Demystifying WS-Security

Herndon, VA
October 11, 2007

Introduction / Logistics

- Good Evening and Welcome to WS-Security
  - This is about WS-Security
  - **NOT** Web Service Security, **NOT** SOA Security (Why?)
  - This hour will cover key concepts of WS-Security, which you can use to “demystify” WS-Security
- Kenneth Lin, Associate, Booz Allen Hamilton
  - Enterprise Security Architect focusing on SOA Security, Identity and Access Management and Healthcare Security
- This presentation is discussion oriented so ***your brain power is needed*** in addition to your ears! (Sit and listen won’t work)
- Cell phones – Off/Muted/Vibrated (PLEASE)

Table Of Contents

Let’s review WS-Security prerequisites…

- XML: Meta-Language for Data-Oriented Interchange
  - Text-based, Platform Neutral, Universal data format, and is Self-describing
- XML Signature: Safeguarding the Identity and Integrity of XML Documents
- XML Encryption: Ensuring Confidentiality of XML Documents
- SOAP: An XML-based packaging scheme to transport XML documents from one application to another
  - SOAP defines an envelope, header, and body for encapsulating the XML data
  - SOAP is transport-independent
- SAML: Security Assertion Markup Language
  - An XML-based standard for exchanging authentication, attribute and authorization decision data between security domains
  - Three XML-based mechanisms: Assertions, Protocol, and Bindings

… and try the first 1 million dollar question …

What is the difference between

a SOAP message and an XML document?

Consider the following scenarios …

- Postal cards
  - Your friend sends you a X’mas greeting card
  - Your friend sends you a wedding invitation
- Voice lines
  - You called 1-800 line to check your credit card balance
  - You called your boss to discuss a salary raise
- How are the mechanisms applied in each scenario different from each other?
- What are the cost differences?
- So, can you tell what is a document and what is a message?

Now that you understand messages vs. documents, let's try the following questions …

- True or False:
  - All SOAP messages are XML documents
  - All XML documents are SOAP messages
  - XML Signatures are to sign XML documents
  - We can use XML Signatures to sign SOAP messages
  - XML Encryptions are to encrypt XML documents
  - We can use XML Encryptions to encrypt SOAP messages
  - SAML facilitates and exchanges identity, authentication and authorization information
  - SAML can facilitate and exchange identity, authentication and authorization assertions for SOAP messages

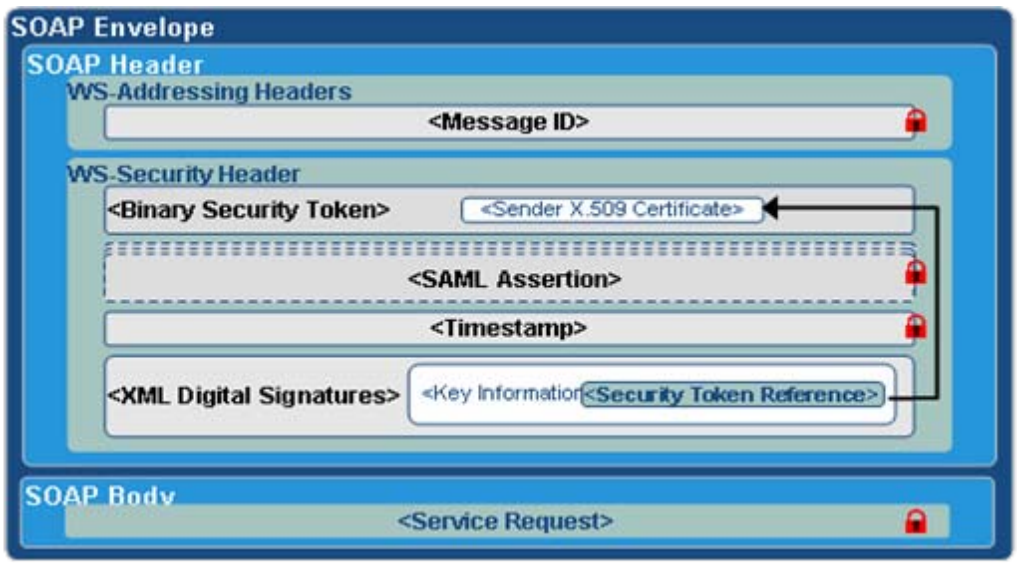
Why do we need WS-Security (WSS)

- The goal (WSS 1.1)
  - WS-Security is to enable applications to conduct secure SOAP message exchanges
  - WSS facilitates secure transactions to the service consumers and providers
- From the previous quiz:
  - We can use XML Signatures to sign SOAP messages
  - We can use XML Encryptions to encrypt SOAP messages
  - SAML can facilitate and exchange identity, authentication and authorization assertions for SOAP messages
- If SAML, XML signature, and XML Encryption can provide security to SOAP messages,

Why do we “still” need WS-Security?

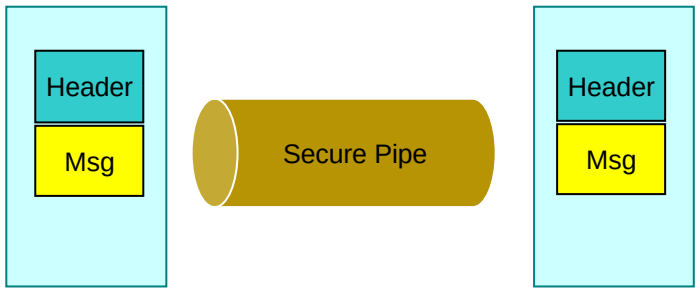
What is WS-Security (WSS)

- Describes how to use security constructs to secure SOAP messages
  - Major WSS security constructs: Security Tokens, XML Signature, XML Encryption
  - WSS defines element names for packaging security constructs into SOAP messages
  - WSS defines processing rules for processing security constructs in SOAP messages
- WSS is SOAP message level security (as opposed to HTTP transport layer security)
- WSS defines no new security technology; it focuses on applying existing security technologies such as X.509 certificates, SAML assertions, XML Signatures, and XML Encryption to SOAP messages

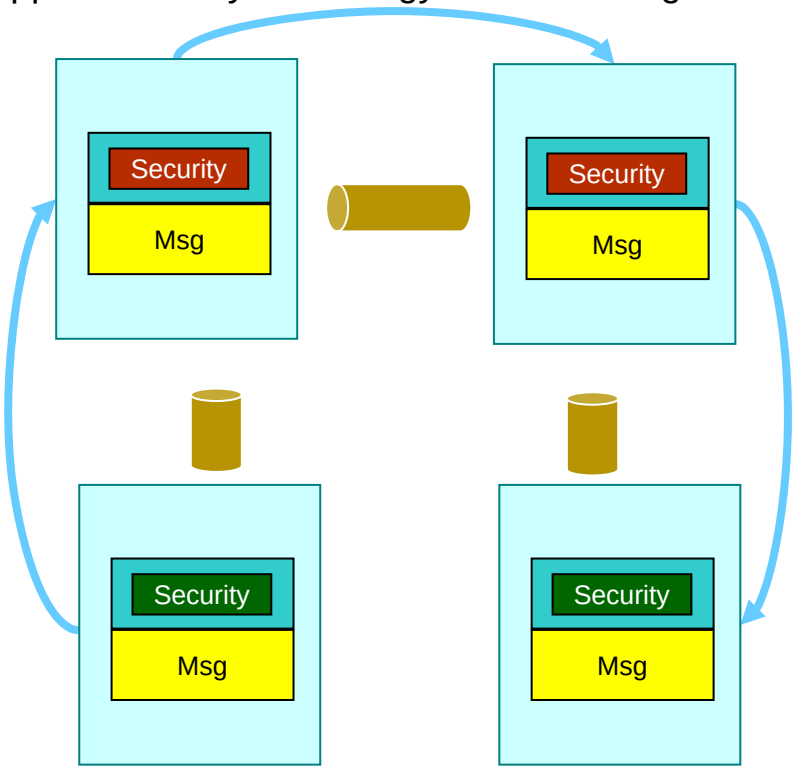


SOAP Message security vs. HTTP Transport layer security

- Transport layer security
  - Establish a secure pipe between two endpoints



- Message security
  - Applies security technology to the message itself



Transport layer security vs. Message security : Pros and Cons

- **HTTP Transport Layer Security**
- Pros
- Mature: Tried and true
- Support: Supported by most servers and clients
- Understood: Understood by most system administrators
- Simpler: Generally simpler than message-level security alternatives

- Cons
- Point to point: Messages are in the clear after reaching SSL endpoint
- Relay point exposure: Cannot have partial visibility into the message
- Inflexible: Cannot have different security for messages in and messages out
- Transport dependent: Applies only to HTTP

- **SOAP Message Level Security**
- Pros
- Persistent: Allows the message to be self-protecting
- Granularity: Portions of the message can be secured to different parties
- Adaptive: Different security policy can be applied to request and response Transport independent

- Cons
- Volatile: standards, tools
- Complex: encompasses many other standards including XML Encryption, XML Signature, X.509 certificates, and many more
- Performance: posses a performance penalty in transmission time, memory and CPU

Table Of Contents

WSS Specification 1.1 Goals and Requirements

- Goals
  - Enable applications to conduct secure SOAP message exchanges
  - Provide a flexible set of mechanisms that can be used to construct a range of security protocols
  - Significant efforts must be applied to ensure that security protocols constructed using WSS are not vulnerable to any one of a wide range of attacks
  - Describe a single-message security language
- Requirements (must support):
  - Multiple security token formats
  - Multiple trust domains (Trust = authority and privileges)
  - Multiple signature formats
  - Multiple encryption technologies
  - End-to-end message content security and not just transport-level security
- Non-Goals (Outside scope)
  - Establishing a security context or authentication mechanisms
  - Key derivation
  - Advertisement and exchange of security policy
  - How trust is established or determined
  - Non-repudiation

WSS 1.1 Message Protection Mechanisms

- Major threats of SOAP message security considered
  - Messages could be modified (integrity) or read (confidentiality) by attacker or
  - An antagonist could send messages to a service that, while well-formed, lack appropriate security claims (authenticity) to warrant processing
  - An antagonist could alter a message (identity and integrity) to the service which being well formed causes the service to process and respond to the client for an incorrect request
- Message Security Model (Sender authenticity)
  - Uses security tokens combined with digital signatures to protect and authenticate SOAP messages
  - Security tokens assert claims and can be used to assert the binding between authentication secrets and security identities (e.g., X.509 Certificates)
  - The recipient of a security token may choose to accept the claims made in the token based on the endorsement by a trusted authority (e.g., certificate authority) or its trust of the producer of the containing message
  - Signatures are used to verify message origin and integrity, and by message producers to demonstrate knowledge of the key used to confirm the claims in a security token and to bind their identity to the messages they create

WSS 1.1 Message Protection Mechanisms (Cont'd)

- Message Protection (Message Integrity and Confidentiality)
  - Protect the message content from being disclosed or modified without detection
  - Provide a means to protect a message by encrypting and/or digitally signing a SOAP body, a SOAP header, or combination of them (or parts of them)
  - Message integrity is provided by XML Signature in conjunction with security tokens to ensure that modifications to messages are detected. Multiple signatures by multiple SOAP actors/roles, and extensible to support additional signature formats
  - Message confidentiality leverages XML Encryption in conjunction with security tokens to keep portions of a SOAP message confidential. Support additional encryption processes and operations by multiple SOAP actors/roles

- Invalid or Missing Claims
  - A message recipient SHOULD reject messages containing invalid signatures, messages missing necessary claims or messages whose claims have unacceptable values
  - An example of a security claim is the identity of the producer; the producer can claim that he is Bob, known as an employee of some company, and therefore he has the right to send the message

WSS 1.1 Security Tokens

- Security tokens are security information included in the message that are typically used for authentication or authorization purposes
- WSS 1.1 defines three security token profiles
  - Username Token (e.g., username/password)
  - Binary Token (e.g., X.509 certificates)
  - XML Token (e.g., SAML assertions)
- Each security token has different wrapper elements:
  - `<wsse:UsernameToken>`
  - `<wsse:BinarySecurityToken>`
  - `<saml:Assertion>`

  There is no `<wsse:SecurityToken>` wrapper element

Security Token Characteristics

- Username Token
  - Old standby username/password option
  - If you have limited value or limited risk data or a user population unwilling or unable to use more sophisticated authentication options
  - Clear-text passwords in SOPA headers is an issue (Use SSL at least)
  - Vulnerable to password guessing, social engineering, dictionary attack, and the etc
- Binary Security Token
  - X.509 v3 certificates (X.509v3) and Kerberos tickets
  - X.509v3 is a digital container for the public key part of a public/private key pair
  - SOAP message's sender is required to provide Proof of Possession – A digital signature with the X.509v3 so that you can verify the sender has access to the corresponding private key
- XML Token
  - SAML Tokens
  - SAML can provider assertions on sender identity, attributes, authentication and authorization
  - The relying party must determine that the sender or the identity being represented by the sender in the situation in which a third party is vouching for the sender really has proof of that sender's identity (via `holder-of-the-key` or `sender-vouches`)

Security Token Examples

Username Token

```
<wsse:UsernameToken>
   <wsse:Username>Zoe</wsse:Username>
   <wsse:Password>IloveDogs</wsse:Password>
</wsse:UsernameToken>
```

Binary Token (X.509 certificate)

```
<wsse:BinarySecurityToken
     wsu:Id="binarytoken"
     ValueType="…#X509v3"
     EncodingType="…#Base64Binary">
   MIIEZzCCA9CgAwIBAgIQEmtJZc0…
</wsse:BinarySecurityToken>
```

XML Token (SAML)

```
<saml:Assertion xmlns:saml="..."
   AssertionID="_a75adf55-01d7-40cc-929f-dbd8372ebdfc"
   IssueInstant="2003-04-17T00:46:02Z"
   Issuer="www.opensaml.org"
   MajorVersion="1"
   MinorVersion="1">
 </saml:Assertion>
```

Table Of Contents

WSS 1.1 Username Token Profile

- Purpose: Document a way of providing a username as a security token
- Within `<wsse:UsernameToken>` element, a `<wsse:Password>` element may be specified
- Two types of `<wsse:Password>`
  - `PasswordText`: information held in the password is “in the clear”
  - `PasswordDigest`: information held in the password is the digest of the information
  - When `PasswordDigest` is used, if the server does not have access to the clear text password, the hash is considered password equivalent
- When use `PasswordDigest` type of a `UsernameToken`, you hash together three items:
  - Time stamp: Add entropy (randomness) to the resulting password digest, and enforce “freshness” constraints on the messages
  - Nonce: An arbitrary set of bytes to prevent replay attacks. The client side generates the nonce and should not send a duplicate nonce within the “freshness” period. WSS recommends nonce be cached by servers during the freshness period
  - Password: clear-text password or shared secret

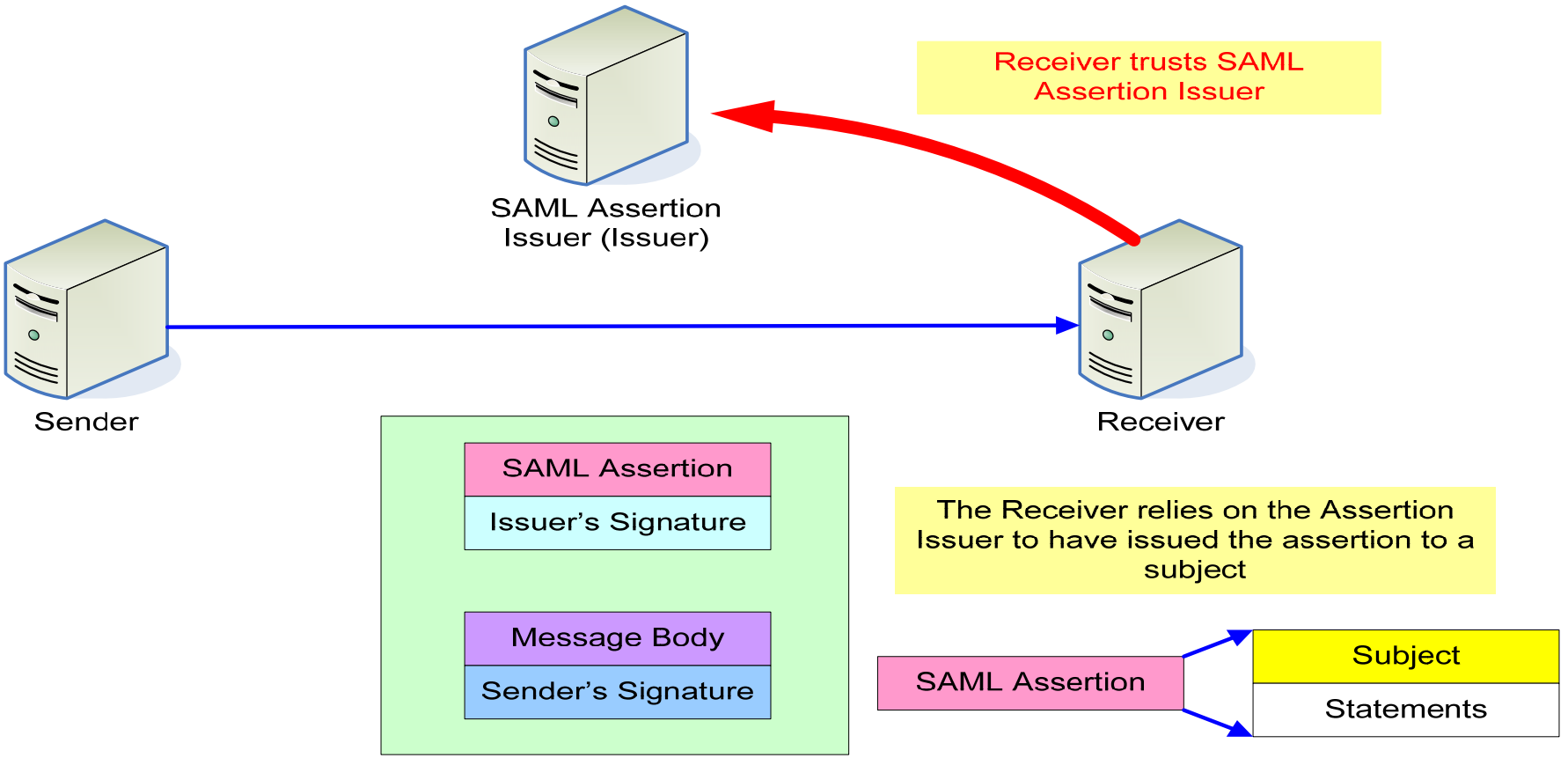
WSS 1.1 X.509 Certificate Profile

- Purpose: Describe the syntax and processing rules for the use of the X.509 authentication framework with the WS-Security specification
- `ValueType` attribute for X.509 v3 certificate: #X509v3
- Token References: the `<wsse:SecurityTokenReference>` element SHALL be used to specify all references to X.509 token types in signature or encryption elements that comply with this profile
- Three reference mechanisms:
  - Reference to a Subject Key Identifier: The `<wsse:SecurityTokenReference>` element contains a `<wsse:KeyIdentifier>` element that specifies the token data by means of a X.509 SubjectKeyIdentifier reference. A subject key identifier may only be used to reference an X.509v3 certificate
  - Reference to a Binary Security Token: The `<wsse:SecurityTokenReference>` element contains a `<wsse:Reference>` element that references a local `<wsse:BinarySecurityToken>` element or a remote data source that contains the token data itself
  - Reference to an Issuer and Serial Number: The `<wsse:SecurityTokenReference>` element contains a `<ds:X509Data>` element that contains a `<ds:X509IssuerSerial>` element that uniquely identifies an end entity certificate by its X.509 Issuer and Serial Number

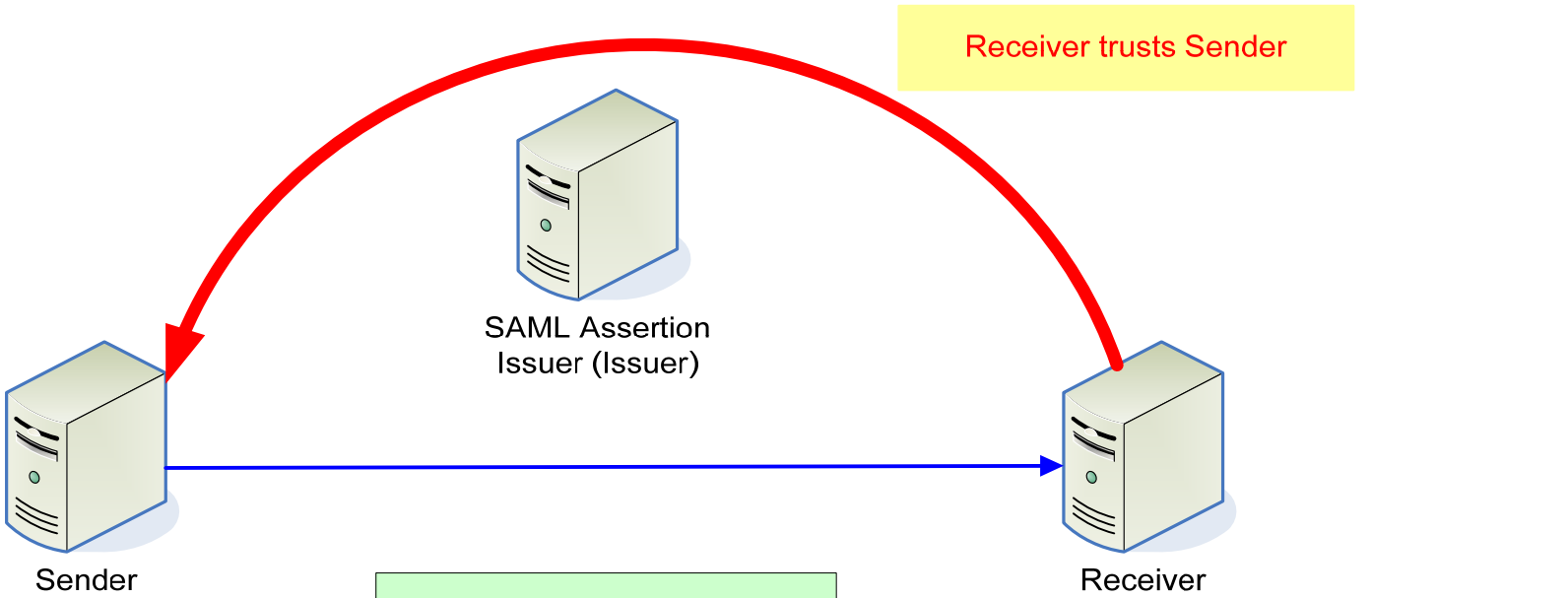
WSS 1.1 SAML Token Profile

- Purpose: Defines the specific mechanisms and procedures for using SAML assertions as security tokens
- Processing Model
  - Actors: Receiver (Responder), Requestor (Sender), SAML Assertion Issuer, and Subject
  - SAML Assertion Issuer issues the SAML assertion to the Subject
  - Requestor (Sender) sends the message with a SAML assertion in the SOAP header within `<wsse:Security>` element
  - Receiver receives the message, validates the SAML assertion and the message, and processes the message
  - **Receiver must be able to determine that the sender – or the identity being represented by the sender in the situation in which a third party is vouching for the sender – really has proof of that sender's identity**
- A SAML assertion contains a collection of zero or more statements

SAML Subject Confirmation – `holder-of-key`



SAML Subject Confirmation – sender-vouches



SAML Subject Confirmation Comparison

	Holder-of-Key	Sender-Vouches
Basis of Trust	The Receiver relies on the Assertion Issuer to have issued the assertion to a subject Receiver trusts SAML Assertion Issuer's certificate	The Receiver relies on the Sender, who vouches for the contents of the User message and the SAML Assertion Receiver trusts Sender's certificate
Validation Mechanism	Receiver verifies SAML assertion using Assertion issuer's certificate Receiver verifies message body using Sender's certificate	Receiver verifies SAML assertion using Sender's certificate Receiver verifies message body using Sender's certificate

SAML Subject Confirmation Exercise – Applying a Credit Card

- Basic Scenario
  - John Smith applies a credit card from the Bank of US. John told the bank that his FICO score is 800
- Exercises
  - In the above scenario, identify the following:
    - Sender/Requestor
    - Receiver/Responder
    - SAML Issuer
  - Who is the “Subject”?
  - Describe the processing scenario using `holder-of-key`
  - Describe the processing scenario using `sender-vouches`
  - For each scenario, discuss benefits and risks

SAML Subject Confirmation Exercise – Selling a Used Car #1

- Basic Scenario
  - Alice is selling a used car. Bob sees the car and wants to buy it. Bob gives Alice a personal check of $8000 signed by Bob to pay for the car
- Exercises
  - In the above scenario, identify the following:
    - Sender/Requestor
    - Receiver/Responder
    - SAML Issuer
  - Who is the “Subject”?
  - Describe the processing scenario using `holder-of-key`
  - Describe the processing scenario using `sender-vouches`
  - For each scenario, discuss benefits and risks

SAML Subject Confirmation Exercise – Selling a Used Car #2

- Basic Scenario
  - Alice is selling a used car. Bob sees the car and wants to buy it. Bob gives Alice a bank certified check of $8000 to pay for the car
- Exercises
  - In the above scenario, identify the following:
    - Sender/Requestor
    - Receiver/Responder
    - SAML Issuer
  - Who is the “Subject”?
  - Describe the processing scenario using `holder-of-key`
  - Describe the processing scenario using `sender-vouches`
  - For each scenario, discuss benefits and risks

SAML Subject Confirmation Exercise – Selling a Used Car #3

- Basic Scenario
  - Alice is selling a used car. Bob sees the car and wants to buy it. Bob gives Alice a personal check from Carol of $8000 signed by Carol to pay for the car
- Exercises
  - In the above scenario, identify the following:
    - Sender/Requestor
    - Receiver/Responder
    - SAML Issuer
  - Who is the “Subject”?
  - Describe the processing scenario using `holder-of-key`
  - Describe the processing scenario using `sender-vouches`
  - For each scenario, discuss benefits and risks

SAML Subject Confirmation Exercise – Selling a Used Car #4

- Basic Scenario
  - Alice is selling a used car. Bob sees the car and wants to buy it. Bob gives Alice a bank certified check under Carol's name of $8000 to pay for the car
- Exercises
  - In the above scenario, identify the following:
    - Sender/Requestor
    - Receiver/Responder
    - SAML Issuer
  - Who is the "Subject"?
  - Describe the processing scenario using `holder-of-key`
  - Describe the processing scenario using `sender-vouches`
  - For each scenario, discuss benefits and risks

Roles Summary of Selling a User Car Scenarios

	#1 Bob pays with his personal check	#2 Bob pays with a certified check	#3 Bob pays with Carol's personal check	#4 Bob pays with Carol's certified check
Sender/ Requester	Bob	Bob	Bob	Bob
Receiver/ Responder	Alice	Alice	Alice	Alice
SAML Issuer	Bob	Bob's Bank	Carol	Carol's Bank
Subject	Bob	Bob	Carol	Carol
# of Parties	2	3	3	4

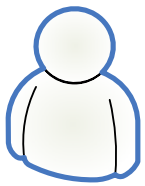
Which one can you trust to give your car?

SAML Subject Confirmation Exercise – Selling a Used Car #5

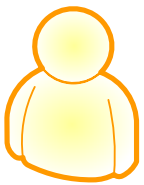
- Basic Scenario
  - Alice is selling a used car. Bob sees the car and wants to buy it. Bob gives Alice a bank certified check under Carol's name of $8000 to pay for the car
  - Alice lives in the same neighborhood with Bob and Carol. Alice knows Carol is Bob's mother. Carol called Alice one day before the transaction said that she will pay the car for Bob.
- Exercises
  - How would the new information impact Alice's decision?

Table Of Contents

WSS Signatures: Multiple XML Signatures

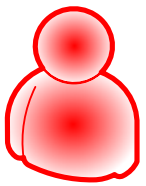


Producer

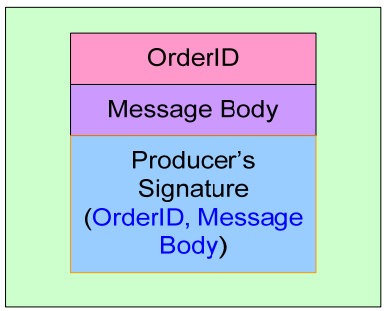


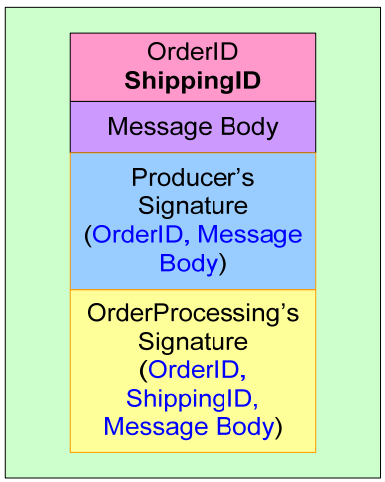
Order Processing

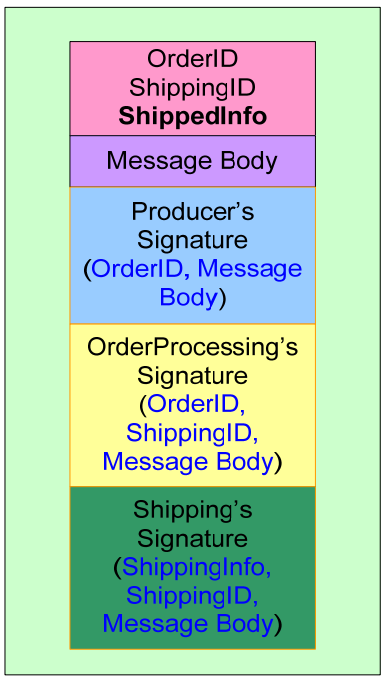
Shipping



Billing







Billing needs to verify everything. How?

Signature Validation and Confirmation

- Signature Validation must FAIL if
  - The syntax of the content of the element does not conform to this specification
  - The validation of the signature contained in the element fails according to the core validation of the XML Signature specification [XMLSIG]
  - The application applying its own validation policy rejects the message for some reason (e.g., the signature is created by an untrusted key – verifying the previous two steps only performs cryptographic validation of the signature)
  - The signature validation shall additionally adhere to the rules defines in signature confirmation section below, if the initiator desires signature confirmation

- Signature Confirmation: Initiator confirms that the message received was generated in response to a message it initiated in its unaltered form
  - Syntax:

```
<wsse11:SignatureConfirmation wsu:Id="..." Value="..." />
```

  - `Value` attribute is the original contents of the `<ds:SignatureValue>` element
  - New in WSS 1.1

WSS Encryption

- Allows encryption of any combination of body blocks, header blocks, and any of these sub-structures by
  - a common symmetric key shared by the producer and the recipient or (Shared Key XML Encryption)
  - a symmetric key carried in the message in an encrypted form (Wrapped Key XML Encryption)
- Characteristics
  - Leverages XML Encryption standard
  - Describes how the two elements `<xenc:ReferenceList>` and `<xenc:EncryptedKey>` can be used within the `<wsse:Security>` header block
  - When a producer or an active intermediary encrypts portion(s) of a SOAP message, it MUST prepend a sub-element to the `<wsse:Security>` header block
  - Defines an element `<wsse11:EncryptedHeader>` for containing encrypted SOAP header blocks
- XML Encryption vs. WSS Encryption
  - In XML Encryption, `<xenc:EncryptedData>` is used to represent encrypted blocks
  - In WSS Encryption, **`<xenc:ReferenceList>`** is used in the security header to point to the parts of the message that have been encrypted (in addition to the `<xenc:EncryptedData>` element)

Shared Key XML Encryption

- Encrypting the body of a SOAP message using a shared (symmetric) key
- Exar

```
<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>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
          </ds:KeyInfo>
          <xenc:CipherData>
            <xenc:CipherValue>...</xenc:CipherValue>
          </xenc:CipherData>
        </xenc:EncryptedData>
    </S11:Body>
</S11:Envelope>
```

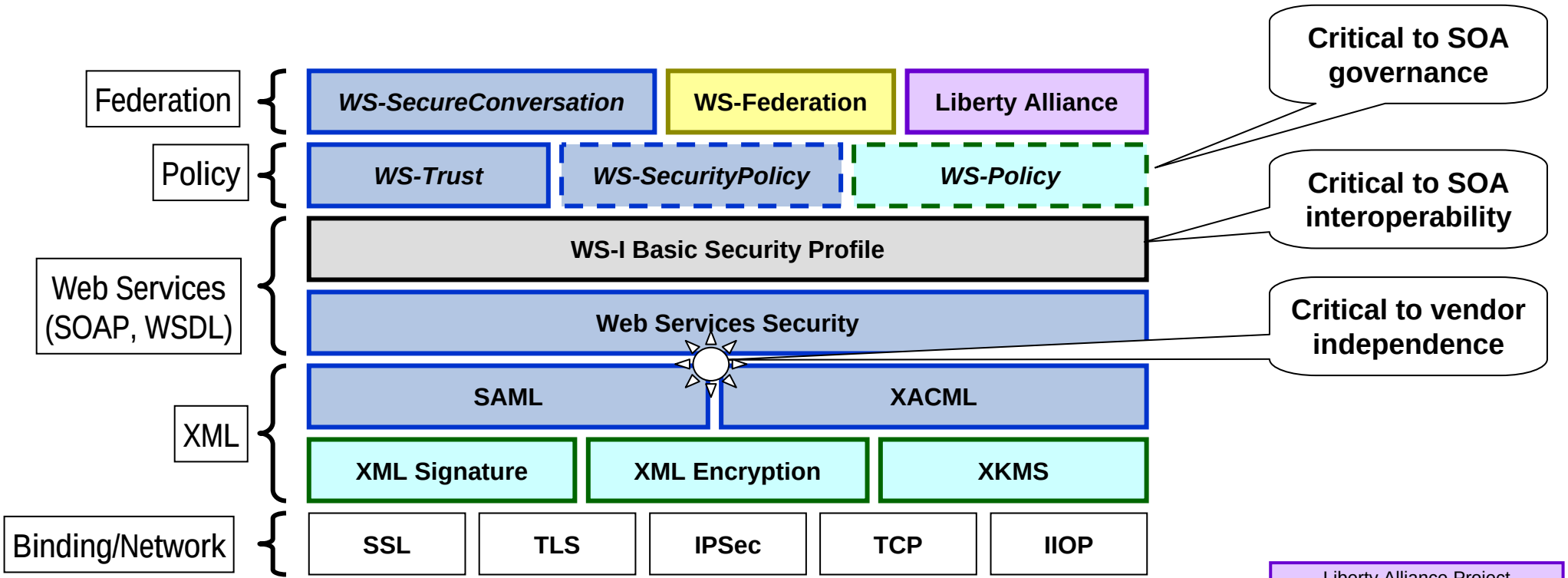
Where is the “encryption key”?

Wrapped Key XML Encryption

- Shared key approach is fast and scalable but a out-of-band key exchange is required
- Wrapped Key XML Encryption (key wrapping or digital enveloping) wraps the shared key using the recipient's public key (also possible with a shared key)
- `<xenc:EncryptedKey>` element is used to wrap keys in the security header
- Example

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
xmlns:ds="..." xmlns:xenc="...">
    <S11:Header>
        <wsse:Security>
            <xenc:EncryptedKey>
              ...
              <ds:KeyInfo>
                <wsse:SecurityTokenReference>
                  <ds:X509IssuerSerial>
                    <ds:X509IssuerName>
                      DC=ACMECorp, DC=com
                    </ds:X509IssuerName>
<ds:X509SerialNumber>12345678</ds:X509SerialNumber>
                  </ds:X509IssuerSerial>
                </wsse:SecurityTokenReference>
              </ds:KeyInfo>
              ...
            </xenc:EncryptedKey>
  ...
        </wsse:Security>
    </S11:Header>
    <S11:Body>
        <xenc:EncryptedData Id="bodyID">
            <xenc:CipherData>
              <xenc:CipherValue>...</xenc:CipherValue>
            </xenc:CipherData>
        </xenc:EncryptedData>
    </S11:Body>
</S11:Envelope>
```

Web Service Security Standards



WS-Security is just one piece

Q&A