Making Web Services Secure:
An Introduction

Kelvin Lawrence
Distinguished Engineer & CTO Dynamic e-business Technology
IBM Software Group, Austin TX
klawrenc@us.ibm.com

NOTE: This is a fast moving area and updated slides may be used in the actual presentation. The updated slides will be on the post conference CD.
About Updated Slides

- Inserted slides are marked “ ★ "

- Modified slides are marked “ ✗ "

- This version of the slides will be on the post conference CD.
Agenda

- What is Security?
- SOAP and Security
  - Protocol-level and Message-level security
  - Why HTTPS is not enough for Web Services
  - A brief review of SOAP messaging
- Security specifications
  - WS-Security
  - PKI (Public Key Infrastructure)
  - XML Encryption
  - XML Signature
  - SAML and XACML
  - XKMS (XML Key Management Service)
  - WS-Security Roadmap (and specifications)

Resources
The Web Services “Stack”

- Business Process Execution Language
- WS-Coordination
- WS-Transactions
- WS-Policy
- WS-Security family of specifications
- WS-Reliable Messaging
- WSDL
- SOAP, SOAP Attachments
- XML, XML Infoset
- Transports
- Other protocols
- Other services
- UDDI
- Transport
- Description and Discovery
- Messaging and Encoding
- Quality of Service
- Business Processes
Security Goals and Requirements

- There is no such thing as absolute security
  - There are only risks
  - There are countermeasures to address these risks
  - Nothing can ever be proven to be 100% secure

- However, we can provide enough security to make e-business practical.
  - Must be based on strong, open standards to ensure interoperability between platforms
  - There are many existing security standards, and new and emerging ones, that we can leverage
  - We need a standard that says how security standards can be used for Web services

- Security requirements vary widely with different applications
  - There's no universal checklist
  - There are common requirements, but they may not all apply
Seven Aspects of Security

1. **identification:** Who are you?
2. **authentication:** How do I know your identity is true?
3. **authorization:** Are you allowed to perform this transaction?
4. **integrity:** Is the data you sent the same as the data I received?
5. **confidentiality:** Are we sure that nobody read the data you sent me?
6. **auditing:** Record of all transactions so we can look for security problems after the fact
7. **non-repudiation:** Both sender and receiver can provide legal proof to a third party (e.g. judge) that the sender did send the transaction, and the receiver received the identical transaction
Browser-model Security

- Browser-model security options
  1. No security – public data access
  2. "Security through obscurity" – see number 1
  3. Firewall security – EAI only
  4. HTTPS/SSL for secure point-to-point communication with known trusted parties

- We can use any of these for Web services
  ➢ (don't recommend #2)
Transport (Protocol-level) Security

- HTTP/SSL provides "protocol-level" or "transport-level" security
  - Identification, basic authentication, encryption, and (implied) integrity
  - Point-to-point security across one connection
  - Convenient, as it does not surface any security issues to the application
  - No other application requirements
Why Isn't HTTPS Enough?

- Limitation 1: Point-to-point security only
  - Identification, authentication, integrity, confidentiality stop at HTTPS end point
  - Implementation may use a gateway (which becomes the HTTPS end point) and continue with a different protocol
  - Security does not extend to external business partner

- We need **end-to-end security** across all connections in the life of a transaction for an effective business environment.
Why Isn't HTTPS Enough? (Continued)

- **Limitation 2: No digital signature for non-repudiation**
  - We want an integrity signature to persist...
  - All the way to a database used for audit trail
  - Prove message has not been modified
  - HTTPS has no signature (that can be used for non-repudiation)

- **Limitation 3: Element-wise encryption**
  - Decryption is necessary to route the message
  - HTTPS encrypts everything...
  - So you have to decrypt everything to route it
  - We may need certain data (credit card #) to remain encrypted all the way to endpoint

- **Limitation 4:**
  - No authorization, auditing
Message-level Security

- Security considerations with SOAP messaging
  - How to include security credentials in the message
  - How to use element-wise encryption: expose some parts for routing, hide critical data from unauthorized parties
  - How to include digital signatures
  - Security persists from originator to processing end-point,
  - For the life of the transaction, the way we need it
  - Security survives call to external business partner
  - Use with, or instead of, protocol-level security
SOAP in a Nutshell...

- The use of XML-formatted messages for implementing a request-response model of communication between two parties

- **SOAP Specification:**
  - Defines vocabulary for the message "envelope"
  - Defines body or "payload" of message only as XML
  - Vocabulary is defined by supplier of a web service

- **Supplementing SOAP Specification:**
  - SOAP with Attachments
  - Provisions for security extensions defined by OASIS WS-Security: SOAP Message Security specification
SOAP Hides the Technology Choices and Implementation Details from Both Parties

- Simple, standard XML messages
  - We only care about message format and content
  - The less we know about the implementation details, the less work for us!
SOAP Message Structure

- SOAP specification defines an "envelope"
  - "Envelope" wraps the message itself
  - Message is a different vocabulary
  - Namespace prefix is used
  - To distinguish the two parts

```
application-specific
message vocabulary

SOAP
envelope
vocabulary
```
XML Namespaces

- A simple qualification scheme that
  - Allows XML elements from more than one vocabulary to be mixed in the same document
  - Distinguishes elements that have the same name but are from different vocabularies
  - The string that names a vocabulary is usually a URI

```xml
<S:Envelope xmlns:S="http://www.w3.org/2002/06/soap-envelope">
  <S:Header> ... </S:Header>
</S:Envelope>
```

the “S” prefix is defined to mark elements from the SOAP 1.2 Envelope vocabulary

the “S” prefix in front of the “Header” element indicates that “Header” is in the vocabulary defined by [http://www.w3.org/2002/06/soap-envelope](http://www.w3.org/2002/06/soap-envelope)

[http://www.w3.org/TR/REC-xml-names/](http://www.w3.org/TR/REC-xml-names/)
A SOAP Request Message

```xml
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2001/06/soap-envelope"
    SOAP-ENV:encodingStyle="http://www.w3.org/2001/06/soap-encoding">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>IBM</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
A SOAP Request Message

```xml
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://www.w3.org/2001/06/soap-envelope"
  SOAP-ENV:encodingStyle="http://www.w3.org/2001/06/soap-encoding">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>IBM</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
One Great Thing about SOAP

- By using HTTP port 80 you can send messages right through the firewall.
  - This easing of security considerations is really convenient for programming!

One bad thing about SOAP...

- By using HTTP port 80 you can send messages right through the firewall!!!
  - Some consider this circumvention of security is really dangerous!
  - It's not dangerous – but it does require additional security considerations in the implementation
  - If we don't use the security offered by the firewall, we need to address security in the SOAP handler
SOAP: Messages and Security

- Security options:
  1. Transport-level security (e.g. HTTPS)
     - identity of other party
  2. Credentials for security of the message as a whole
     - confidentiality – cannot be read by an unauthorized entity
     - integrity – has not been changed by an unauthorized entity
     - authorization – able to prove your access rights, etc
  3. Separate credentials for security of message body
Security Specifications
WS-Security: SOAP Message Security

- A foundational set of SOAP message extensions that can be used when building secure Web services
  - Defines SOAP usage for other security specifications
- Supports, integrates and unifies several popular security technologies for message-level security:
  - Kerberos, Public Key Encryption, HTTPS, IPSEC, XrML
  - XML Signature, XML Encryption, XKMS from W3C
  - SAML, XACML from OASIS
  - Six new specifications (see later, and second session)
- Goals:
  - Enable enterprises to protect their investments and assets as business processes become increasingly recast as Web services
  - Same-domain and cross-domain secure messaging
  - Platform-neutral interoperability
  - End-to-end security
WS-Security: SOAP Message Security

- Flexible, composable specification
  - Designed to be used as basis for securing Web services
  - Wide variety of security models including PKI, Kerberos, and SSL

- Provides support for
  - Multiple security token formats
  - Multiple trust domains
  - Multiple signature formats
  - Multiple encryption technologies

- The token formats and semantics for using these are defined in the associated binding documents.

- This specification replaces and extends earlier work
  - e.g. the IBM/Microsoft W3C "SOAP-Sec" Note (January 2001) is now obsolete
SOAP Message Structure (2)

The SOAP specification defines the "envelope" vocabulary

- The "envelope" wraps the message itself
- The message is a different vocabulary
- A namespace prefix is used to distinguish the two parts

WS-Security defines extensions to the "envelope" vocabulary

- Container for security tokens
  - Username
  - x.509 certificate
  - Kerberos ticket
  - XrML
  - XML Signature
  - SAML
- Encryption details
WS-Security in OASIS

- TC Chartered Summer of 2002
  - First meeting was Sept 5th 2002
- The WSS-TC is responsible for:
  - The “core” WS-Security specification
    - WS-Security: SOAP Message Security
  - A set of “token” profiles for:
    - Username/Password
    - X.509
    - Kerberos
    - SAML
    - XrML
    - Minimal (small footprint)
    - XCBF (biometric)

WS-Security in OASIS (2)

- WSS-TC status:
  - The TC Has just completed (on October 19th 2003) the “Public Review” phase for:
    - WS-Security: SOAP Message Security
    - Username/Password
    - X.509
  - These specs are currently “Committee Drafts”
  - The TC is currently focused on remaining profiles
  - WS-I.org is now working on Basic Security Profile
The security element

The WS-Security specification defines a vocabulary that can be used inside the SOAP envelope. `<wsse:Security>` is the container for security-related information.

```xml
<S:Envelope xmlns:S="http://www.w3.org/2002/06/soap-envelope">
    <S:Header>
            Security information
        </wsse:Security>
    </S:Header>
    <S:Body>
        App-specific content
    </S:Body>
</S:Envelope>
```

Define and use WS-Security namespace
Putting Security Tokens in the SOAP Envelope

- **A Security Token** is a collection of “claims”
  - A claim is a declaration made by some entity, such as name, identity, key, group, privilege, capability, etc
  - “username” is an example of an unsigned security token

- **A Signed Security Token** is one that is cryptographically signed by a specific authority
  - An X.509 certificate is a signed security token
  - A Kerberos ticket is also a signed security token

- **An XML Security Token** is one that is defined with a separate XML schema, rather than simple or encrypted text
  - SAML and XrML are examples (SAML example shown later)
  - Can be included directly in `<wsse:Security>` container
The **UsernameToken** element

This element can be used to provide a user name within a `<wsse:Security>` element.

```xml
<S:Envelope xmlns:S="http://www.w3.org/2002/06/soap-envelope">
    <wsse:UsernameToken wsu:ID="myToken">
      <wsse:Username>KRL</wsse:Username>
    </wsse:UsernameToken>
  </wsse:Security>
</S:Body>
</S:Envelope>
```

Security Info

App-specific content
Signed security tokens, such as a Kerberos ticket or x.509 certificate, are binary content. They must be encoded for inclusion in the wsse:Security container.

```
<S:Envelope
 xmlns:S="http://www.w3.org/2002/06/soap-envelope">
 <S:Header>
  <wsse:Security
   <wsse:BinarySecurityToken wsu:ID="myToken"
    ValueType="wsse:Kerberosv5ST"
    EncodingType="wsse:Base64Binary">
    XIFNWZz99UUbalqIEmJZc0
  </wsse:BinarySecurityToken>
  </wsse:Security>
 </S:Header>
 <S:Body>
  App-specific content
 </S:Body>
</S:Envelope>
```
WS-Security Examples

- Let’s look at some other simple WS-Security examples...
  - Username token
    - Username and password
    - Username and password digest
  - Binary security tokens
- Let’s also look at
  - The WS-Security schema files
    - secext.xsd
    - utility.xsd
WS-Security AppNotes


- Examples of using WS-Security 1.0 Specification with SOAP messaging
  - With complete annotations of the various parts and what they do
  - Published in August 2002 by IBM and Microsoft

- Let’s take a very quick look...
Key Security Technologies

- Let's look at the existing XML security standards in the light of their use in Web services security:
  - PKI (public key infrastructure and certificates)
  - W3C XML Signature (digital signatures)
  - W3C XML Encryption
  - W3C XKMS (key management service)
  - OASIS SAML (Secure Authorization Markup Language)
  - OASIS XACML (Access Control Markup Language)

- So, let's examine how the WS-Security specification allows them to be used inside of a SOAP envelope:
PKI – Public Key Infrastructures

- Two complementary, asymmetric keys per owner
  - "Private key" is privately held by one individual or entity
  - Corresponding "public key" can be accessed by anyone

- PKI and Encryption
  - Encrypt using receiver's public key
  - Only the receiver's private key can decrypt the content

- PKI, Authentication, and Integrity
  - Create a certificate using content and sender's private key
  - Anyone can authenticate the source and content using sender's public key, and only that key will work

- PKI depends on Trust
  - Are keys from an unauthorized imposter?
  - Must trust key-issuer's authority

- For an in-depth introduction, visit:
XML-Signature Syntax and Processing 1.0

- Often called "XML Digital Signature" or "XML DSIG"
- Who: joint work between W3C and IETF
  - http://www.w3.org/Signature/
- Purpose: proof of integrity of XML content
  - The signed content has not changed since it was sent
  - Definition of schema for the signature (KeyInfo)
  - Procedures for computing and verifying such signatures
  - Signature survives parsing/generation operations
  - Sign entire document, portions, or combinations of these
- Status: W3C Recommendation, February 2002
- Implementation: see XML Security Suite
- Related specification: XML Exclusive Canonicalization
  - Specifies order of processing in computing a signature
  - http://www.w3.org/TR/xml-exc-c14n/
HTTPS vs. XML Signature

- **HTTPS:**
  - Implied integrity:
    - Assume encryption cannot be broken.
    - If we can decrypt it, the message must not have changed.
  - Integrity only for the life of the connection
  - Integrity of the entire content

- **XML Signature:**
  - Cryptographic integrity check (certificates and keys)
  - Integrity can be proven at any time in the life of the message, including after the fact from a transaction logging database
  - Integrity of the whole, any subsection, arbitrary data, or any combinations of these
Digital Signature: Sender

This example shows how message content can be signed, and the signature included in the message.

Sender's Private Key → Transform → Signature

Resource

Sender's Public Key

The public key can be included in the message, or made available in a key registry.

Key Registry

The public key can be included in the message, or made available in a key registry.
Digital Signature: Receiver

**Note!** Signatures verify that the content has not changed. They do **not** guarantee that the message has not been read by an unauthorized party.

Public key is taken from the message, or queried from a key registry.
The Digital Signature Transform

On sender side:
1. Creates a "digest" of the material to be certified. Essentially a hash of the content.
2. Encrypt the digest using sender's private key
3. Encrypted digest is the certificate
4. Send certificate with content

On receiver side:
1. Decrypt the certificate using sender's public key, yielding the sender's digest
2. Create a "digest" with the same method as was done on the sender's side
3. Compare sender's digest with the one we created in step two. They must be identical to pass the integrity test.

See http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/Overview.html#sec-Algorithms for details on digesting algorithms
XML Example: A Signed External Resource

```xml
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="#wssecurity_body_id_2601212934311668096_1040651106378">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>AWQKpmksMpzzT4PxcizO980gVHw=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>bNhT+DsNN9PR [binary data has been truncated]</SignatureValue>
  <KeyInfo>
    <wsse:SecurityTokenReference>
      <wsse:Reference URI="#wssecurity_binary_security_token_id_1603091_4272645"/>
    </wsse:SecurityTokenReference>
  </KeyInfo>
</Signature>
```

**What is signed?**
- Signature Value
- Public Key (optional)
SOAP with XML Signatures

- How do we use XML Signatures with SOAP messages?
  - XML Digital Signatures tells us how to sign arbitrary XML content.
- WS-Security defines new elements in the SOAP header and body to hold additional security elements.
  - In particular, it says how XML Signature should be used if present in a SOAP message.
  - Standardization of the new SOAP elements is essential for achieving interoperability
SOAP Example:  Header with Signature

```xml
<soapenv:Envelope>
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
      <wsse:BinarySecurityToken EncodingType="wsse:Base64Binary">
        MIIDQTCC ... [remaining binary data has been truncated]
      </wsse:BinarySecurityToken>
    </wsse:Security>

    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      ....see XML Signature example for full content...
    </Signature>
  </soapenv:Header>

  <soapenv:Body>
    <R:AppRequest xmlns:R="someURL"></R:AppRequest>
  </soapenv:Body>
</soapenv:Envelope>
```
W3C XML Encryption Specifications

Who: W3C Working Group
- [http://www.w3.org/Encryption/](http://www.w3.org/Encryption/)
- Started as joint proposal by IBM, Microsoft, Entrust

Purpose:
- Encrypting data and representing the result in XML
- Can encrypt: an entire XML document, elements, element content, arbitrary data, or a combination of these
- `<EncryptedData>` replaces encrypted element or content, or is the root of an encrypted document

Status: W3C Recommendations, December 2002
- XML Encryption Syntax and Processing 1.0
- Decryption Transform for XML Signature 1.0

Examples:
- Free IBM WSTK (listed in Resources)
- WebSphere Application Server v4 and v5
HTTPS Compared to XML Encryption

- HTTPS Encryption
  - Entire stream is encrypted – it's "all or nothing"
  - Must decrypt entire content for routing, \textit{etc.}
  - Encryption lasts only for the life of the connection

- XML Encryption
  - Can encrypt entire content, selected elements, element content, or arbitrary data
  - Various parties can read different parts
  - Some content can be left readable to allow routing, with no effect on encrypted content
  - Encryption persists through the life of the message, could even be encrypted in transaction logging database
Example: Entire Document Encrypted

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Bank of the Internet</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Unencrypted

```
<EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
    Type='http://www.isi.edu/in-notes/iana/assignments/media-types/text/xml'>
  <CipherData><CipherValue>A23B45C56</CipherValue></CipherData>
</EncryptedData>
```

- Can't tell what kind of transaction it is
  the result is similar when encrypting
  Arbitrary (non-XML) data

This is an abbreviation of the content, to fit the slide. Actual encrypted content would be longer.
Example: Element Encryption

```xml
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Bank of the Internet</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Unencrypted

```xml
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
                   Type='http://www.w3.org/2001/04/xmlenc#Element'>
    <CipherData><CipherValue>A23B45C56</CipherValue></CipherData>
  </EncryptedData>
</PaymentInfo>
```

Entire `<CreditCard>` element is encrypted, including the fact that it is a `<CreditCard>`
Example: Element Content (elements)

```xml
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Bank of the Internet</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Unencrypted

```xml
<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>
```

- Limit and Currency are visible
- Card Number, Issuer, and Expiration are encrypted
Example: Element Content (char data)

```xml
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Bank of the Internet</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Unencrypted

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

Only card number (character data) is hidden
SAML – Security Assertion Markup Language

- **Purpose:**
  - XML identity (name) and authorization **assertions**
  - Does not **perform** authentication / authorization...
  - ...provides a receipt that this has been done elsewhere

- **SAML assertions are presented as signed XML content**
  - SAML defines the format of assertions and protocols for getting them

- **Who:** Oasis “Security Services” Technical Committee

- **Status:** SAML 1.1: OASIS Standard, November 2002
SOAP with SAML Assertion

- Example:
  - SAML asserts "Mark Colan is an employee of IBM"
  - Document is signed by IBM Corporation
  - Assertions can be used for authorization test
    IF you trust the authority of the signer
  - Assertions can specify authorizations, or
    authorizations can be implied by authentication
Use of SAML and WS-Security

SAML is included directly in `<wsse:Security>`, with a `wsse:SecurityTokenReference` to locate it.

```xml
<S:Envelope>
  <wsse:Security>
    <saml:Assertion MajorVersion="1" MinorVersion="0"
      AssertionID="SecurityToken-ef375268"
      Issuer="elliotw1"
      IssueInstant="2003-07-23T11:32:05.6228146-07:00"
      xmlns:saml="um:oasis:names:tc:SAML:1.0:assertion">
      ...SAML content...
    </saml:Assertion>
    <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier wsu:id="..."
        ValueType="saml:Assertion">
        SecurityToken-ef375268
      </wsse:KeyIdentifier>
    </wsse:SecurityTokenReference>
  </wsse:Security>
</S:Envelope>
```

...etc...

Security Info
XACML – XML Access Control Markup Language

Purpose:
- Element-wise access control policy for XML documents
- "Representation of rules that specify the who, what, when and how of information access" – Simon Y. Blackwell, chair
- Defined as schema and namespace

Who's doing it:
- OASIS working group in coordination with SAML group
  - http://www.oasis-open.org/committees/xacml/

Status:
- OASIS Standard: February 2003

Based on prior work
- IBM Tokyo Research Labs:
- WWW9 research work: http://www9.org/w9cdrom/419/419.html
XKMS – XML Key Management Service

- A proposal from Verisign, WebMethods, Microsoft
  - [http://www.w3.org/TR/xkms/](http://www.w3.org/TR/xkms/) is the XKMS 1.0 "note" to W3C, used as input to spec creation process
  - XKMS 2.0 spec is being worked on at W3C

- Purpose:
  - Create schema specifications for operations used to access a standard key management Web service
  - X-KRSS: XML Key Registration Service Specification:
    - register my key
  - X-KISS: XML Key Information Service Specification:
    - locate a key

- Sample Implementation:
- IBM WSTK (see Resources section)
XKMS – XML Key Management System

XKMS / PKI host

Register key / Revoke certificate / Recover key (X-KRSS)

Send signed document (SOAP)

Signer host

Verify Signature (X-KISS)

Verifier host
WS-Security: Roadmap


- Serves as technical overview for WS-Security
- Describes other specifications that support other security requirements

Available now: WS-Policy, WS-Trust, WS-Secure Conversation
Planned: WS-Privacy, WS-Federation, WS-Authorization
WS-Security: Roadmap


- Includes scenarios to help with the creation of the specs:
  - Direct Trust using Username/Password and Transport-Level Security
  - Direct Trust using Security Tokens
  - Security Token Acquisition
  - Firewall Processing
  - Issued Security Token
  - Enforcing Business Policy
  - Privacy
  - Smart Clients
  - Web Clients
  - Mobile Clients
  - Enabling Federation
  - Validation Service
  - Supporting Delegation
  - Access Control
  - Auditing

- Now let's have a look at the new WS-Security specs.
WS-Policy

Now consists of four specifications:

- **WS-Policy (Web Services Policy Framework)**
  - Framework used to describe a broad range of service requirements, preferences, and capabilities

- **WS-Policy Attachments**
  - How to bind policies to XML, WSDL, UDDI artifacts

- **WS-Policy Assertions**
  - Defines basic assertions that can be specified in a policy
  - *e.g.*: specify character coding, natural language, spec versions

- **WS-Security Policy Language**
  - Defines security policies
WS-Trust

- Model for direct and brokered trust relationships
  - Third parties and intermediaries
  - Primitives and extensions for using security tokens
    - issue, exchange, validate
  - Manage credentials across different trust domains
WS-Privacy

- Planned.
  - Will be a model for how users state privacy preferences, and for how Web Services state and implement privacy practices.
  - More information: see Web Services Security Roadmap
WS-Secure Conversation

- Security for long-running conversations
  - Manage and authenticate message exchanges
  - Security context exchange
  - Use of session keys
  - Asynchronous exchanges

WS-Federation

- Planned.
  - Will describe how to manage and broker the trust relationships in a heterogeneous federated environment including support for federated identities.
WS-Authorization

- Planned.
  - Will define how Web services manage authorization data and policies.
  - More information: see Web Services Security Roadmap
The Role of the WS-I Organization

Achieve Web services interoperability
Encourage Web services adoption
Accelerate Web services deployment

“WS-I will act as a standards integrator, therefore bringing some coherence to the effort carried out concurrently by the W3C, Oasis, OAG and other informal groups.” - Gartner Group
Resources
IBM's Free WSDK 5.0

- **WebSphere SDK 5.0**
  - On-ramp to Web services for Java programmers
  - An integrated kit for creating, discovering, invoking, and testing Web services
  - Embedded IBM WebSphere Application Server Express V5.0 included
  - Support for WS-Security, XML Encryption, XML Digital Signatures
  - Includes an implementation of JDK 1.3.1
  - Migration is available to production version of IBM WebSphere Application Server V5.0

- **Versions available FREE for Windows and Linux**
  - Available for free download from
Emerging Technologies Toolkit (Formerly WSTK)

- Contains many technologies useful for Web services
- For Web Services Security, it features demos:
  - **AXIS Digital Signature Demo**: uses Apache Axis handlers to process digital signatures without app-level code
  - **AXIS Encryption Demo**: same for XML Encryption processing
  - **Federated Identity Demo**: supports cross enterprise authentication within a heterogeneous federation of services and security systems
  - **Security Policy demo**: client evaluates the policy assertions available for two services and determines whether the request should be digitally signed or not
  - **WS-Privacy demo**: a simple Web app maintains personal data in the Profile Service, and grants or denies access to that data based on authorization decisions made by the Privacy Policy
  - **XKMS** key management demo
WebSphere Application Server v5

New! A next generation application server providing extended J2EE application optimization and integration capabilities for composing and choreographing adaptable applications based on a progressive Web services-oriented architecture.

The core Web services J2EE 1.3 certified application server enabling industry-leading QoS and flexible deployment options.

An easily approachable “on ramp” to e-business, providing fast and productive development, deployment of dynamic Web applications.

A J2EE-certified Web services application server specifically optimized to the unique QoS of z/OS.
IBM alphaWorks

http://ibm.com/alphaWorks

- Hundreds of tools for Web Services, XML, Java
  - Early versions of features that may be in products
  - Some are solid production-code (XML4J, LotusXSL)
  - Some are experimental, prototypes
  - Free download and use

- Web Services Toolkit and demos
People-based workflow
Learn more about business processes, their relationship to workflow and Web services today, and the challenges that lie ahead. (Articles)

- Speed-start Web services: Want to build and deploy Web services based on open standards? We have collected all the tools, training, and support you need to get started today. (Articles)
- Web services interoperability, Part 2: Download a version of the demo application from Part 1 for IBM WebSphere and run it on your own local computer. (Articles)
- Explore the Web Services Bus, Part 2: See how the Web Services Bus stacks up against the SOAP-based Apache Axis, as Greg Flury walks you through a simple example Web service. (Articles)
- First look at WS-I Usage Scenarios: Check out the new usage scenarios for WS-I Basic Profile conformance based on the first public Working Group Draft of the WS-I Usage Scenarios document. (Articles)
- Web services interoperability, Part 1: Implement a purchase order business process across many different Web services environments. (Articles)

- Web Services Toolkit demo: Learn how the WSTK will help you design and execute Web service applications that can automatically find one another and collaborate in business transactions. Check out the other Web services demos, too.
- Web services standards and specifications: View this listing of all current open standards and specifications that define the Web services family of protocols.
- Invoking Web services from SQL: Integrating relational data with Web services Web services: This tutorial demonstrates an easier way to generate user-defined functions (UDFs) that provide access to Web services from database applications. The generated UDFs can be used in SQL statements to combine relational data with dynamic data retrieved from a Web service.

Discussion forums
Tutorial: Securing Web Services

- Learn more about WS-Security from this new tutorial:

- Discusses:
  - Basic security requirements
  - HTTP and HTTPS capabilities
  - WS-Security Roadmap (future technologies)
  - WS-Security and XML Signature, XML Encryption
  - XML and Java code examples and a lab you can do with the free WSDK5

- Includes exercises to practice what you learn
Web Services Security: Moving Up the Stack

- Overview of recently released specifications
- (WS-Trust, WS-Policy, WS-Secure Conversation)

Questions?

- [ibm.com/webservices](ibm.com/webservices)
  - Whitepapers on IBM's vision of e-Business On Demand enabled by Web services
- [ibm.com/developerWorks/webservices](ibm.com/developerWorks/webservices)
  - Web Services Zone on developerWorks - resources for customers and developers on the use of XML
- [ibm.com/alphaworks](ibm.com/alphaworks)
  - Site for free emerging tools and technologies from IBM
- [oss.software.ibm.com](oss.software.ibm.com)
  - UDDI4J, WSDL4J open source Java class libraries
- [xml.apache.org](xml.apache.org)
  - Apache SOAP and other open source XML tools

Get updated versions of this talk or copies of many other talks on Web services and XML