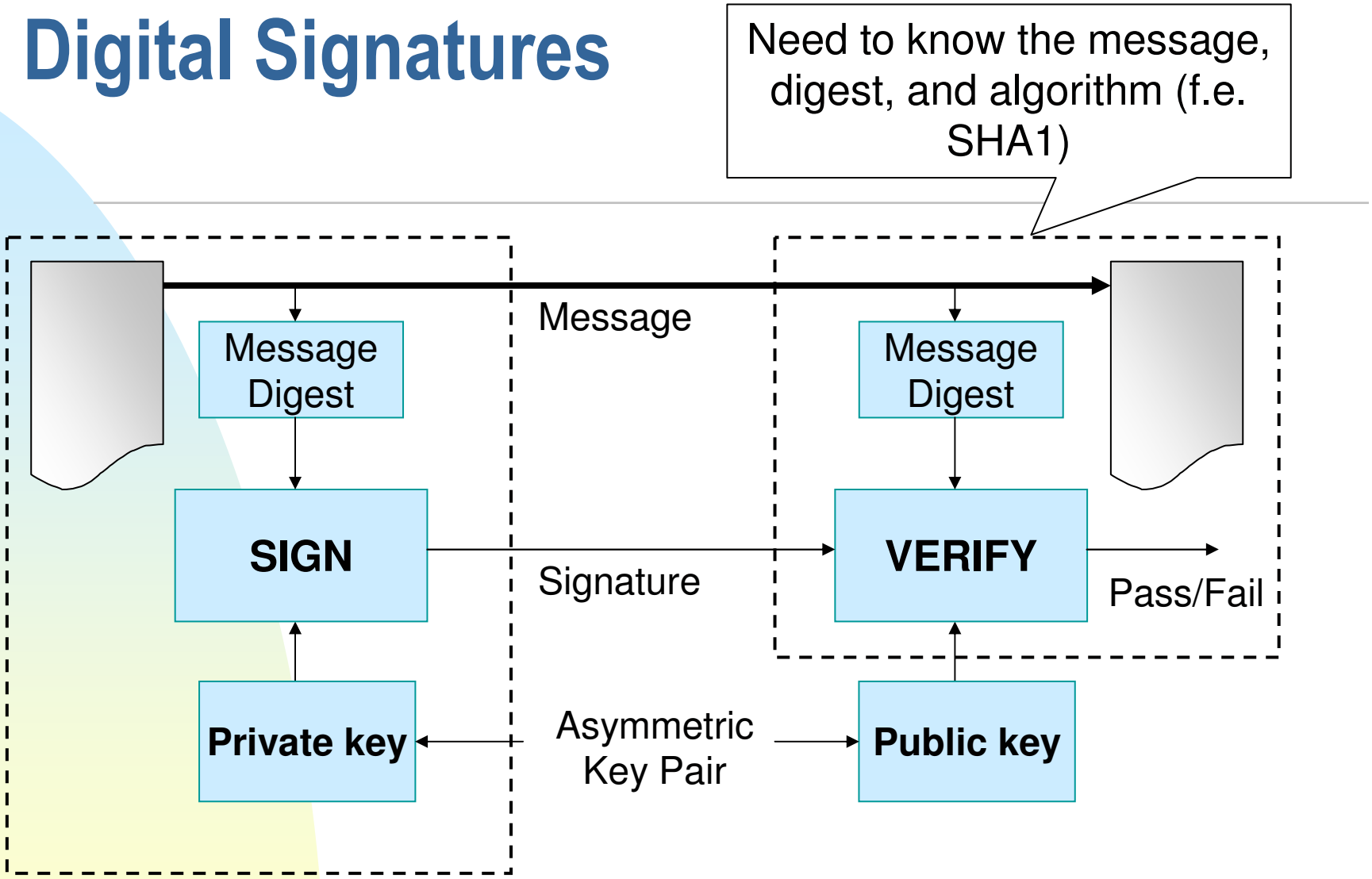
Need for XML security

- XML document can be encrypted using SSL or IPSec
  - this cannot handle the different parts of the document
  - documents may be routed hop-by-hop
  - different entities must process different parts of the document
- SSL/TLS/IPSec provide message integrity and privacy only when the message is in transit
- We also need to encrypt and authenticate the document in arbitrary sequences and to involve multiple parties

Basic XML Security

- XML Digital Signatures (XMLDSIG)
- XML Canonicalization

Digital Signatures



A word about digests & canonicalization

- Secure Message Digest = binary value that depends on **all** input bits in a non-reversible manner
- Ex 1 digest('Hello world') = c0fe
- Ex 2 digest('Hello world') = beef
- The different values for ex1 and ex 2 above are sometimes not what is desired
- Want same digest for "same" data → "Same" data must have exact same bits!
- Solution: canonicalize to standard syntax (e.g. n * space = 1 space above)

XML Digital Signatures

- Digests calculated and a <Reference> created
- Then a <Signature> element created from <Reference>, keying information, signature algorithm, and value
  - The signature is actually calculated over the SignedInfo subset of this information

XML Digital Signatures

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

Detached signature of the content of the HTML 4 in XML

Canonicalization method:
etc. Applied to
edInfo

Signature algorithm: DSA

Reference to HTML 4 XML spec (detached)

Digest value calculated over the identified data after transformations

+ digest + encrypt. For

KeyInfo indicates the key to be used to validate the signature

gned!
validation of the
and validation
gest within

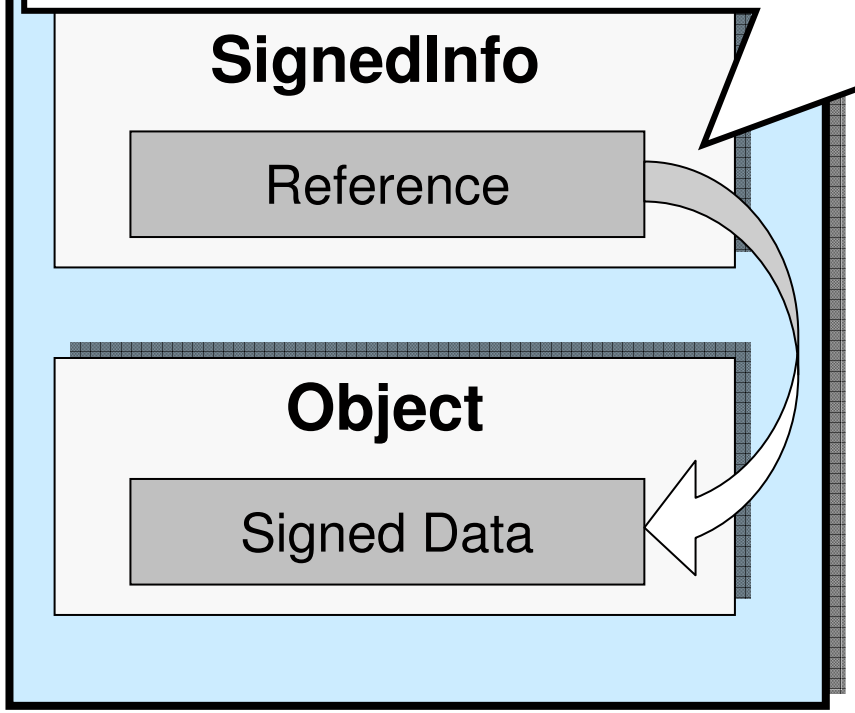
[s01] <
[s02] <
[s03] <
[s04] <Signat
[s05] <
[s06] <
[s07] <
[s08] <
[s09] <DigestM
[s10] <DigestVa
[s11] </Referenc
[s12] </SignedInfo>
[s13] <SignatureV
[s14] <KeyInfo>
[s15a] <KeyValue>
[s15b] <DSAKeyValue>
[s15c] <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
[s15d] </DSAKeyValue>
[s15e] </KeyValue>
[s16] </KeyInfo>

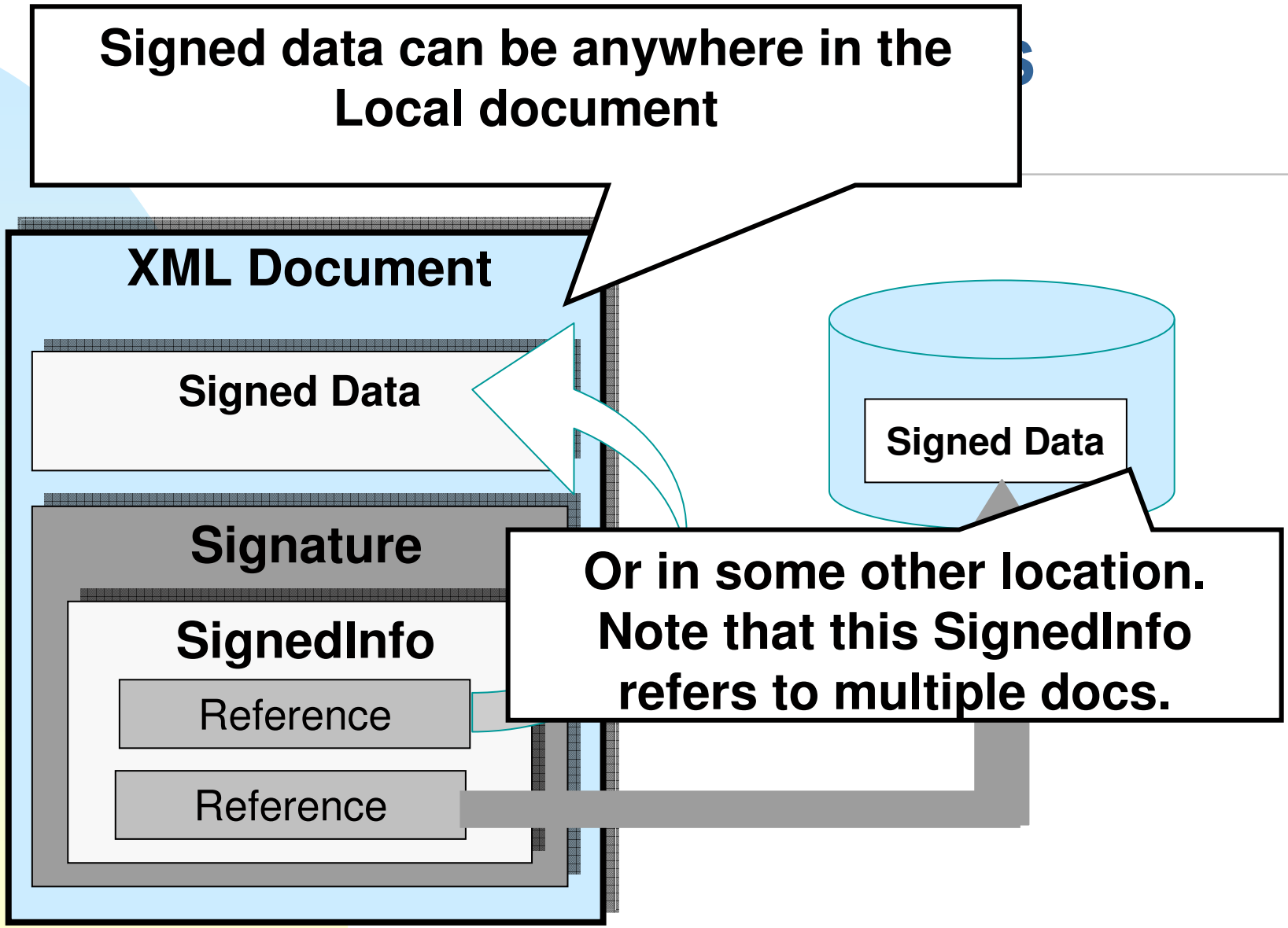
ig#">
w.w3.org/TR/2001/REC-xml-c14n-200
Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
EC-xhtml1-20000126/">

XML Digital Signatures (cont.)

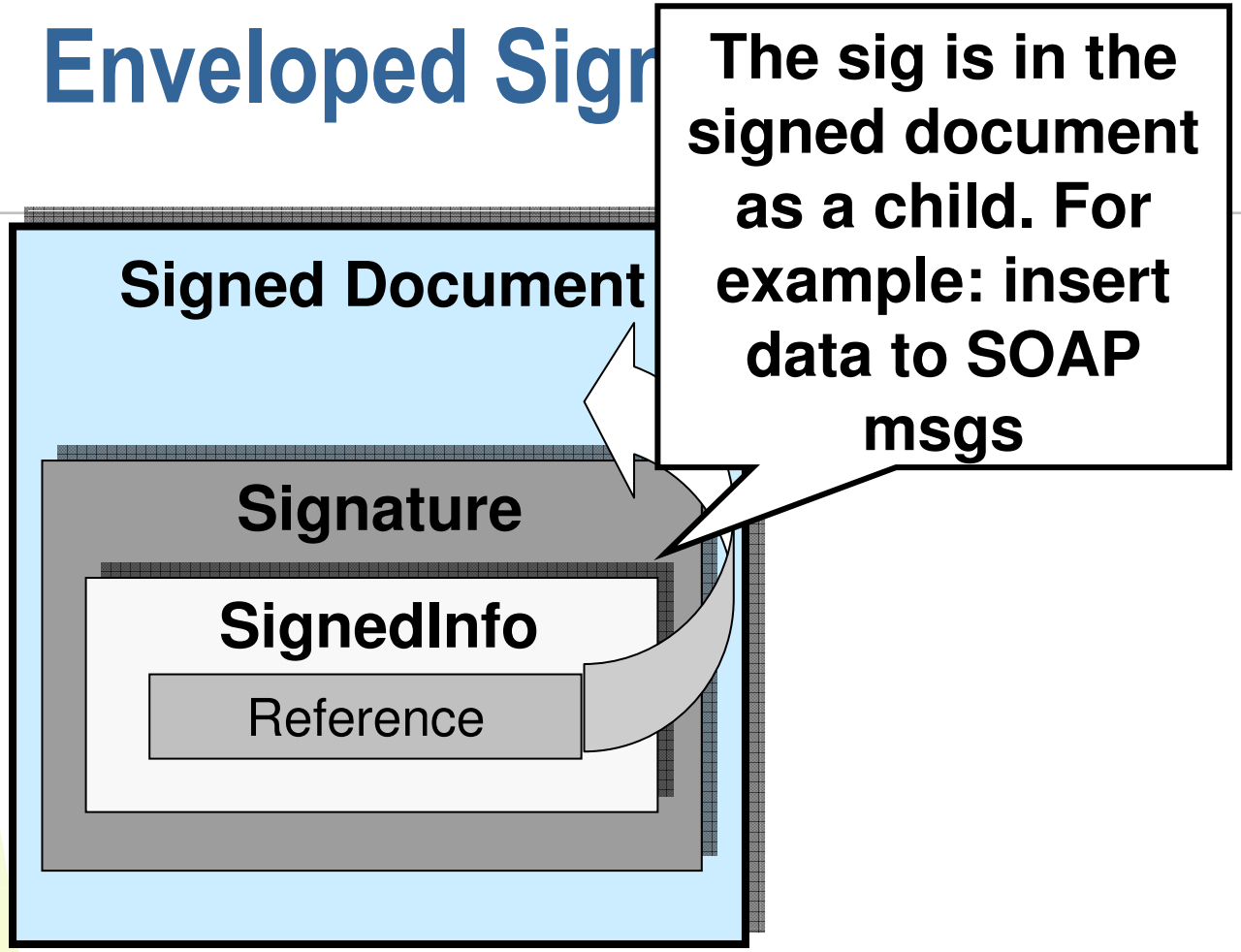
- The data being signed can be inside the <Signature>, within an <Object> element (enveloping), or
- external to the <Signature> in the same document or elsewhere (detached), or
- surrounding the <Signature> (enveloped), or
- any combination of these.

SignedInfo refers to object (sig is parent), object digested & thus in SignatureValue. Can be useful for SOAP messages




Signed data can be anywhere in the Local document
XML Document
Signed Data
Signature
SignedInfo
Reference
Reference
Signed Data
Or in some other location. Note that this SignedInfo refers to multiple docs.

Enveloped Sign



XML Signatures (cont.)

- To verify an XML digital signature
  - Verify the digests in each Reference, and
  - Verify the signature value over the SignedInfo with the appropriate key and given signature algorithm

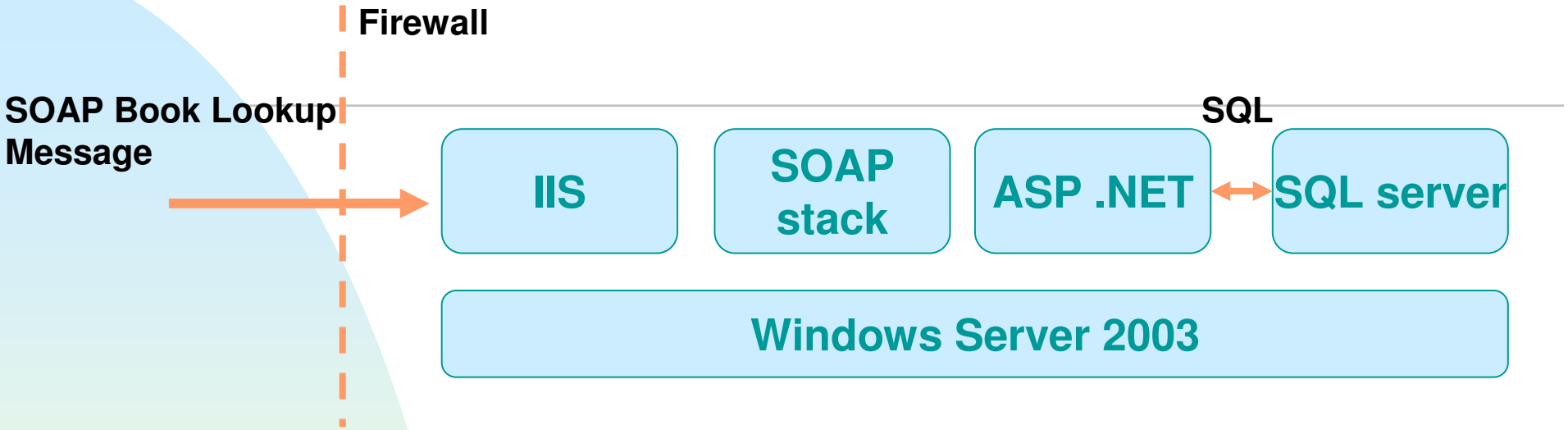
What about <Transforms>?

- A way to specify a sequence of algorithmic processing steps to apply
  - to the results retrieved from a URI to
  - Produce the data to be signed, verified, or decrypted.
  - Can include compression, encoding, subset extraction, etc. For example using XPath
  - Not needed in simple cases, but essential in complex cases

High-level view to WS security

- Security is as strong as the weakest link
- The options for an attacker are:
  - Attack the Web Service directly
    - Using ”unexpected” XML
  - Attack the Web Services platform
  - Attack a WS security tool
  - Attack the underlying operating system or network connection

Example – SQL Injection




```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=”..”>
<SOAP-ENV:Header><SOAP-ENV:Header>
<SOAP-ENV:Body>
<BookLookup:searchByISBN xmlns:Booklookup=”..”>
<BookLookup:ISBN>1234567810</BookLookup:ISBN>
</BookLookup:searchByISBN>
</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

VB.NET code:

```
Set myRecordset = myConnection.execute(”SELECT * FROM myBooksTable
WHERE  ISBN=”” & ISBN_Element_Text & ””)
```

Becomes

```
SELECT * FROM myBooksTable WHERE ISBN = ’1234567810’
```

Attack: SQL Injection


```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=”..”>
<SOAP-ENV:Header><SOAP-ENV:Header>
<SOAP-ENV:Body>
<BookLookup:searchByISBN xmlns:Booklookup=”..”>
<BookLookup:ISBN>’; exec master..xp_cmdshell ’net user Joe pass /ADD’;--
</BookLookup:ISBN></BookLookup:searchByISBN>
</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

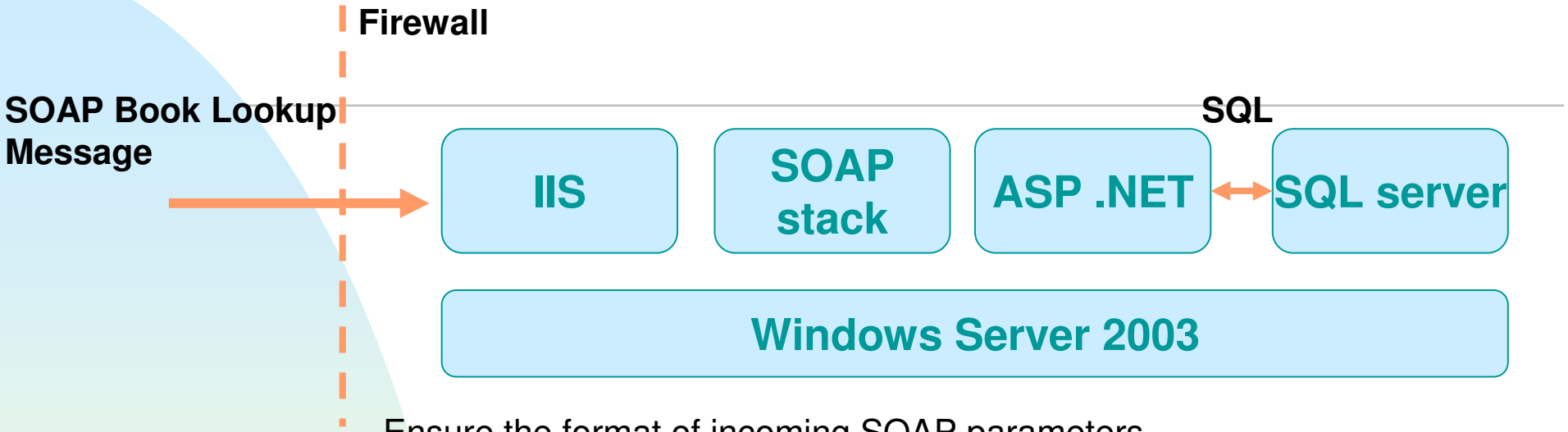
VB.NET code:

```
Set myRecordset = myConnection.execute(”SELECT * FROM myBooksTable
WHERE  ISBN=”” & ISBN_Element_Text & ””)
```

Becomes

```
SELECT * FROM myBooksTable WHERE ISBN = ’’; exec master..xp_cmdshell ’net
user Joe pass /ADD ’;—
```

Solution



Ensure the format of incoming SOAP parameters

<simpleType name="isbn"><restrictions base="string"><pattern value="[0-9]{10}"/></restriction></simpleType>

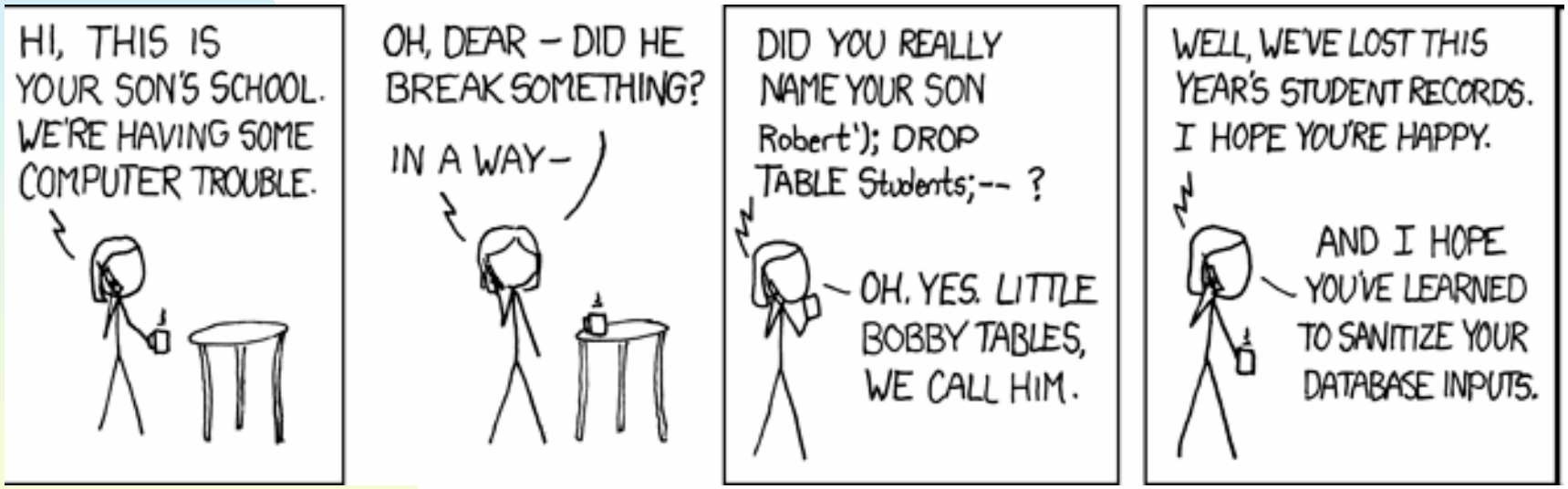
Validate this Schema against the data isolated by the following XPath expression:

/Body/BookLookup:searchByISBN/BookLookup:ISBN

1234567810 passes

'exec master..xp_cmdshell 'net user Joe pass /ADD'-- fails

So now you get xkcd #327...



XML & WS Security Standardization

- Core specification: XML Signature
- WS-Security
  - SOAP with security tokens
    - A security token represents a set of claims.
    - Self-generated or issued by a trusted party
  - Relies on XML Signature & Encryption
- SAML (Security Assertion Markup Language)
  - A XML-based framework (schemas) for the exchange of authentication and authorization information
  - Mainly for integration, up to relying parties to decide to what authentication authority to trust
  - Assertions can convey information about authentication acts performed by subjects, attributes of subjects, and authorization decisions about whether subjects are allowed to access certain resources
  - Authentication statements merely describe acts of authentication that happened previously
- SAML & WS-Security allow a SOAP message to include information about the end-user's authentication status

Who are specifying the standards?

- Joint IETF/W3C
  - XML Signature (www.w3.org/Signature)
- W3C
  - XML Encryption (www.w3.org/Encryption/2001)
  - XML Key Management (XKMS) (www.w3.org/2001/XKMS)
- OASIS
  - WS-Security
    - SOAP Message Security specification etc.
  - SAML: Security Assertion Markup Language
  - XACML: Extensible Access Control Markup language
- Web Services Interoperability Organization (WS-I)
  - Basic security