Secure Web Services

Geoffrey Fox, Marlon Pierce
Community Grids Lab
Indiana University

Introduction

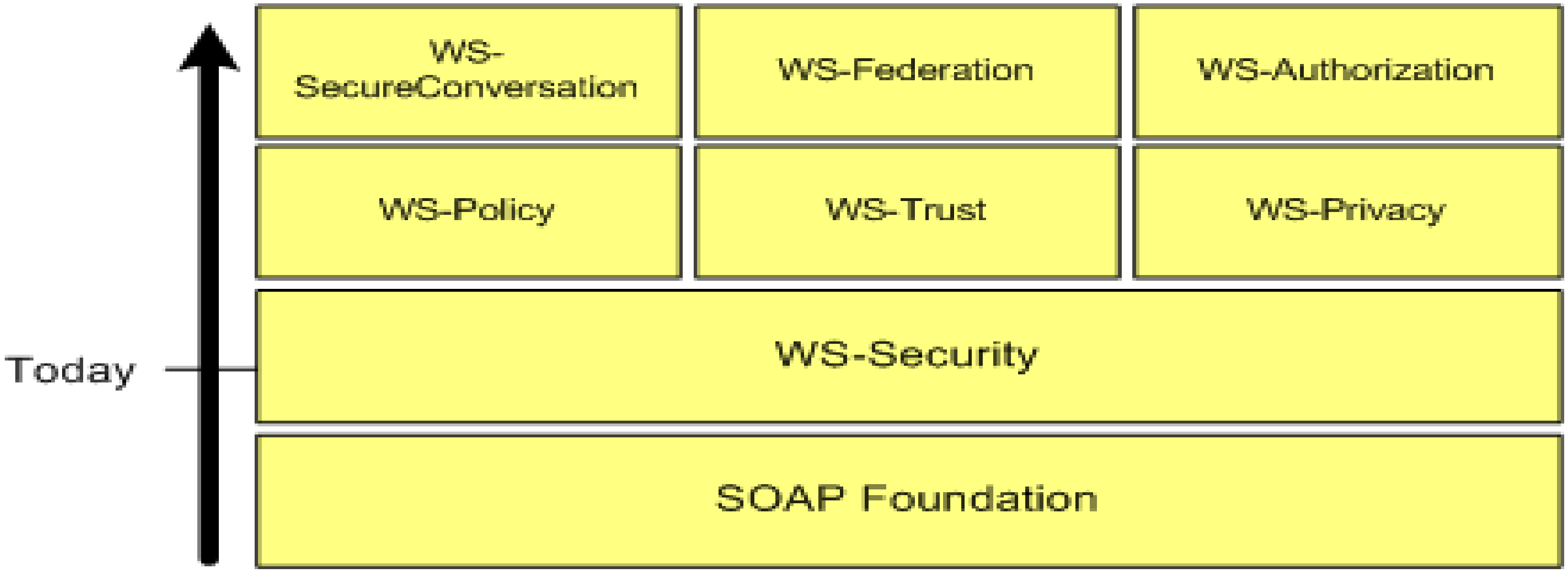
- Techniques for securing messages and authenticating communicators are centuries old.
- Securing Web Services has several parts
  - XML Message Security Concepts
  - Practical Implementations
- We will primarily examine the first.
- However, the WS-I profile concentrates heavily on the second

Outline

- Security Concepts and Considerations
  - Security concept classifications
  - Threat classifications
  - Scope
  - Network security layers
- XML Message Security
  - SOAP Message Security
  - XML Digital Signatures
  - XML Encryption
- WS-I Security Profile: Integrating XML Message Security with transport security.
- Shibboleth and SAML
- Other Standards
  - WS-Federation

Original Security Roadmap

- The original (2002) WS-Security road map is shown below.
  - WS-Security-->SOAP Message Security
- A comprehensive list of specifications is available from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wssecurspecindex.asp
  - WS-I is the crucible for these standards.
- We will concentrate on secure SOAP messages.



Source Material

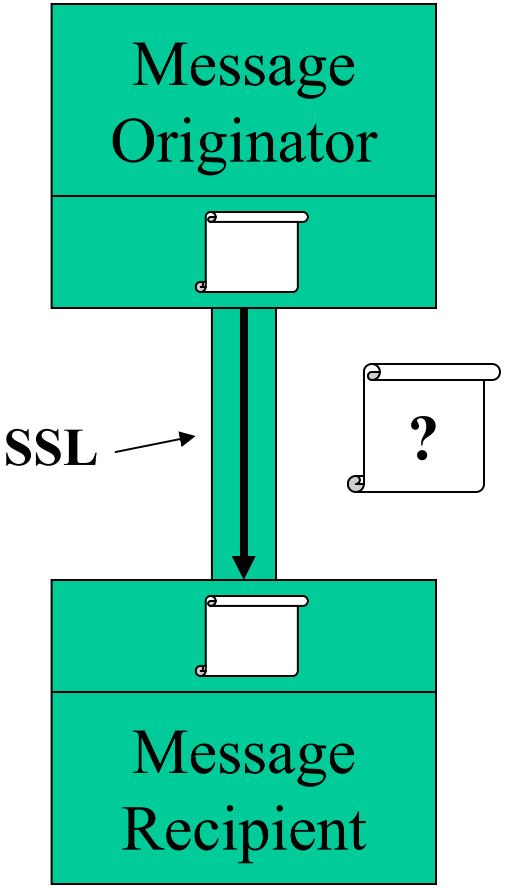
- WS-I Basic Security Profile
  - Working Group Draft: http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0-2004-05-12.html
  - Security Scenarios: http://www.ws-i.org/Profiles/BasicSecurity/2004-02/SecurityScenarios-0.15-WGD.mht
- SOAP Message Security 1.0:
  - Specification: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
  - Schema: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
- XML-Signature:
  - Specification: http://www.w3.org/TR/xmldsig-core/
  - Schema: http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd
- XML Encryption Specification: http://www.w3.org/TR/xmlenc-core/

Security Concepts and Considerations

Review basic security ideas, threats, and network architectures

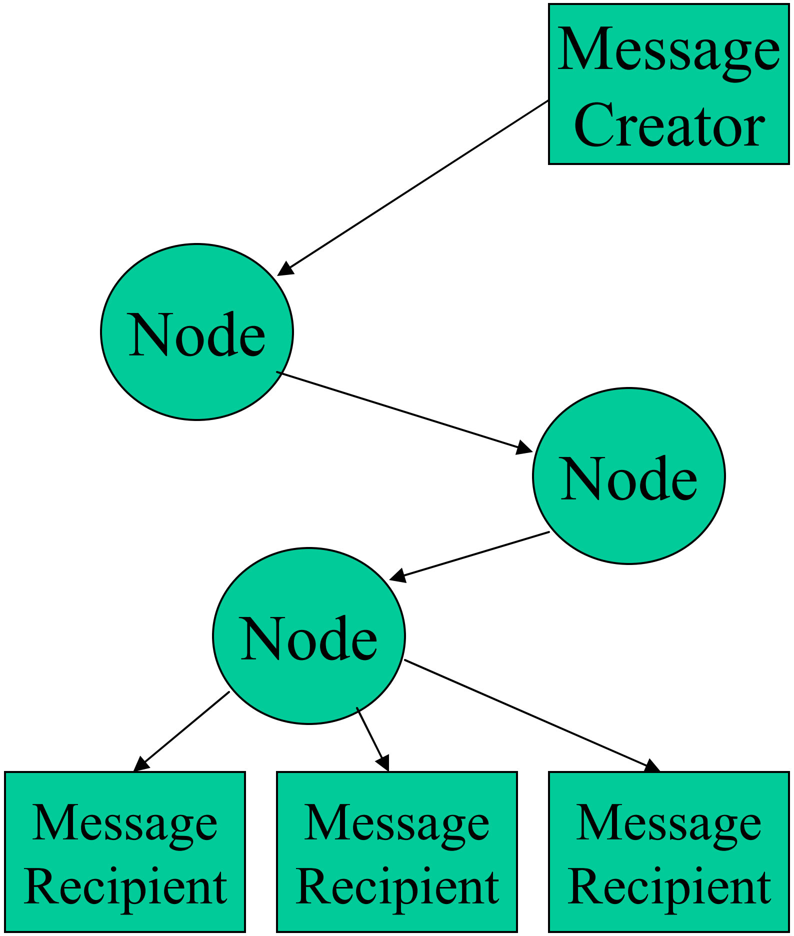
Web Service Security Basic Picture

- Web Services operate by exchanging (typically) SOAP messages.
- These messages may travel over secure network connections
  - Leverage typical Web security techniques like certificates and HTTPS
- The SOAP messages themselves may be signed, encrypted, and otherwise secured.
- Note this picture does not show any SOAP intermediaries.

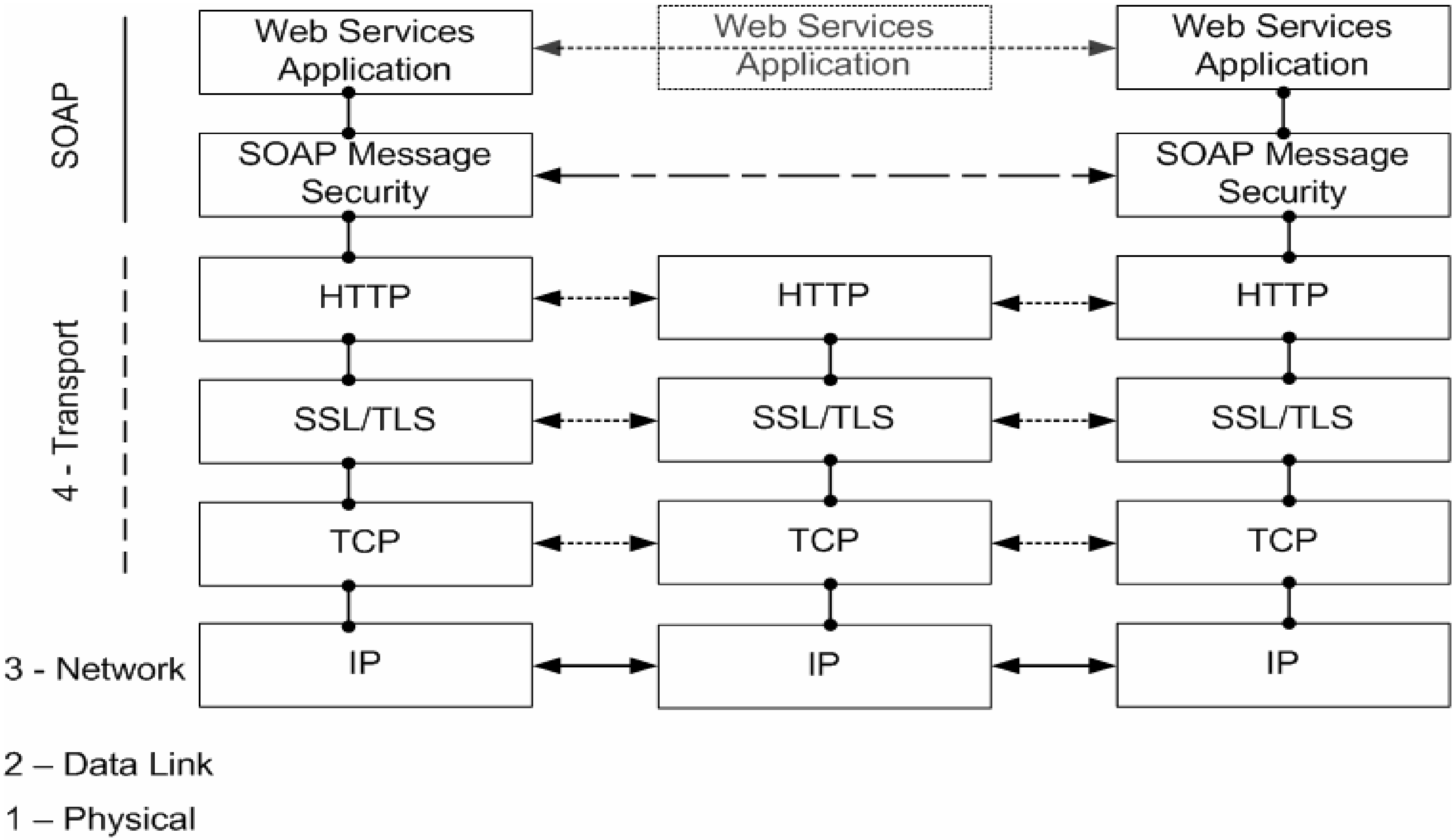


Security Challenges for Web Services

- The previous picture represents the commonplace client-server style security.
  - Message level security is redundant.
- But SOAP allows for other messaging patterns:
  - Multiple relaying brokers.
  - Multiple recipients.
- Each hop represents a different network connection.
  - May want to authenticate peers at each step.
  - Nodes may partially process messages.



Web Service Security Stack



Basic Terminology: Authentication

- Peer Authentication: corroboration that an entity is who it claims to be.
  - This applies to originators, relayers, and final recipients of messages.
  - You may think of these as software entities: web servers, client programs, broker nodes, etc.
  - In all cases, they may be required to prove their identity
  - Think: Web servers with X.509 certificates, HTTP Authentication
- Data (Message) Authentication: corroboration that the contents of the message come from the asserted source.
  - Note that messages may be handled by many different entities. This applies typically to the message originator.
  - Think: digital signatures.

Basic Terminology: Integrity

- Data Integrity: transmitted messages have not been changed, tampered with, etc. The recipient receives the same message that was sent.
  - Can be implemented in both the transport level (SSL) and message level (XML-Signature).
  - Transport level works point-to-point, or in a sequence of point-to-point transmissions.
  - Message level works independently of network connections. Necessary for multi-stepped transmissions.
  - Think: message hashing

Basic Terminology: Data Confidentiality

- Used to keep message transmissions private.
  - Typically, this is just encryption/description as we normally think of it.
- Can be implemented at both the transmission and message level
  - HTTPS and XML-Encryption
- SOAP provides additional confidentiality requirements.
  - Different sections may be encrypted by different keys.
  - Sections of XML may have layered protections
  - EX: when transmitting credit card info, different processors may have the right to see your name, your purchase, the cost, your card number, etc.

Message Uniqueness

- A particular message instance should only be transmitted once to the final recipient.
  - Ex: avoid multiple charges for the same purchase, or multiple submissions of the same job to an “expensive” computing resource.
- SSL connections provide this.
- The message level scenario is somewhat complicated.

Additional Security Concepts

- Authorization: does the authenticated entity have the right to access a resource?
  - Think: UNIX file permissions
  - Related to policy.
- Delegation and Trust: can an authenticated entity give another entity the right to act on its behalf?
- Federation: sharing security information and trust across security domains and implementations.
- Although important, these are currently out of scope of conservative WS-I security profile.
  - Hard to get right
  - Authorization and policy are very broad topics.
  - Delegation and federation introduce security compromises.

Some Web Service Threats

Threat	**Description**
Message Alteration:	The message content is changed in some way.
Message Snooping:	An unauthorized entity “sees” the message (perhaps processing it).
Impersonation:	an entity pretends to be another entity, sending or receiving unauthorized messages.
Message Replay:	Can involve both partial and complete message replay.
Man-in-the-Middle:	The MITM impersonates both the sender and the recipient.
Denial of Service	Death by a thousand cuts

SOAP Message Security Preview

An initial look before XML Signature
and XML Encryption

SOAP Message Security 1.0

- The current OASIS standard supersedes earlier WS-Security specifications.
- As (excessively) established in the previous section, WS security can involve both transport and message level security.
  - Messages may be signed and encrypted.
- How do we do this at the message level?

SOAP Security and Headers

- SOAP headers are the extensibility point for SOAP messages.
- This is where we put the security metadata
  - Security tokens, message digests, signing algorithms, etc.
- The following shows a sample SOAP message (abbreviated)
- SOAP security builds on XML-Encryption and XML-Digital Signatures, so we will detour through these before looking at this in detail.

```xml
<?xml version="1.0" encoding="utf-8"?>
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
xmlns:ds="...">
<S11:Header>
<wsse:Security xmlns:wsse="...">
  <ds:Signature>
        <ds:SignedInfo>
                <ds:CanonicalizationMethod Algorithm= ""/>
                <ds:SignatureMethod Algorithm=""/>
        </ds:SignedInfo>
        <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
        <ds:KeyInfo>
                <wsse:SecurityTokenReference>
                        <wsse:Reference URI="#MyID"/>
                </wsse:SecurityTokenReference>
        </ds:KeyInfo>
  </ds:Signature>
</wsse:Security>
</S11:Header>
<S11:Body wsu:Id="MsgBody“>…</S11:Body>
</S11:Envelope>
```

XML Signatures

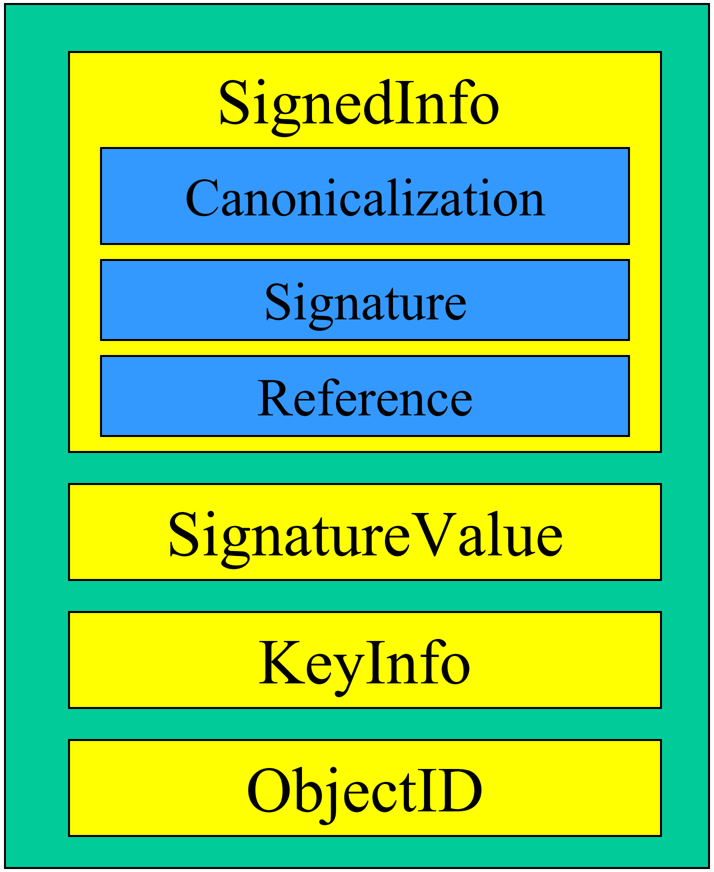
Digitally signing XML messages

XML Signature Intro

- The XML Signature specification represents a general way of signing XML content.
- Cryptographic “signing” involves the following steps:
  - A one-way hash of the message is created.
  - The hash is signed with a private key.
  - The signed hash and the message are transmitted.
- The recipient verifies the signature by hashing the received message and comparing this to the decrypted signature.
  - Use the sender’s public key to decrypt.
  - The two hashes should be bitwise identical.
- XML Signature tags provide both the signature and the tags necessary to verify it.
  - Envoloped/enveloping signatures that wrap child elements are not allowed by WS-Security.
  - Detached signatures apply to some other part of the document outside the tree, or even a remote document.

XML Signature Schema Synopsis

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



SignedInfo

Canonicalization

Signature

Reference

SignatureValue

KeyInfo

ObjectID

What Is a One-Way Hash?

- A hash function takes a variable length input and produces a fixed length output.
  - One-way==unique mapping of input to output.
- For cryptographic hashes, this amounts to a permanent mangling of the message.
  - You can’t guess the input from the output.
  - Similar input messages have extremely different hashes. A single bit change in input completely changes the output.
  - There is no decryption operation.
- Messages will always produce the same hash, so you can verify that data has not been changed by reproducing the hash.
- This is much faster than encryption/decryption.

A Signing Example

```
<Signature Id="MyFirstSignature" xmlns=http://www.w3.org/2000/09/xmldsig#>
  <SignedInfo>
    <CanonicalizationMethod Algorithm=“…"/>
    <SignatureMethod Algorithm=“…"/>
    <Reference URI=“…">
            <Transforms>
                        <Transform Algorithm=“…"/>
            </Transforms>
            <DigestMethod Algorithm=“…"/>
            <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
  <KeyInfo>
    <KeyValue>
            <DSAKeyValue> </DSAKeyValue>
     </KeyValue>
  </KeyInfo>
</Signature>
```

Notes

- The Algorithm attributes have been abbreviated
  - They provide URIs that point to named algorithms (i.e. SHA-1 message digesting).
- The next slide gives some examples

Some Algorithm URI Examples

Canonicalization	http://www.w3.org/TR/2001/REC-xml-c14n-20010315
Digest	http://www.w3.org/2000/09/xmldsig#sha1
Signature	• http://www.w3.org/2000/09/xmldsig#dsa-sha1 • http://www.w3.org/2000/09/xmldsig#rsa-sha1
Encryption	•http://www.w3.org/2001/04/xmlenc#tripledes-cbc •http://www.w3.org/2001/04/xmlenc#aes128-cbc

Tag Element	Purpose
CanonicalizationMethod	The name of the method used to create canonical XML.
SignatureMethod	Name of the method used to hash and sign the content.
Reference	Contains the digest method and the digest value. Can occur multiple times. URI attribute points to the digested resource.
Transforms (optional)	URI list for all the operations (XSLT, canonicalization, etc) that have been applied before digesting.
SignatureValue	The base64 encoded value of the signature.
KeyInfo (optional)	Includes the key that can be used to validate the signature.

XML Canonicalization

- One-way hashes, and thus digital signatures, depend on exact, bit-for-bit matches of the messages.
- This is difficult for XML
  - ASCII endlines are different: \m,\n
  - Different XML documents can be equivalent (see right)
  - Duplicated namespaces are allowed but cause canon. problems.
- Must specify the Canonicalization algorithm.

- <name>Bob</name>
-
  ```
  <name>
        Bob
    </name>
  ```
-
  ```
  <s11:Envelope
          xmlns:xx=“[someurl]”
          ymlns:yy=“[sameurl]”>
          <xx:FName>..</xx:FName>
          <yy:LName>..</yy:LName>
  </s11:Envelope>
  ```

Software

- A list of XML digital signature software is available here:
  - http://www.w3.org/Signature/

XML Encryption

Encryption rules for XML messages

XML Encryption Schema Summary

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

Key Concepts of Encrypted XML

- Encrypted XML is still XML
  - The encrypted value (in base64 encoding) of the original document is placed in another XML document.
- Encryption is granular
  - You can encrypt portions of a document, and you can successively
  - EX: child and gchild elements become progressively more sensitive, so apply encryptions to them in succession.
- XML encryption is mechanism-independent.
  - Specify the mechanism with a URI.

A Simple Example

- **Before**

```
<?xml version='1.0'?>
<PaymentInfo>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000'
      Currency='USD'>
      <Number>…</Number>
      <Issuer>…</Issuer>
      <Expiration>…</Expiration>
   </CreditCard>
</PaymentInfo>
```

- **After**

```
<?xml version='1.0'?>
<PaymentInfo>
<Name>John Smith</Name>
<EncryptedData
      Type='http://www.w3.org/2001/04/
      xmlenc#Element'
      xmlns='http://www.w3.org/2001/0
      4/xmlenc#'>
   <CipherData>
      <CipherValue>A23B45C56
      </CipherValue>
   </CipherData>
   </EncryptedData>
</PaymentInfo>
```

What Happened?

- First, note that the encrypted XML is still XML.
- We replaced everything after “John Smith” with new tags:
  - <EncryptedData> brackets the encrypted elements.
  - <CipherData> holds <CipherValue>, which holds Base64 binary data
    - The encoding for the encrypted data.
  - CipherData may also point to an external data source.
- Note we could actually have encrypted the elements hierarchically.
  - Expiration, issuer, and number could be encrypted separately from the CreditCard element, using different keys.

Including Additional Information

- The simple example assumes the recipient has all the necessary information to decrypt the message in some off-line fashion:
  - The decryption key.
  - Information about algorithms
- But you can of course include this information in the message.
  - Keys are added using techniques discussed in digital signature notes.
  - The EncryptionMethod element specifies the method used.
- EncryptionMethod's URI argument points to a standard name for the chosen method.

```
<PaymentInfo>
<Name>John Smith</Name>
<EncryptedData>
  <EncryptionMethod
    Algorithm="[Some URI]">
  <CipherData>
    <CipherValue>A23B45C56
    </CipherValue>
  </CipherData>
  </EncryptedData>
</PaymentInfo>
```

XML Encryption Software

- XML Encryption software is available from here:
  - http://www.w3.org/Encryption/2001/Overview.html.

SOAP Message Security 1.0

Using signatures and encryption to
secure web service messages.

SOAP Message Security 1.0 Mechanisms

- SOAP Message Security 1.0 (or SMS 1.0 in these notes) is designed to do the following:
  - Ability to send security tokens as part of the message
    - X509 certificates, kerberos tickets, etc.
    - These may be needed by the service to perform some operation using external security mechanisms.
  - Message integrity
    - Support multiple signature formats
  - Message confidentially
    - Support multiple encryption technologies

What Is Out of Scope?

- Establishing authentication tokens
  - These may use other mechanisms (Kerberos, PKI).
  - SMS 1.0 just transports tokens
- Deriving keys
  - Secure Conversation Specification, not (yet) part of WS-I.
- Establishing security contexts
  - Secure Conversation Specification, not (yet) part of WS-I.
- Establishing trust
  - WS-Trust: ftp://www6.software.ibm.com/software/developer/library/ws-trust.pdf
- Non-repudiation
  - Because someone will always ask

Building Up an Example

- The example on the right shows a pre-secured SOAP message.
- Namespace assignments have been removed to save space.
  - S11: namespace is the SOAP 1.1 spec.
  - X: namespace is some external namespace.
- The empty header will be filled in.

```
<?xml version=“1.0”
    encoding=“utf-8”?>
<s11:Header></s11:Header>
<s11:Envelop …>
    <s11:Body>
          <x:execCmd>
                   rm –r *.*
          </x:execCmd>
    </s11:Body>
</s11:Envelop>
```

Add the Security Information

- We start by adding the tag <Security> to the SOAP header.
- <wsse:Security> is used to sandwich the security section of the SOAP header.
- As usual, we can optionally specify actor and mustUnderstand attributes
  - Use role for S12.

```
<s11:Header>
  <wsse:Security
    xmlns:wsse=“…”
    S11:actor=“”
    S11:mustUnderstands=“”
    >
  </wsse:Security>
</s11:Header>
```

<Security> Schema Definition

- The full definition is given on the right.
- As you can see, it allows you to include ANY other elements from any other schema.
- This will allow us to include (for example) digital signature elements.
- Or anything else.

```
<xsd:complexType
    name="SecurityHeaderType">
  <xsd:sequence>
    <xsd:any
       processContents="lax"
        minOccurs="0"
        maxOccurs="unbounded">
    </xsd:any>
  </xsd:sequence>
  <xsd:anyAttribute
        namespace="##other"
        processContents="lax" />
</xsd:complexType>
<xsd:element name="Security“
    type="wsse:SecurityHeaderType">
```

Now Add in Signature Information

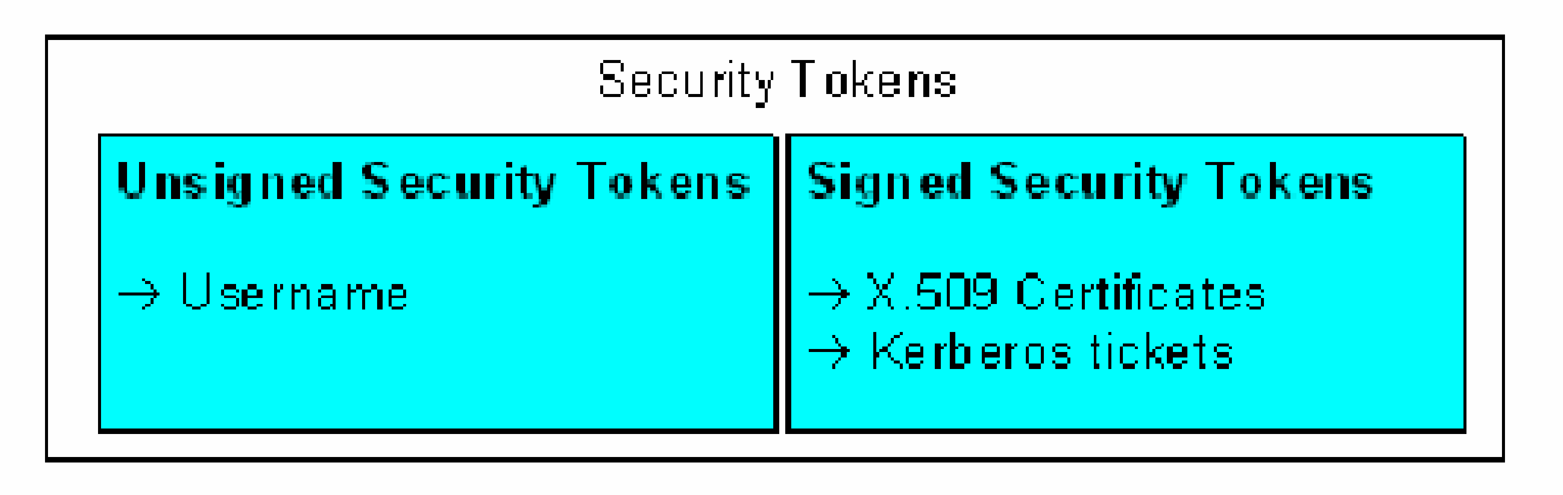
```
<s11:Header>
    <wsse:Security xmlns:wsse=“…”>
        <ds:Signature>
            <ds:SignedInfo>
              <ds:CanonicalizationMethod Algorithm=“”/>
              <ds:SignatureMethod Algorithm=“”/>
              <ds:Reference URI=“#MsgBody” >…
              </ds:Reference>
           <ds:SignedInfo>
           <ds:SignatureValue>…</ds:SignatureValue>
           <ds:KeyInfo> [To be expanded] </ds:KeyInfo>
        </ds:Signature>
    </wsse:Security>
</s11:Header>
<s11:Body  wsu:Id=“MsgBody” >…</s11:Body>
```

Notes

- We follow the same steps as in our earlier digital signature examples, with a few twists:
- The <Reference>’s URI attribute points to the body of the message.
- That is, we specify that the digested and signed part of the XML document is the SOAP body using the standard XML Signature technique.
  - “Detached” signing
  - Envelop signing is not allowed.
- <S11:Body> uses the Id attribute from the Web Services Utility schema to name itself.

Security Tokens

- Clarify some SMS 1.0 terminology
  - Claim: a declaration made by an entity.
    - Identity, key, group membership, privilege, etc.
  - Security Token: is a collection of claims
- Tokens may be signed or unsigned.



Security Tokens

Unsigned Security Tokens

→ Username

Signed Security Tokens

→ X.509 Certificates
→ Kerberos tickets

User Name Token Schema

- This token type includes 0 or more user name values.
- And it can include 0 or more elements from any thing else (xsd:any).
- And we can include attributes, also from other schemas.
- wsse:AttributeString just defines an xsd:string element that includes wsu:Id and possibly other (wildcard) attributes.

```
<xsd:complexType
    name="UsernameTokenType">
  <xsd:sequence>
    <xsd:element
      name="Username"
      type="wsse:AttributedString" />
    <xsd:any processContents="lax"
      minOccurs="0"
      maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute ref="wsu:Id" />
  <xsd:anyAttribute
    namespace="##other"
    processContents="lax" />
 </xsd:complexType>
```

UsernameToken in Action

```
<s11:Header>
    <wsse:Security>
          <wsse:UsernameToken>
                    <wsse:Username>
                    marpierc
                    </wsse:Username>
                    <wsse:Username>
                    mpierce
                    </wsse:Username>
          </wsse:UsernameToken>
    </wsse:Security>
</s11:Header>
```

- We might include this in a SOAP header.
- Two user names are included for the same entity.

Binary Security Tokens

- The schema definition is shown on the right.
- It really is just a string with attributes for including a Base64 binary blob.
- The wsse:EncodedString is an extension of wsse:AttributeString with an “EncodingType” attribute.
- EncodingType is an xsd:anURI that points to a named encoding.
  - Usually this is Base64
- Tokens must also include a ValueType attribute.
  - URI pointing to a formal definition.
- This is primarily intended for including X.509 and Kerberos tickets in the SOAP message.

```
<xsd:complexType
    name="BinarySecurityTokenType"
    >
 <xsd:simpleContent>
  <xsd:extension
      base="wsse:EncodedString">
     <xsd:attribute
          name="ValueType"
          type="xsd:anyURI" />
  </xsd:extension>
 </xsd:simpleContent>
</xsd:complexType>
```

Security Token References

- The previous slides assume that the tokens are included in the message.
- These can of course be external and pulled in from outside.
- Schema definition is to the right.
- Main body is a <choice> of <any> schemas (for extensibility).
- Attributes are wsu:Id (seen before) and wsse:Usage.
- The Usage attribute is a list of URIs.
  - Serve as formal names for usage patterns.

```
<xsd:complexType
    name="SecurityTokenReferenceType">
  <xsd:choice minOccurs="0"
      maxOccurs="unbounded">
    <xsd:any
    processContents="lax" />
  </xsd:choice>
  <xsd:attribute ref="wsu:Id" />
  <xsd:attribute ref="wsse:Usage"
    />
  <xsd:anyAttribute
    namespace="##other"
    processContents="lax" />
</xsd:complexType>
```

Token Reference Mechanisms

- <SecurityTokenReference> can include any elements from any schema, but it is normally intended to include SMS 1.0 elements.
- SMS 1.0 provides the following elements:
  - Reference: provides a URI (or fragment) to locate the external key.
  - KeyIdentifier: Use this as a non-URI unique identifier. Typically a hash of a unique name.
  - KeyName: A human-readable version of the KeyIdentifier.
  - EmbeddedReference: Use this to embed the token directly in the Token Reference.
    - For example, you can embed a SAML token here.

Digital Signatures and SOAP Message Security

- SMS 1.0 uses XML-Signature to sign messages.
- SMS 1.0 puts a few restrictions on signatures
  - Should not use Enveloped or Enveloping signature transforms.
    - Reason: headers may change in processing, breaking the signature's digest.
  - Exclusive XML Canonicalization is recommended.
    - This only copies namespaces explicitly used into the canonical document.

Where Do I Sign?

- You may use SMS 1.0 procedures to sign both the message content and any security tokens.
- Signed messages have these additional rules
  - Signing info must be prepended to any existing <wsse:Security> information.
  - All <ds:Reference> elements should point to some resource in the same SOAP envelop.

A Full Example

- The following text shows a signed message, including
  - The signature (signed digest)
  - The digest value
  - The binary security token that can decrypt the signature
  - Enough info (canonicalization, signing, and encryption algorithms) to allow you to verify the message contents.
- We use the wsu:Id to point to the signed content.
- We don’t sign the security token in this example.
  - It is a public key, so not secret.
  - Everything will fail if it is tampered with.

```
<?xml version="1.0" encoding="utf-8"?>
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="...“ xmlns:ds="...">
<S11:Header>
<wsse:Security>
  <wsse:BinarySecurityToken ValueType="...#X509v3" EncodingType="...#Base64Binary“
     wsu:Id="X509Token"> MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
     </wsse:BinarySecurityToken>
  <ds:Signature>
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm=“…"/> <ds:SignatureMethod Algorithm=“…"/>
      <ds:Reference URI="#myBody">
        <ds:Transforms><ds:Transform Algorithm= “…"/></ds:Transforms>
        <ds:DigestMethod Algorithm= “…"/> <ds:DigestValue>EULddytSo1...</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>BL8jdfToEb1l/vXcMZNNjPOV... </ds:SignatureValue>
  <ds:KeyInfo>
    <wsse:SecurityTokenReference><wsse:Reference
     URI="#X509Token"/></wsse:SecurityTokenReference>
   </ds:KeyInfo>
 </ds:Signature>
</wsse:Security>
</S11:Header>
<S11:Body wsu:Id="myBody"> … </S11:Body>
</S11:Envelope>
```

Encrypting Messages

- SOAP Message Security uses XML Encryption for message confidentiality.
- Note that we may encrypt both the body and the header, or portions thereof.
- The encrypted part replaces the original
  - <EncryptedData> replaces the original section.
  - We thus must create a manifest in the header, in <wsse:Security> for each <EncryptedData> section.
  - This information is put in the <ReferenceList> element.
- The SOAP header may also carry along encrypted keys necessary to decrypt the message.
  - Session keys encrypted with the recipient’s public key.
  - Recipient decrypts with private key, then uses session key to decrypt the message.
  - This is more efficient: PKI decryption only applied to small session key, which in turn decrypts the much larger message.

An Encryption Example

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
xmlns:ds="..." xmlns:xenc="...">
<S11:Header>
    <wsse:Security>
            <xenc:ReferenceList>
              <xenc:DataReference URI="#bodyID"/>
            </xenc:ReferenceList>
    </wsse:Security>
</S11:Header>
<S11:Body>
    <xenc:EncryptedData Id="bodyID">
            <ds:KeyInfo>
                        <ds:KeyName>…</ds:KeyName>
            </ds:KeyInfo>
            <xenc:CipherData>
                        <xenc:CipherValue>...</xenc:CipherValue>
            </xenc:CipherData>
    </xenc:EncryptedData>
</S11:Body>
</S11:Envelope>
```

Shibboleth and SAML Overview

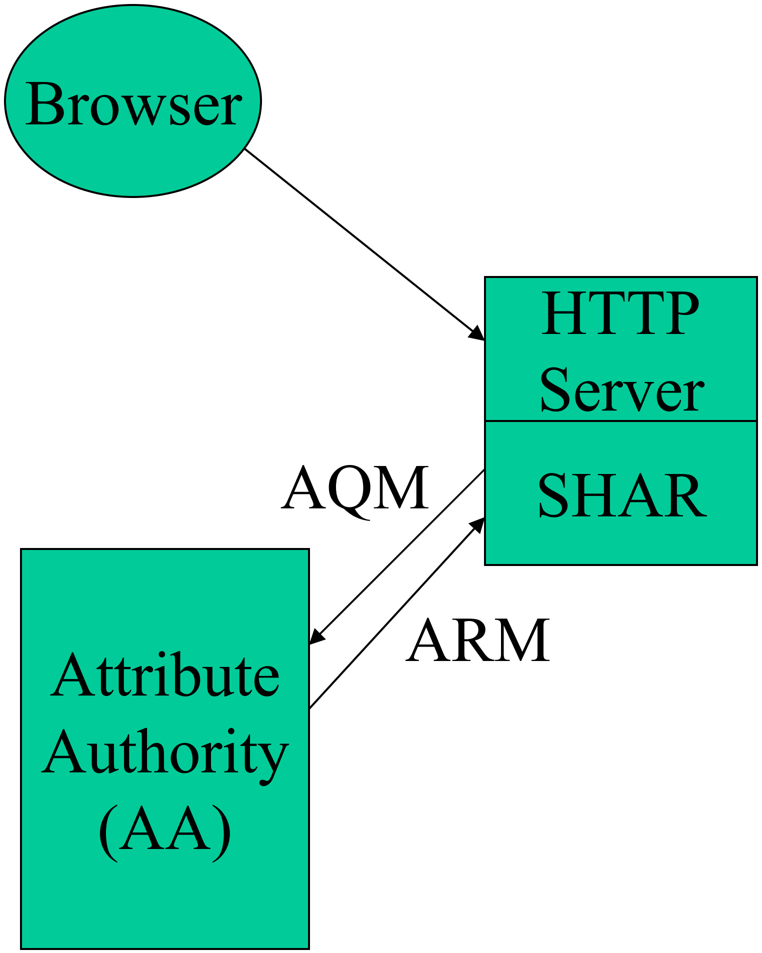
Some approaches to federation and authorization. Slides adopted from presentation by Liang Fang.

What Is Shibboleth?

- Shibboleth is a authorization system designed to control access to web material.
- It is designed specifically to meet US university system requirements
  - Student identity must be protected
    - Students should be able to view digital material anonymously.
  - Universities are federated in various ways (state, regional associations, MSI collaborations) so Web resources must be treated similarly.
- Thus Shibboleth has two major components
  - Access controls based on attributes rather than identity.
  - Federation.

How Does Shibboleth Work?

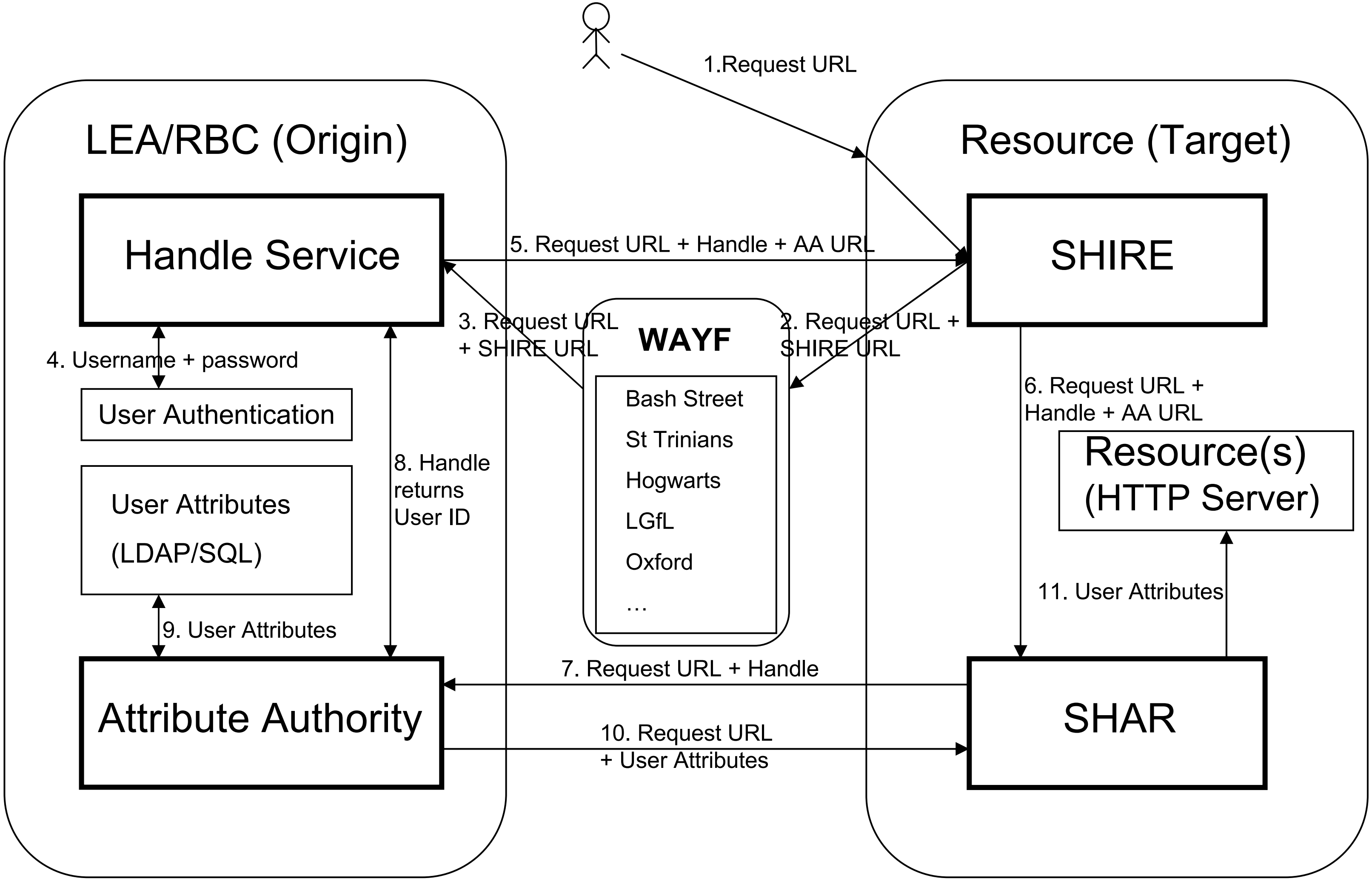
- A student is registered with his/her local university.
  - Attributes stored in LDAP, for instance.
- Student requests a resource from a modified HTTP Server.
- The server's SHAR requests attributes from the appropriate Attribute Authority.
  - AQM=Attribute Query
  - ARM=Attribute Response
- SHAR accepts or denies the request based on available attributes.



Federating Resources

- The previous picture assumes a single deployment (one university or department, for example).
  - It assumes the SHAR knows the correct AA to contact.
- To federate resources, we need additional services to find appropriate AAs for a given user.
- Shibboleth defines the “Where Are You From?” service (or WAYF) to do this.
  - Actually, the WAYF interacts with registered Handle Services, which are capable of associating the SHARS with AAs.
- The WAYF is effectively the federating piece.

Shibboleth User Authentication



LEA/RBC (Origin)

Handle Service

4. Username + password

User Authentication

User Attributes

(LDAP/SQL)

8. Handle returns User ID

9. User Attributes

Attribute Authority

1.Request URL

2. Request URL + SHIRE URL

3. Request URL + SHIRE URL

WAYF

Bash Street

St Trinians

Hogwarts

LGfL

Oxford

…

5. Request URL + Handle + AA URL

7. Request URL + Handle

10. Request URL + User Attributes

Resource (Target)

SHIRE

6. Request URL + Handle + AA URL

Resource(s) (HTTP Server)

11. User Attributes

SHAR

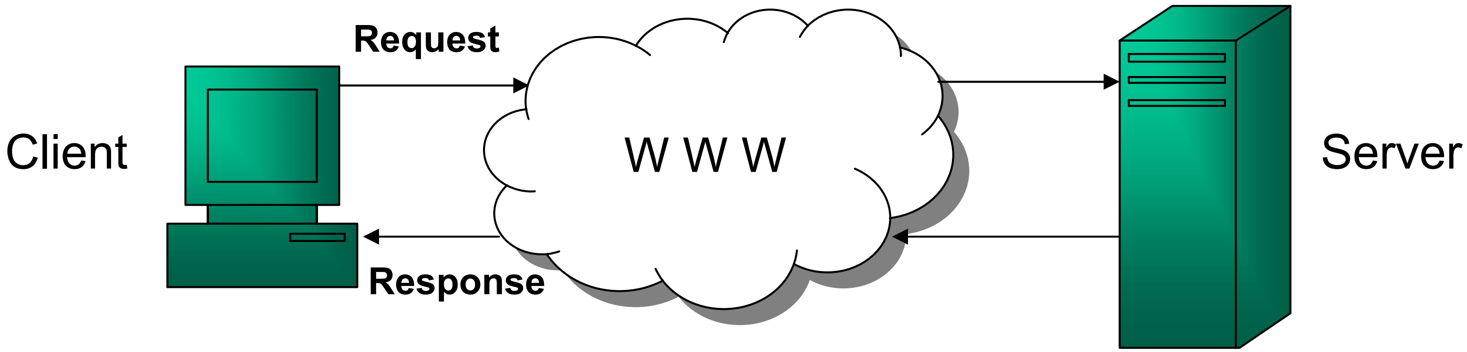
SAML and Shibboleth

- Shibboleth services are Web Services.
  - Communicate with SAML assertions.
- Shibboleth based on SAML:
  - SAML's attribute statement and assertion format
  - Query/response protocol for the AQM and ARM messages
- The two are compatible but independent technologies.
  - Shibboleth focuses on the browser users, while SAML deals with general scenarios including authorization decisions

Conclusion

End of Web Service Security, except for questions.

Accessing a Web Resource



- Client user accesses a free resource
- Client user is authenticated via a username and password and accesses a protected resource

Common Issues in Authentication

- High administrative burden
- Exposure of personal information
- Lack of traceability
- Password leakage
- Many passwords problem
- Resource accessibility is restricted
- Complicated to use

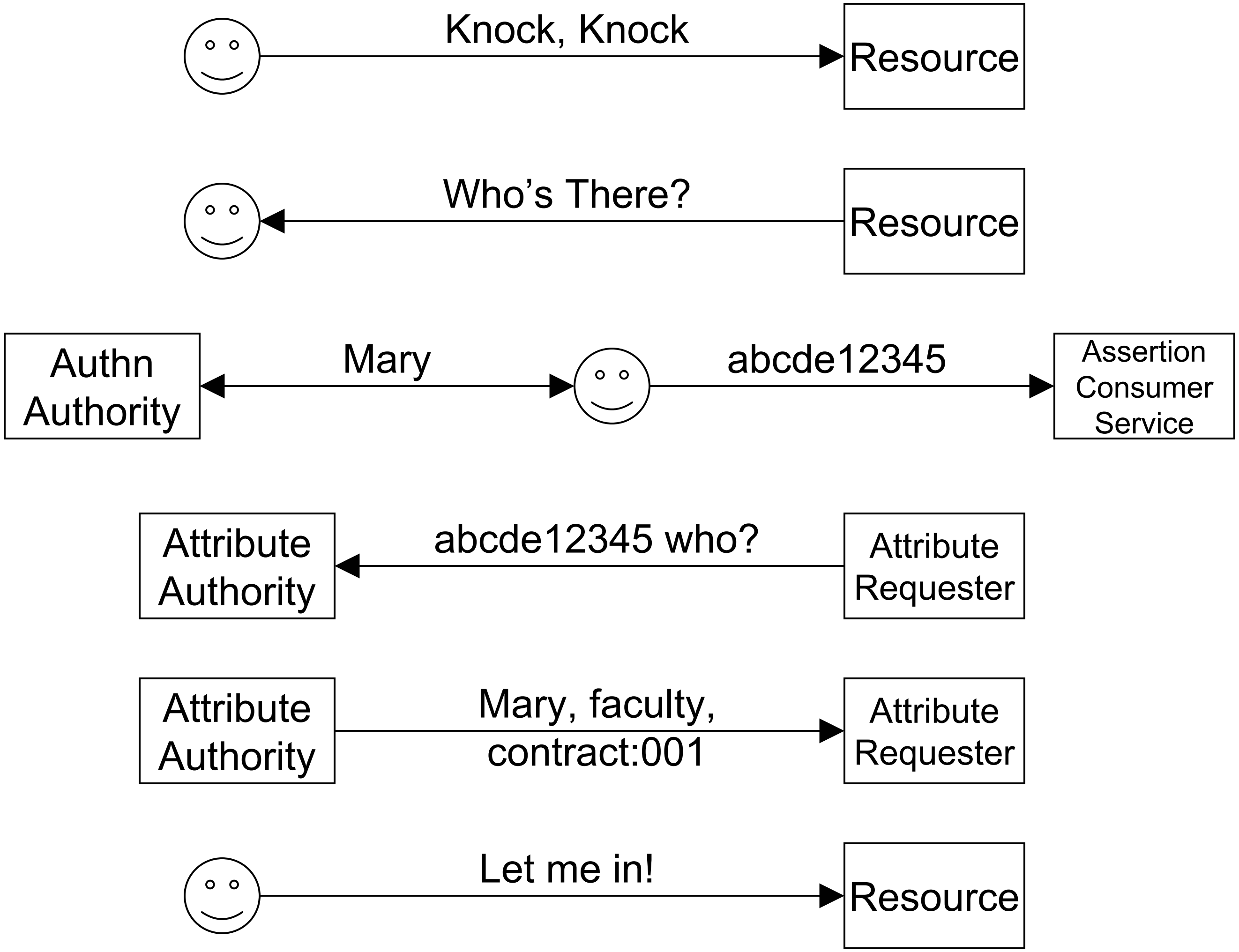
What is Shibboleth?

- Open source attribute-based single sign-on software with an emphasis on user privacy, built on the SAML 1.1 specification
- A provider and consumer of innovations in federated identity standards
- An enabling technology for Internet2, international, and regional efforts at federation in education and research

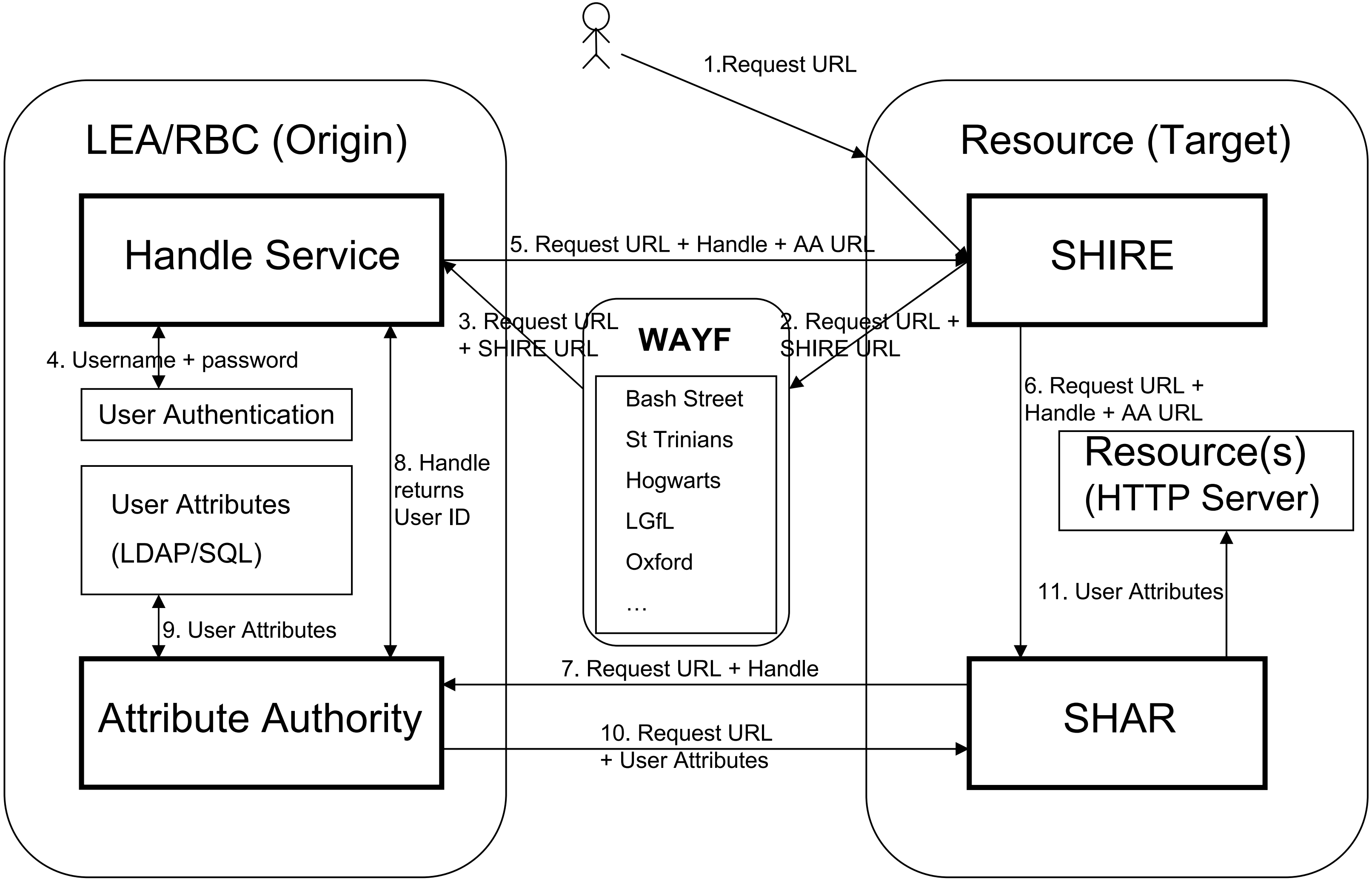
Use Cases

- Traditional web single sign-on
- Shared electronic learning resources
- Research resources (grids)
- Outsourced academic or administrative services
- Account linking across sites
- Delegated trust in portal scenarios (e.g. meta-searching)

High Level Architecture



Shibboleth User Authentication



Privacy

- Keep my identity secret
- Don’t share any of my privacy info with anyone else unless I authorize it.

Federations

- Shibboleth “federations” are sets of sites that share common trust and operational metadata.
- Federations generalize bilateral arrangements between sites so policy can be delegated and scaled.
- Deployments can span federations and one-off agreements, and the PKI accommodates this.

Federated Identity

- Users authenticate to their “home” or “origin” institution (identity provider)
- Identity becomes one of many attributes potentially sent to target sites (service providers)
- Authorization enforced by service provider, identity/attribute provider, or both
- Partitions responsibility, policy, technology, and trust

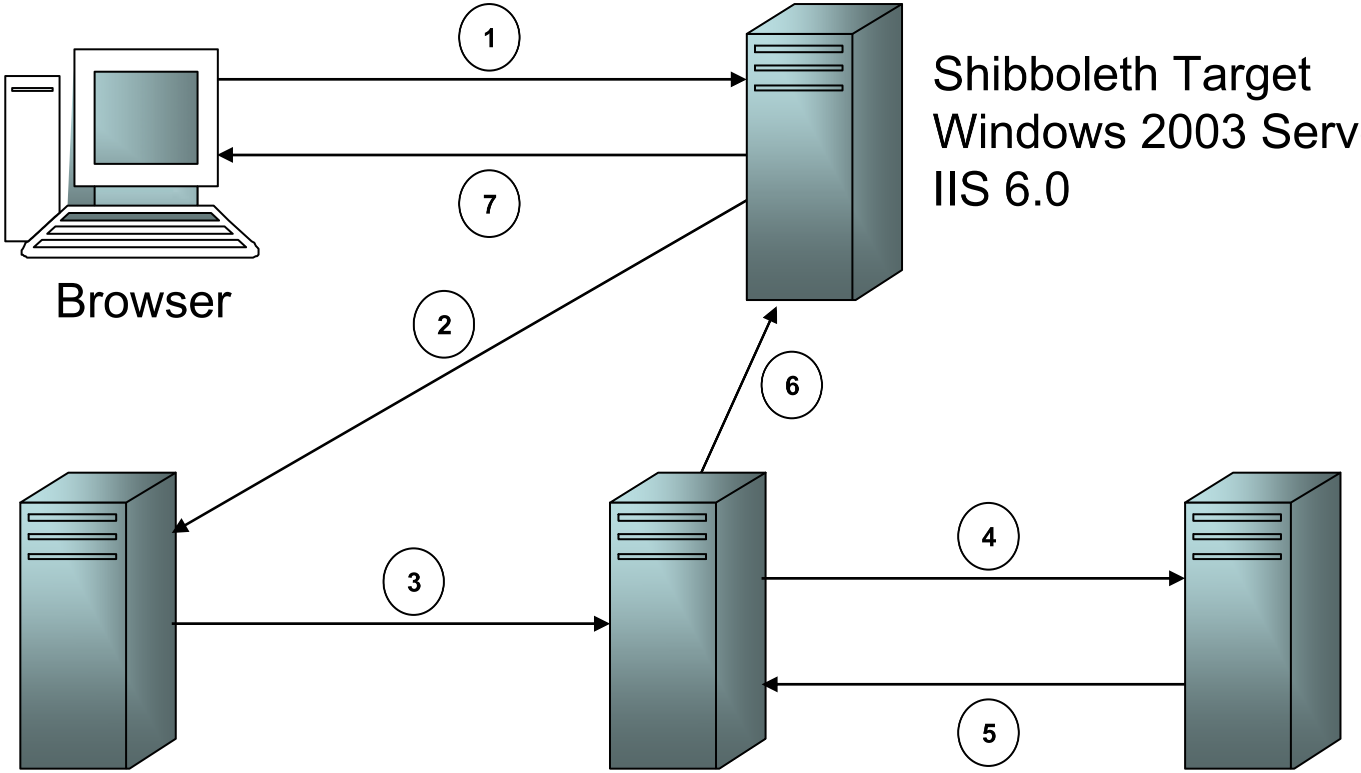
SAML 1.1 and Shibboleth

- Shibboleth based on SAML 1.x:
  - SAML’s Attribute statement and assertion format
  - Query/response protocol for the AQM and ARM messages
  - Shibboleth focuses on the browser users, while SAML deals with general scenarios including authorization decisions

Pros and Cons

- Pros
  - Low administrative burden
  - Exposure of personal information under user’s control
  - Same identity for all resources
  - User traceability
  - Resources can be accessed from any location
- Cons
  - (Possible) multi-stage authentication
  - Risks by federation

Shibboleth Demonstration



Shibboleth Target
Windows 2003 Server
IIS 6.0

Browser

WAYF Service
Windows 2003 Server
IIS 6.0

Shibboleth Origin
Windows XP Pro
Apache Server 2.0.49

LDAP Directory
(Active Directory)
Windows 2003 Server

Project Deliverables

- An open source SAML implementation (http://www.opensaml.org/)
- Java-based “origin” implementation (authentication and attribute authorities)
- “Target” implementations for Apache, IIS, with additional deployment vehicles in development, including Java and non-web application scenarios
- Federated PKI-based trust fabric