

# **Lesson 13 – Securing Web Services (WS-Security, SAML)**

Service Oriented Architectures Security

Module 2 - WS Security

Unit 1 – Auxiliary Protocols

**Ernesto Damiani**

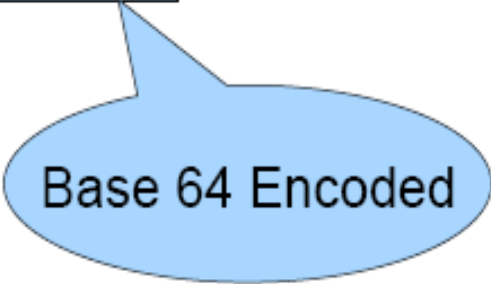
---

Università di Milano

# <CipherData> element

- This element stores or refers to the encrypted data:
  - <CipherValue> container for binary encrypted data

```
<CipherData>  
  <CipherValue>BA234C96D1</CipherValue>  
</CipherData>
```



Base 64 Encoded

- <CipherReference> reference to an URL of the encrypted data. Can include a pipeline of Transform elements like XML Signature, that specify how to filter the referenced data before it is decrypted

# <KeyInfo> element (1)

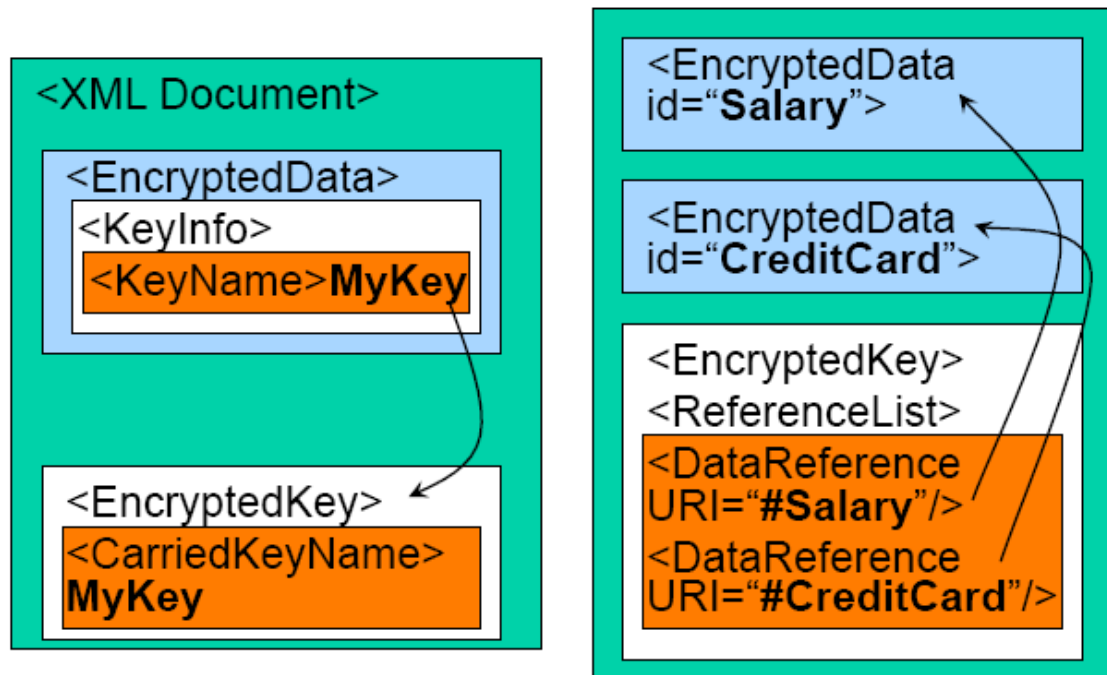
- Describe the key used to encrypt the data
- Whereas in XML Signature, this is usually a public key, in XML Encryption this is usually a shared encryption key
- In general, public keys can be safely included with a message. Instead, it is not safe to embed shared keys!!

## <KeyInfo> element (2)

- XML Encryption provides several mechanisms to agree/retrieve the decryption key:
  - Key is omitted (out-of-band)
  - Key is referenced: <KeyName> <RetrievalMethod>. These elements are used to identify which of the secret keys (shared between the parties) should be used and how the shared key should be retrieved. With them, the same key can be used to encrypt different parts of the same document.
  - Key is regenerated: <AgreementMethod>
  - Key is included in encrypted form: <EncryptedKey>

# Sharing keys within the same message

- It is possible to reuse the same `<EncryptedKey>` element to decrypt multiple `<EncryptedData>` elements



# Using XML Encryption (1)

## Encryption Process

1. Choose an algorithm (3DES, AES)
2. Choose a key and define how to represent it
  - Key is generated or looked up
  - Key is omitted from the message
  - Key is described in the <KeyInfo> section
3. Serialize the XML data to a byte stream
  - Element (with tags)
  - Content (tags omitted)

## Using XML Encryption (2)

4. Encrypt the byte stream
5. Encode the result in the `<CipherData>` element
6. Build the `<EncryptedData>` element with the information required to decrypt it

# Using XML Encryption (3)

## Decryption Process

1. Determine algorithm (3DES, AES)
2. Determine key
  - Key and algorithm could be agreed upon in advance
  - If Key is encrypted, decrypt it (this is recursive)
3. Decrypt key
  - CipherValue (decode the embedded Base-64 byte stream)
  - CipherReference (dereference the URI and apply the specified Transforms before the data is decrypted)



# Using XML Encryption (4)

4. Process XML content: parse the serialized XML and substitute the original `<EncryptedData>` element with the decrypted XML element (or content)
5. Process non-XML content described by the `MimeType` and `Encoding` attributes of the `<EncryptedData>` element

# Using XML Encryption together with XML Signature

# XML Signature and XML Encryption

- Message Confidentiality and Integrity are both important requirements of a secure message exchange
- XML Signature and XML Encryption have been designed to work together to achieve this
- Problem: in which order should they be applied?  
Sign or encrypt first?
  - Encryption metadata is sent in clear. If not signed, encrypted data/metadata could be corrupted by an attacker to prevent decryption of the message.
  - If signatures are sent in the clear, attackers could strip them from a message or replace them entirely without the recipient noticing.

# Example 1: Encrypt the signed data (1)

```
<Document>  
  <Order id="order">  
    <Customer id="1235312">  
      <Address>...</Address>  
    </Customer>  
    <Items>  
  <Item id="Book123"><Price currency="CHF">99</Price></Item>  
    </Items>  
  </Order>
```

```
<Signature>  
  <SignedInfo>  
    <Reference URI="#order">...</Reference>  
  </SignedInfo>  
  <SignatureValue>...</SignatureValue>  
  <KeyInfo><X509Data>...</X509Data></KeyInfo>  
</Signature>
```

```
</Document>
```

# Example 1: Encrypt the signed data (2)

<Document>

```
<EncryptedData id="encryptedData">  
  <CipherText>  
    <CipherValue>...</CipherValue>  
  </CipherText>  
  <KeyInfo>  
    <EncryptedKey>...</EncryptedKey>  
  <KeyInfo>  
</EncryptedData>
```

</Document>

- The signature is hidden inside the encrypted XML
- The order is clear: 1. Decrypt; 2. Verify signature
- **Problem:** the Encryption metadata is not protected with a signature

## Example 2: Sign the encrypted data

<Document>

<EncryptedData id="encryptedData1">

<CipherText>

<CipherValue>...</CipherValue>

</CipherText>

<KeyInfo>

<EncryptedKey>...</EncryptedKey>

<KeyInfo>

</EncryptedData>

<Signature>

<SignedInfo>

<Reference URI="#encryptedData1">...</Reference>

</SignedInfo>

<SignatureValue>...</SignatureValue>

<KeyInfo><X509Data>...</X509Data></KeyInfo>

</Signature>

</Document>

# Decrypt Transform in XML Signature (1)

- When a message is received, it may not be clear in which order signature validation and decryption should be applied
- To make the order of encryption and signature explicit, the Decrypt transform has been added to the XML signature standard
- This transform is used to distinguish whether the signature applies to the <EncryptedData> or to the decrypted data.

```
<Transform Algorithm="...decrypt#XML">  
  <Except URI="#encryptedDataID">  
</Transform>
```

# Decrypt Transform in XML Signature (2)

- The XML Signature processor will decrypt all referenced `<EncryptedData>` elements except the one identified by the `<Except>` element
- With this solution, default processing always applies decryption before signature verification; unless such transform is specified by the sender



# WS-Security



© Scott Adams, Inc./Dist. by UFS, Inc.

# WS-Security overview (1)

- The WS-Security standard applies XML security (XML Encryption and XML Signature) to implement secure SOAP message exchange across multiple and independent trust domains
- **Goals:** security at the message level (end-to-end)
- **Solution:** apply encryption and signatures within a SOAP message independent of the transport. Parts of the message body can be encrypted, signatures are stored in the header

# WS-Security overview (2)

- WS-Security features support for:
  - multiple signature technologies
  - multiple encryption technologies
  - multiple security token formats
- OASIS standard, April 2004

# Message Security vs. Transport Security (1)

## Message Security

- Disadvantages

- Immature standards only partially supported by existing tools
- Securing XML is complicated

- Advantages

- Different parts of a message can be secured in different ways.
- Asymmetric: different security mechanisms can be applied to request and response
- Self-protecting messages (Transport independent)

# Message Security vs. Transport Security (2)

## Transport Security

- Disadvantages

- Point 2 Point: the complete message is in clear after each hop
- Symmetric: request and response messages must use same security properties
- Transport specific

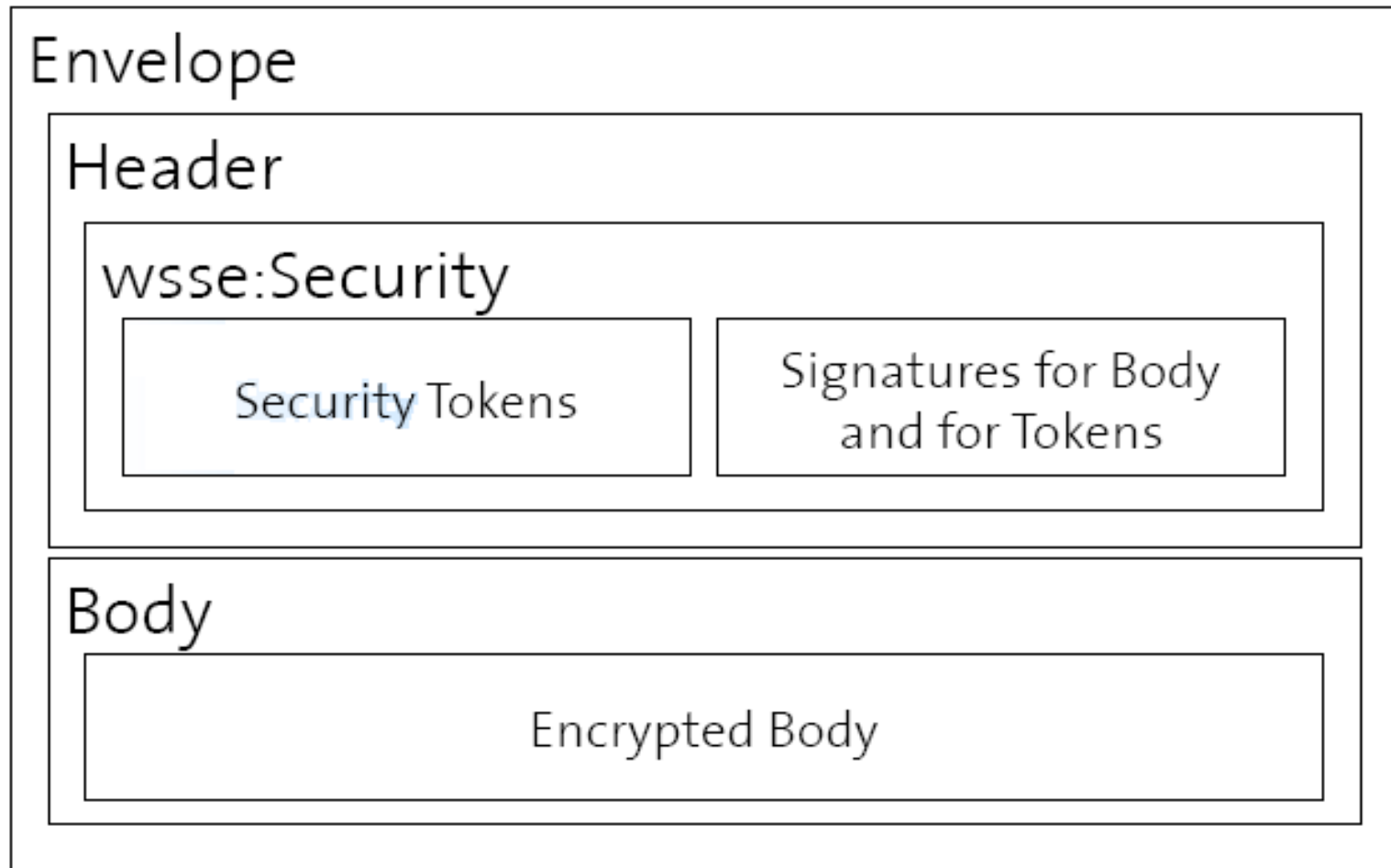
- Advantages

- Widely available, mature technologies (SSL, TLS, HTTPS)
- Understood by most system administrators

# Protecting SOAP messages

- Security Threats to a SOAP message:
  - a message could be read by an attacker
  - a message could be modified by an attacker
  - a message could be sent by an attacker
- To address these threats, WS-Security applies a combination of:
  1. encryption (Ensure the confidentiality of the message)
  2. signatures (Verify the origin and the integrity of a message)
  3. security Tokens (Authorize the processing of the message based on the credentials associated with the message)
- Messages with invalid signatures and incorrect or missing tokens are rejected

# A secure SOAP message



# Security Tokens (1)

- WS-Security supports a variety of authentication and authorization mechanisms by including the corresponding tokens into the Security header of the message:
  - Simple Tokens
    - Username/Clear Password
    - Username/Password Digest
  - Binary Tokens
    - X.509 certificates
    - Kerberos



# Security Tokens (2)

- XML Tokens
  - SAML assertions
  - XrML (eXtensible Rights Markup Language)
  - XCBF (XML Common Biometric Format)
- Token reference
  - WS-SecureConversation

# Security Tokens and Identity (1)

- A security token can be used to claim the identity of the source of a message
- Username/PasswordText is the simplest token used to convey identify but it is also not secure (SOAP messages should not contain passwords in clear)
- Username/PasswordDigest deals with this problem:

```
<UsernameToken>  
  <Username>Scott Tiger</Username>  
  <Password Type="PasswordDigest">XYZAAAQ</Password>  
  <Nonce>123521</Nonce>  
  <Created>2005-11-24T15:00:00Z</Created>  
</UsernameToken>
```

# Security Tokens and Identity (2)

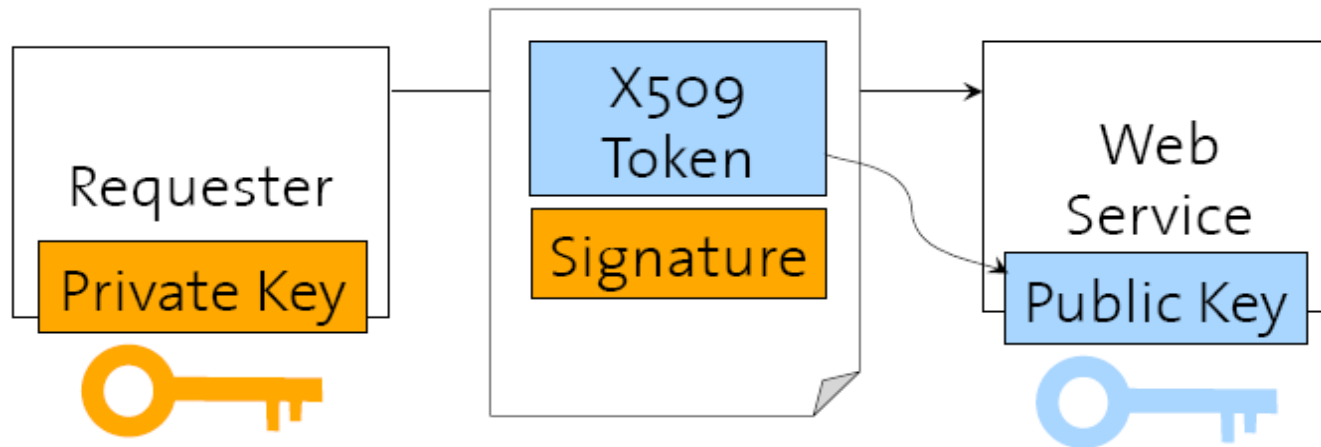
- To produce the digest, the password is hashed together with a timestamp and a nonce
  - Protection against replay attacks
  - The server must store the plain-text password

# Security Tokens and Authentication (1)

- A security token can be signed to authenticate a claim made by the sender of the message
- Signatures associated with tokens can be verified by the recipient to authenticate the identity of the sender

# Security Tokens and Authentication (2)

- Example: X509 certificates (public keys) should be signed in order to provide authentication of the sender (proof of possession of the corresponding private key)



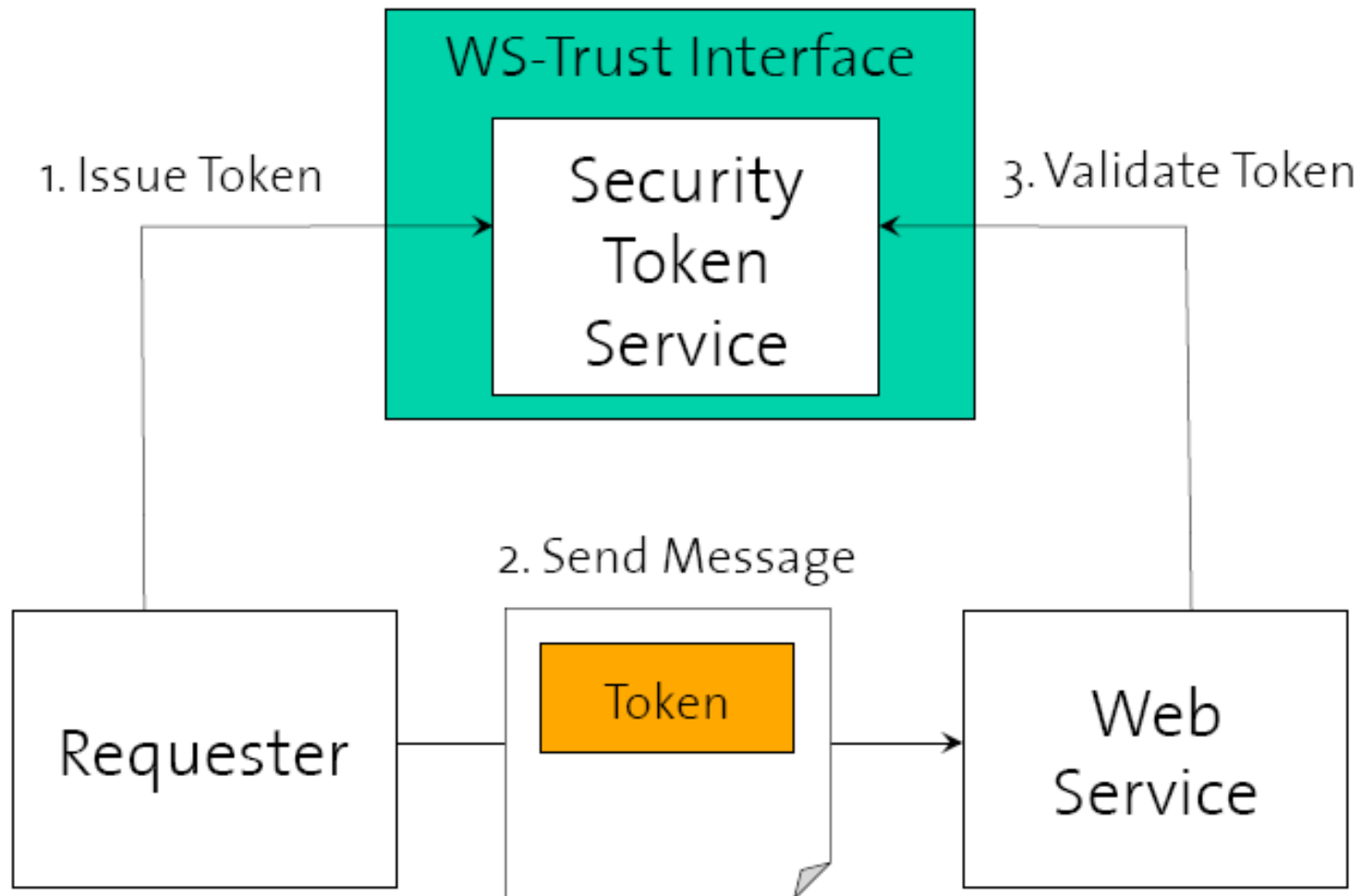
# Federated Security Domains (1)

- Different systems may belong to different security domains that use different security mechanisms and policies
- Although SOAP enables interoperability between these systems, the translation of security metadata between different domains remains a problem
- WS-Security is a first step towards providing standardized syntax and semantics for representing security information

# Federated Security Domains (2)

- WS-Trust adds a standard interface for a security token service provider used to:
  - Issue and Renew Security Tokens to be attached to a SOAP message with WS-Security
  - Validate Security Tokens from a different domain
  - Translate Security Tokens across domains that share a trust relationship (WS-Federation)

# Putting it all together





# WS-SecureConversation (1)

- The security handshake involving the creation of tokens and their validation may impose a high performance overhead
- WS-SecureConversation defines a shared security context to be reused across the exchange of multiple messages
  - The same combination of security credentials (authentication, authorization) and encryption keys can be reused

# WS-SecureConversation (2)

- Once the conversation is established, the requester and the service share a secret:
  - the client does not have to include the security metadata for each message
  - the service does not have to revalidate the same tokens for each message
- This is implemented using a special token:  
<SecurityContextToken>

# **SAML (Security Assertion Markup Language)**

# SAML overview (1)

- The Security Assertion Markup Language (SAML) predates WS-Security, as it was standardized at OASIS in November 2002 (v1.0), August 2003 (v1.1), March 2005 (v2.0)
- **Goal**: enable loosely coupled identity management
- **Solution**: define a format and protocol for interoperable exchange of security information (or assertions) about subjects (human users or computer systems) that have to be identified within a certain security domain

# SAML overview (2)

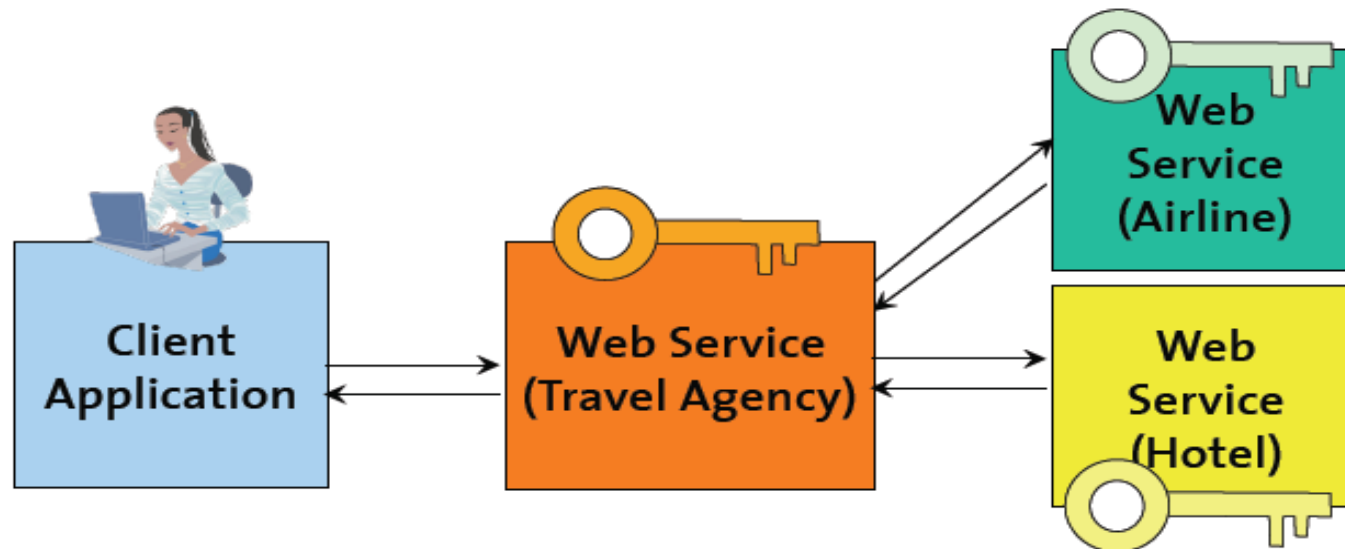
- Use cases supported by standard profiles:
  - Single Sign On (SSO) and Single Logout
  - Identity Federation
  - Privacy-preserving identification
  - Securing Web service messages: SAML assertions are used as WS-Security tokens.
- SAML also defines protocol for clients to request assertions from “SAML authorities” and for services to verify assertions with trusted “SAML authorities”

# Portable and Federated Identity (1)

- SAML enables Single Sign On and the transfer of identity credentials across different trust domains
- Credentials established at the initial service, where the user is authenticated, are forwarded to other services that can trust them

# Portable and Federated Identity (2)

- This is done without a centralized authentication registry that should be shared and trusted by everyone (example: Project Liberty)



# SAML concepts (1)

- SAML uses XML to describe security assertions that can be understood across security domains
- SAML defines a standard protocol to generate, exchange and process assertions
- SAML bindings map how a SAML document is transported:
  - SAML requires HTTPS
  - SAML can be used inside SOAP messages to represent WS-Security tokens.



# SAML concepts (2)

- SAML Assertions and the corresponding protocols are used for:
  - Authentication: verification of identity credentials
  - Attributes: information associated with subjects (e.g., the user address or it's the current balance status of the account)
  - Authorization: grant (or deny) access to a resource for an authenticated subject. (As of SAML 2.0, this feature uses XACML).
  - Custom assertions

# SAML Assertion Metadata Example

```
<Assertion Version="2.0" AssertionID="123042134"  
    IssueInstant="2005-11-23...">  
  <Issuer>saml.ethz.ch</Issuer>  
  <Subject>  
    <NameID Format="emailAddress">  
pautasso@inf.ethz.ch</NameID>  
    <SubjectConfirmation Method="holder-of-key">  
      <SubjectConfirmationData>  
        <ds:KeyInfo>...</ds:KeyInfo>  
      </SubjectConfirmationData>  
    </SubjectConfirmation>  
  </Subject>  
  <Conditions NotBefore="2005-11-23..."  
    NotOnOrAfter="2005-11-24..."><OneTimeUse/>  
  </Conditions>  
  ...statements...  
</saml:Assertion>
```

# Authentication Assertions (1)

- An Authentication Assertion Statement is produced by an authentication authority (issuer) to claim that:
  - a subject (with some identification)
  - with a certain method (or context class)
  - at a certain time

was successfully identified

# Authentication Assertions (2)

- Depending on the method, the authentication assertion can be trusted with a certain level of confidence to represent the digital identity of the subject for some period of time



# Authentication Methods (1)

- To describe how a subject identity was authenticated, SAML 2.0 defines the following authentication context classes:
  - Internet Protocol Address
  - UserName/Password over HTTP or HTTPS
  - Secure Remote Password
  - IP Address and Username/Password
  - SSL/TLS Certificate Based Client Authorization
  - Kerberos Ticket
  - Public Key (X.509, PGP, SPKI, XML Signature)

# Authentication Methods (2)

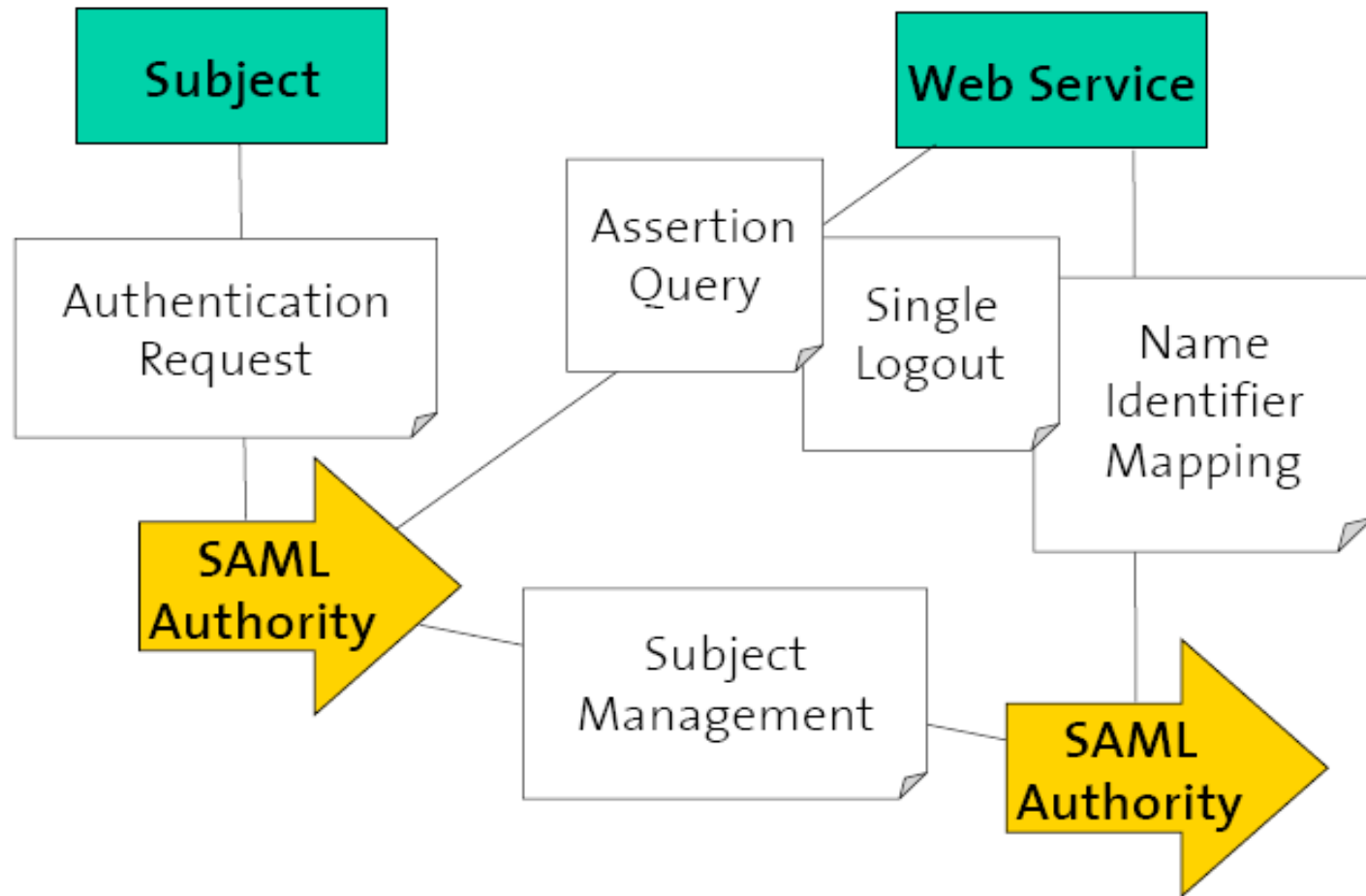
- Telephone Number
- Smartcard: One Factor, Two Factor
- Mobile: One Factor, Two Factor
- Previous Session
- Unspecified

# Attribute Assertions

- An authority asserts that the subject is associated with the specified attributes:
  - SAML profiles show how to apply attributes to standardize access to directories of user attribute information:
    - LDAP/X.500
    - DCE PAC (Privilege Attribute Certificate)
    - XACML (eXtensible Access Control Markup Language)
  - Additionally, attributes can model accounting related information: what is the credit amount left in the account or the payment status for a user

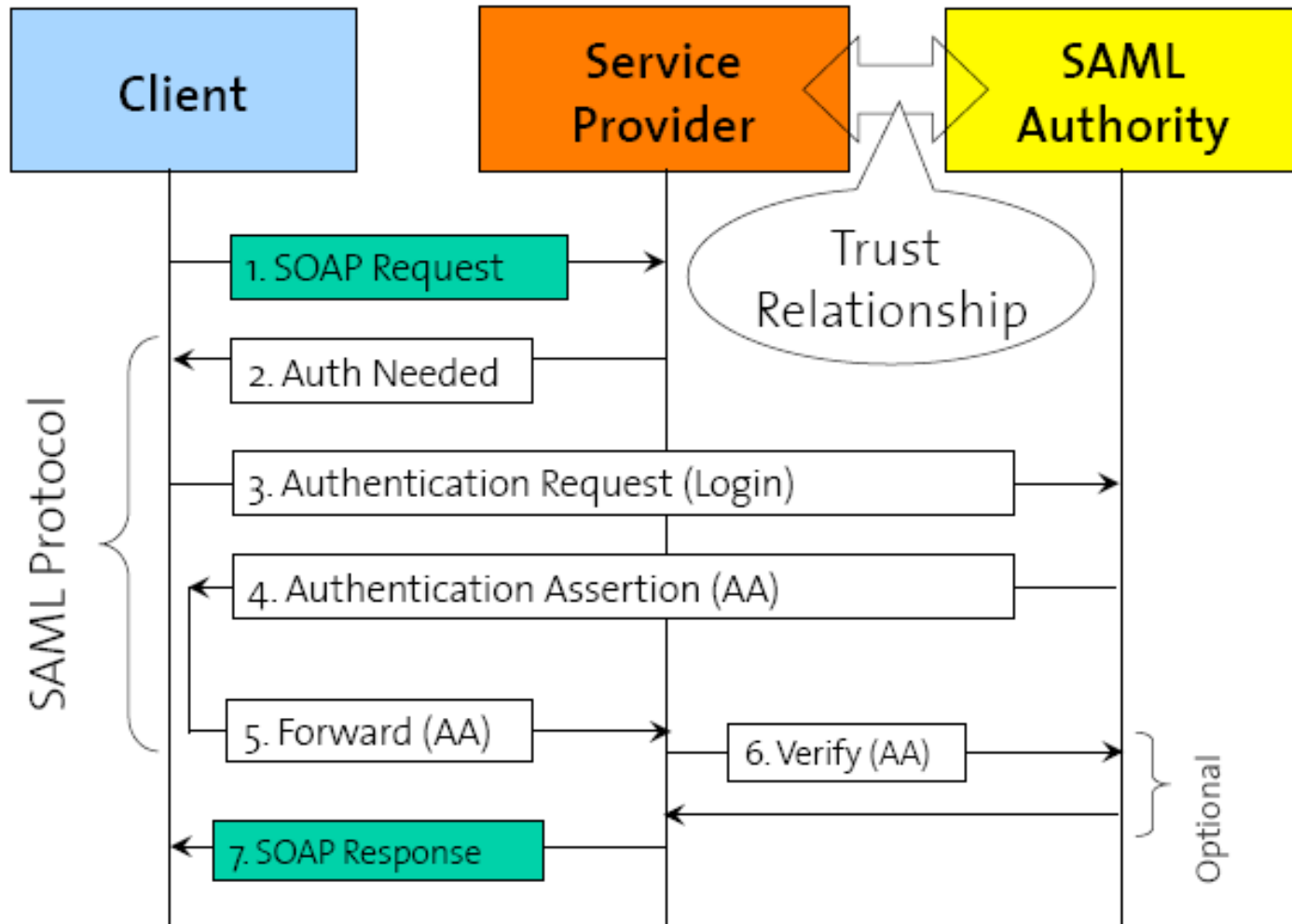


# SAML Protocols





# Putting it all together



# **XACML (eXtensible Access Control Markup Language)**

# XACML overview (1)

- **Goal:** represent access control policies in XML
- **Solution:** define an XML schema for representing authorization rules to grant (or refuse) subjects the access to target resources to perform specific actions.
- **Features:**
  - fine grained control: targets referenced using URLs
  - consistent with and building upon SAML

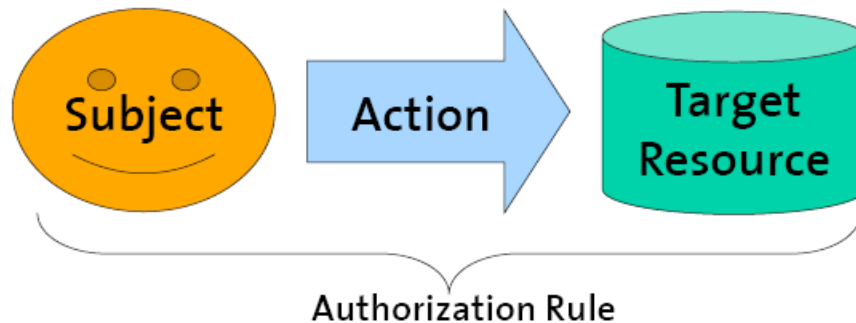
# XACML overview (2)

- **Benefits**

- interoperability of different security tools (Migration of rules through import/export)
  - uniform way to specify access control policies
  - reuse of generic access control service
  - enable the consolidation of access control policies across the enterprise: centralization reduces costs
- OASIS Standard released February 2003 (v1.0), August 2003 (v1.1) and March 2005 (v2.0)

# What is Access Control?

- Authorization is the permission granted to a subject to perform some action on some target resource



- Rights management tools control whether a subject is granted the authorization rights
- Access rights can be granted to individual subjects, but also to groups of subjects (or roles)

# XACML Rule Example (Simplified)

```
<Rule RuleId="1" Effect="Permit">
  <Description>Allow Daniel to send a message</Description>
  <Target>
    <Subjects>
      <Subject><SubjectMatch MatchID="string-equal">
        <AttributeValue>Daniel</AttributeValue>
        <SubjectAttributeDesignator AttributeId="subject-id"/>
      </SubjectMatch></Subject>
    </Subjects>
    <Resources><Resource><ResourceMatch MatchID="anyURI-equal">
      <AttributeValue>uri:message</AttributeValue>
      <ResourceAttributeDesignator AttributeId="resource=id"/>
    </ResourceMatch></Resource></Resources>
    <Actions><Action><ActionMatch MatchID="string-equal">
      <AttributeValue>send</AttributeValue>
      <ActionAttributeDesignator AttributeId="action-id"/>
    </ActionMatch></Action></Actions>
  </Target>
</Rule>
```

# XACML architecture

- XACML works together with SAML to implement an authorization authority

