



8

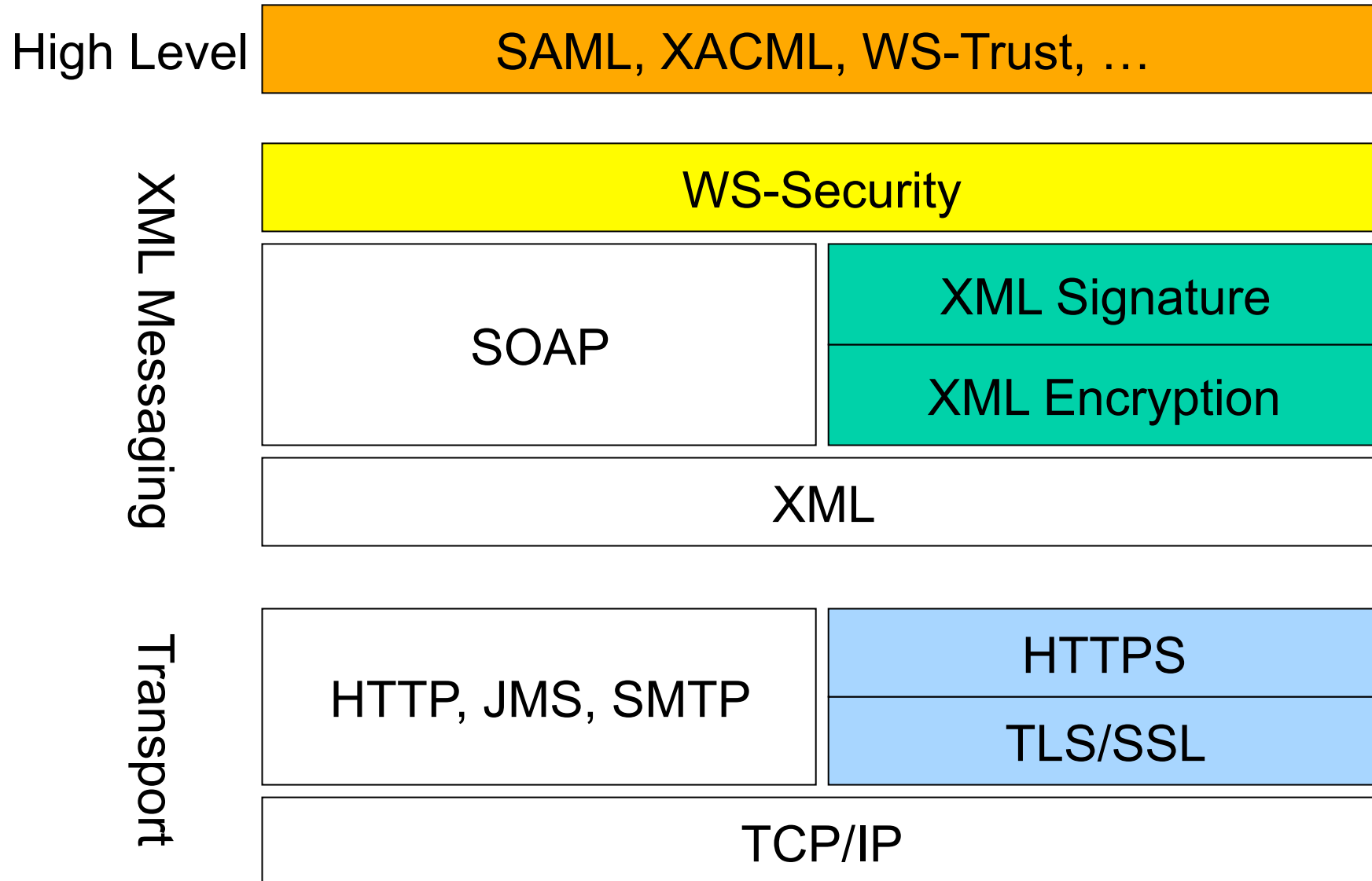
Securing Web Services (WS-Security, SAML)

Gustavo Alonso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
alonso@inf.ethz.ch
<http://www.iks.inf.ethz.ch/>

Web Services Security Standards



Security Standards Overview



Security Standards Stack

WS-Authorization	XACML
WS-SecurityPolicy	
WS-SecureConversation	XKMS
WS-Federation	SAML
WS-Trust	
WS-Security	
SOAP	

Main Security Specifications

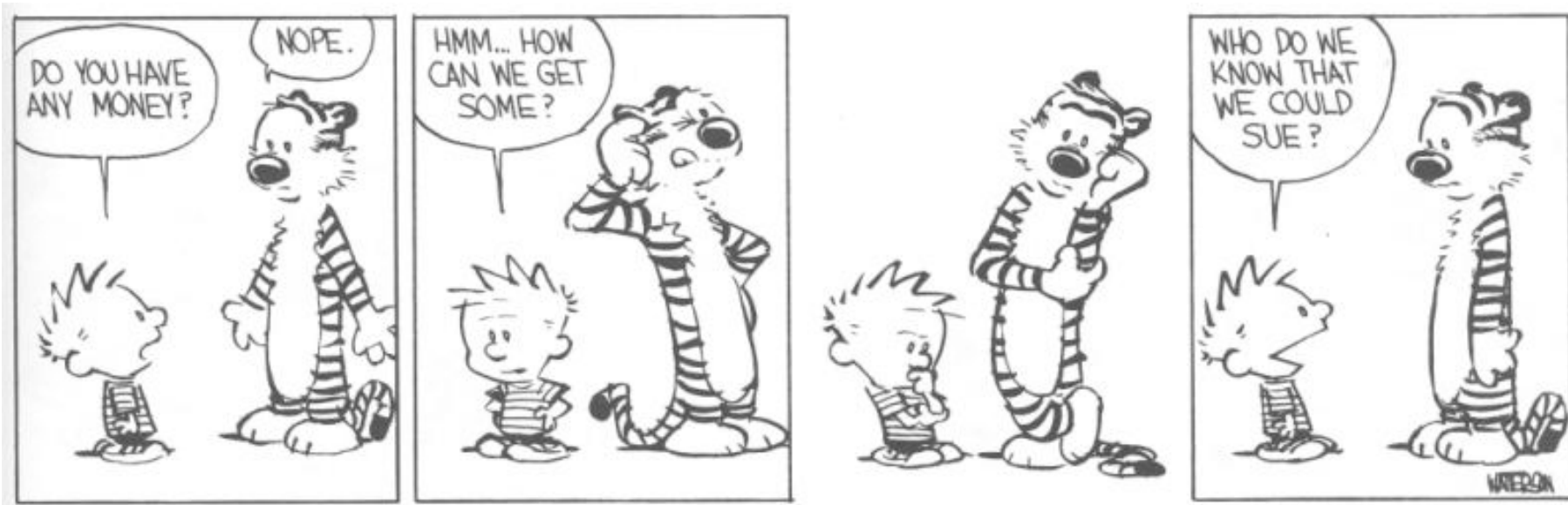
- ❑ XML Signature (XMLDSIG)
 - Message Integrity and Sender/Receiver Identification
- ❑ XML Encryption (XMLENC)
 - Message Confidentiality
- ❑ WS-Security (WSS)
 - Securing SOAP Messages
- ❑ SAML
 - Interoperable security metadata exchange
- ❑ XACML
 - Access Control

Other Security Specifications

- ❑ WS-Trust and WS-Federation
 - Federating multiple security domains
- ❑ WS-SecureConversation
 - Securing multiple message exchanges
- ❑ WS-SecurityPolicy
 - Describing what security features are supported or needed by a Web service
- ❑ XrML
 - Digital Rights Management
- ❑ XKMS
 - Key Management and Distribution



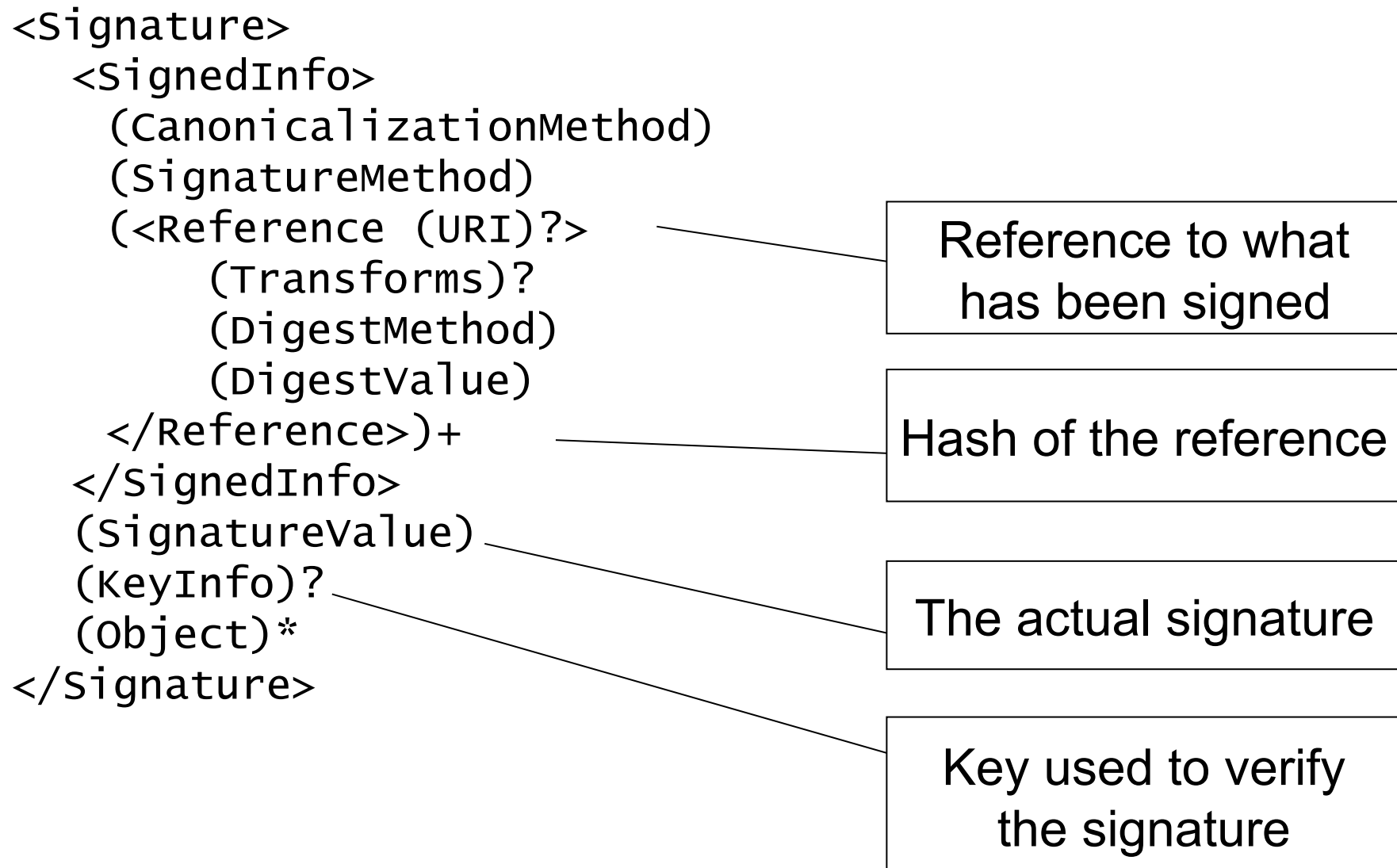
XML Signature



XML Signature Overview

- ❑ **Goals:** Ensure **integrity** of XML messages; identify their source/destination; ensure non-repudiation.
- ❑ XML signature prescribes how to compute, store and verify the digital signature of:
 - entire XML documents
 - parts of XML documents
 - “anything that can be referenced from an URL”, this includes non-XML objects, such as Images.
- ❑ Complex and flexible standard:
 - It is possible to apply multiple signatures over the same XML content
 - Supports a variety of codes and authentication protocols
- ❑ Joint W3C/IETF standard, August 2001

XML Signature Structure



XML Signature Simplified Example

```
<Signature>  
  <SignedInfo>  
    <Reference URI="http://www.google.com"/>  
  </SignedInfo>  
  <SignatureValue>Base-64 encoded </SignatureValue>  
  <KeyInfo>...</KeyInfo>  
</Signature>
```

Generating the signature

- ❑ Reference Generation
 1. Dereference the <Reference URL> to access the XML content that needs to be signed
 2. Apply the Transforms
 3. Compute the <DigestValue> applying the <DigestMethod> to the transformed content
 4. Store the result in the <Reference> element
- ❑ Signature Generation
 1. Create the <SignedInfo> element
 2. Transform it to canonical form
 3. Compute the <SignatureValue> applying a <SignatureMethod>
 4. Bundle it all together with the <KeyInfo> and <Object> elements
- ❑ **Note:** what is actually signed is the <Reference>, which contains a digest (hash) of the original content, which is only indirectly signed.

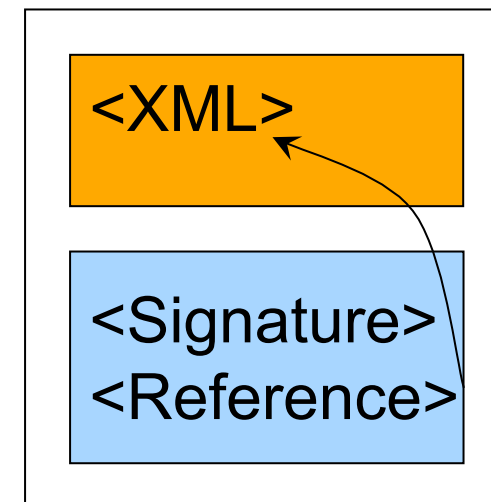
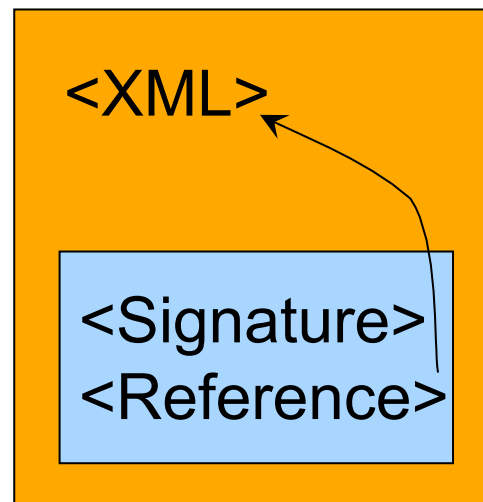
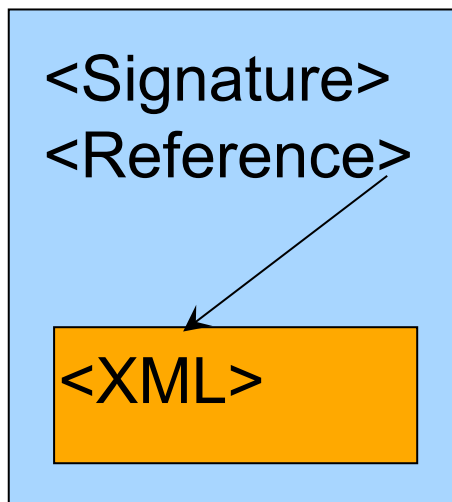
Validating the signature

- ❑ Reference Validation
 1. Dereference the <Reference URL> to access the XML content that needs to be validated against the digest
 2. Apply the same Transforms
 3. Compute a hash using the same <DigestMethod>
 4. **Compare the <DigestValue> with the result.**

- ❑ Signature Validation
 1. Canonicalize the <SignedInfo> element
 2. Get the Key following the <KeyInfo> element
 3. Compute the hash with the <SignatureMethod>
 4. **Compare it with the <SignatureValue>**

XML Signature Position

- ❑ **Enveloping Signature:** the signature wraps the signed element
- ❑ **Enveloped Signature:** the signature is contained inside the signed element
- ❑ **Detached Signature:** the signature refers to a separate element (inside or outside the document)



<Reference> Element

- ❑ The reference element points to the resource that is being digitally signed (URI attribute)
- ❑ There must be at least one Reference element (but more are possible in the same signature)
- ❑ Examples:
 - An element of the same document
URI=“**#CustomerInformation**”
 - The root of the container document
URI=“”
 - An external XML document
URI=“**http://www.swisscom.ch/order.xml**”
 - A fragment of an external document
URI=“**http://www.swisscom.ch/order.xml#Total**”
 - An external non-XML resource
URI=“**http://www.swisscom.ch/order.pdf**”

<Transformation> Element

- ❑ A Reference element contains a set of transform elements, which are applied in a pipelined fashion to the content of the referenced resource
- ❑ The same transformations (in the same order) should be used when generating and validating a digest
- ❑ Standard Transforms:
 - Canonicalization
 - Enveloped Signature Transform
 - Decrypt Transform
- ❑ Optional Transforms:
 - Base-64
 - XPath Filtering
 - XSLT Transform

Canonicalization (C14N)

- ❑ The problem:
 - Signatures are sensitive to single bit changes
 - XML data can have multiple (and equivalent) serializations. Examples:
 - An XML document from a Windows system will use CR+LF, but can still be parsed in UNIX
 - Whitespace can be represented with TAB
 - **Mismatch** between data used by crypto algorithms (raw bytestream: octets) and the XML representation (XML Infoset)
- ❑ The solution:
 - Give a precise (and standard) procedure for producing XML “strings” out of XML infosets.
 - This procedure is called Canonicalization

Canonicalization Example

```
<?xml version="1.0"?>  
<!DOCTYPE doc SYSTEM "doc.dtd">  
< PurchaseOrder >  
  <Customer name = "Swisscom Mobile" />  
  <Date          > 2005 11 22 <      /Date>  
  <!-- Time unknown -->  
  <Items/>  
</ PurchaseOrder>
```

Original XML Document

```
<PurchaseOrder>  
  <Customer name="Swisscom Mobile"/>  
  <Date> 2005 11 22 </Date>  
  <Items></Items>  
</PurchaseOrder>
```

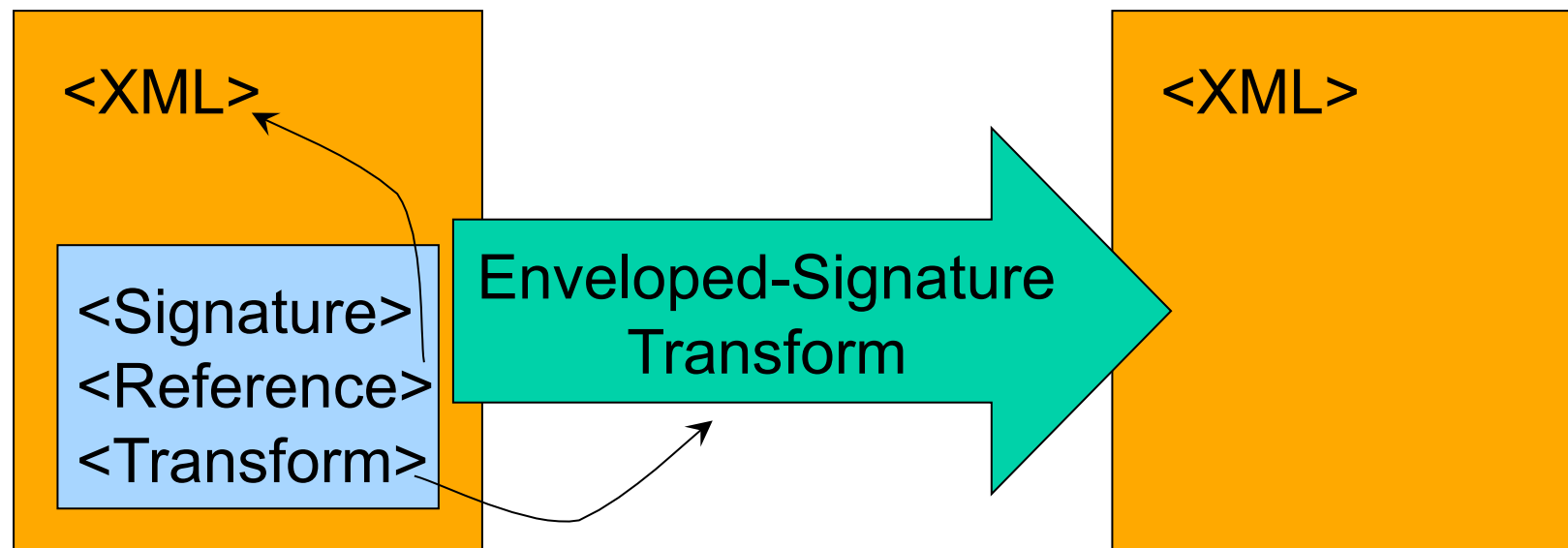
Canonical Form

Some XML Canonicalization Rules

1. UTF-8 encoding
 2. Linebreaks are normalized to LF (ASCII #xA)
 3. Character and entity references are replaced
 4. CDATA sections are replaced with their content
 5. XML declaration and DTD definition are removed
 6. <Empty/> elements converted to <Empty></Empty>
 7. Attribute value delimiters are set to double quotes
 8. Superfluous namespace declarations are removed
 9. Default attributes are explicitly added to elements
 10. Namespace declarations are sorted before the attributes (also sorted)
- ❑ For the whole set of rules, ref:
<http://www.w3.org/TR/xml-c14n>

Enveloped Signature Transform

- ❑ This signature is needed in order to sign an element which is the parent of the <Signature> (Otherwise, the signature should be used as input to compute itself, which makes it impossible to compute)
- ❑ This transform simply removes the <Signature> element from the document



Describing and storing the signature



- ❑ These elements describe how a signature was computed and store its value in encoded format:
 - The <DigestValue> contains the Base-64 encoded value of the digest
 - The <SignatureValue> contains the Base-64 encoded value resulting from encrypting the digest of the <SignatureInfo> element with the key described in the <KeyInfo>
 - The <DigestMethod> describes the algorithm used to compute the <DigestValue> (e.g., SHA1)
 - The <SignatureMethod> describes how the <SignatureValue> was computed (e.g., RSA-SHA1) using the key

<KeyInfo> element

- ❑ The <KeyInfo> provides information about the key used to validate the <SignatureValue>
- ❑ It is quite flexible:
 - The element can be omitted (The parties exchanging the message agree on the key using an out-of-band mechanism)
 - Key is embedded in the message
 - Key is referenced from the message
 - It supports several kinds of Keys used with different cryptographic standards:
 - DSA/RSA
 - X.509 certificates
 - PGP
- ❑ The same element is used in XML Encryption

XML Signature and Security

XML Signature targets these security aspects:

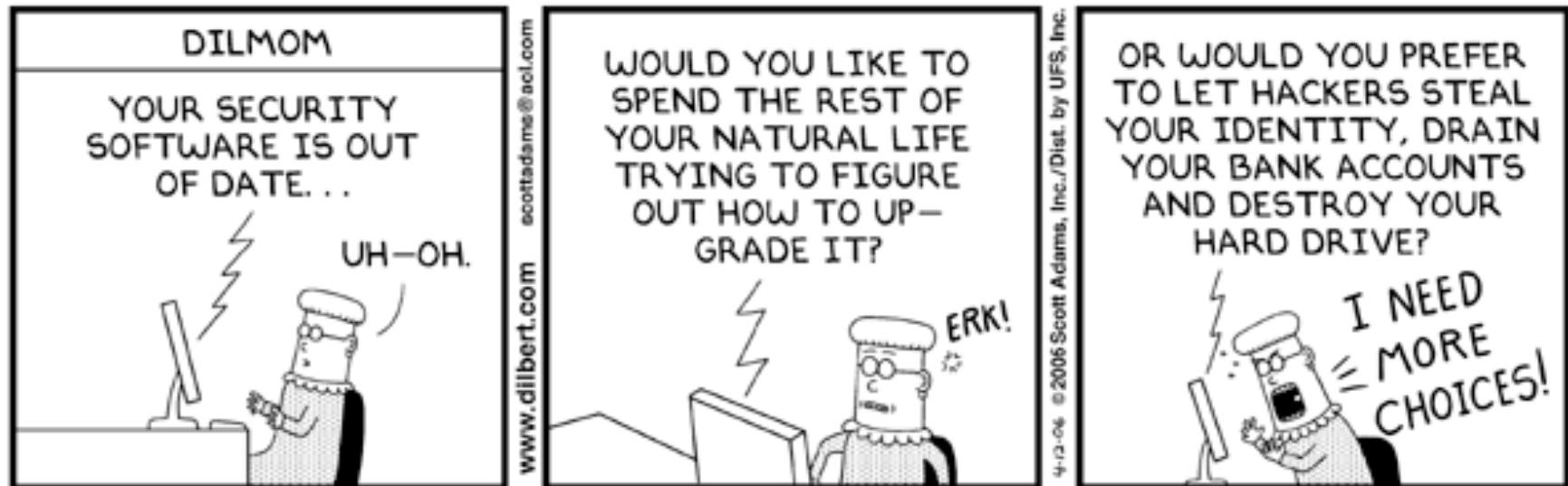
1. Integrity of the message content/external resource:
 - Reference validation
2. Integrity of the signature
 - Signature validation
3. Identity of the source of the document
 - Signature validation
 - Warning: only if using a `<SignatureMethod>` based on public/private key

What you see is what you sign:

- **Transforms** modify and filter the data before it is signed, so they should be used carefully



XML Encryption



XML Encryption Overview

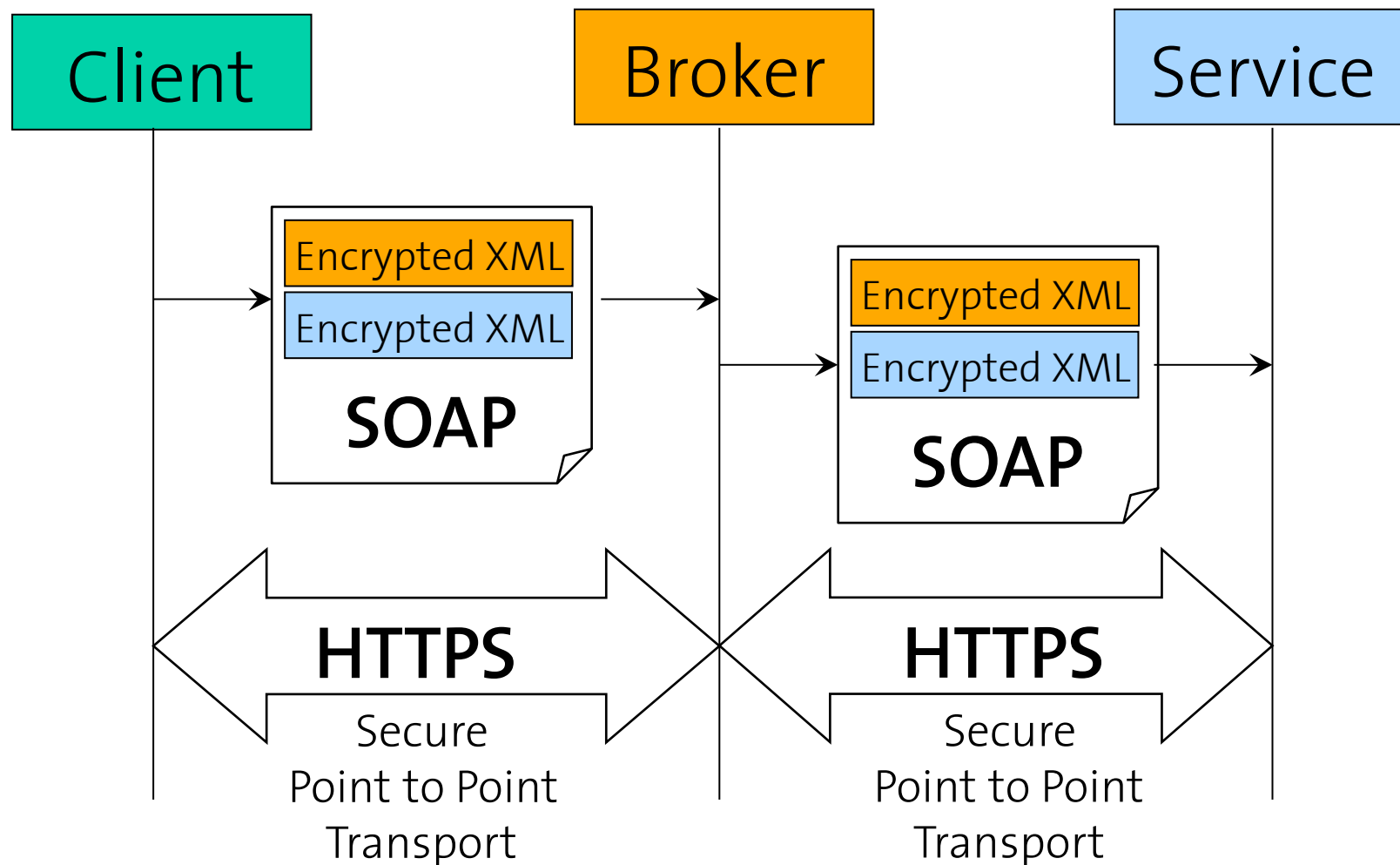
- ❑ Goal: ensure **confidentiality** of XML Messages
- ❑ Solution: obfuscate parts of an XML document, while maintaining a correct XML syntax
- ❑ Features:
 - End to End (Multi-hop scenario)
 - Full or Partial encryption
 - Flexibility: different parts of a message can be read by different parties using different keys
- ❑ Challenges and problems:
 - Is an encrypted XML document still XML?
 - How to validate an encrypted XML document with respect to its XML schema?
- ❑ W3C Recommendation, December 2002

XML Encryption vs. XML Signature

- ❑ XML Encryption complementary to XML Signature
- ❑ Different purposes:
 - XML Encryption = Confidentiality
 - XML Signature = Integrity and Identity
- ❑ Some overlap in the specifications (e.g., <KeyInfo>)
- ❑ Difference:
 - XML Encryption. Encrypted XML is replaced by the <EncryptedData> element
 - XML Signature: Signed XML is referenced from the <Signature> element
- ❑ Warning: Encrypted data which is not signed can still be tampered with!

XML Encryption Scenario

- Guarantee confidentiality at the SOAP message level
(Selected parties may access different message parts)



XML Encryption Example

```
<Employee>  
  <ID>222-654-456</ID>  
  <Name>Markus Bach</Name>  
  <Salary currency="CHF">100000</Salary>  
</Employee>
```

Original XML Document

```
<Employee>  
  <ID><EncryptedData>...</EncryptedData></ID>  
  <Name>Markus Bach</Name>  
  <EncryptedData>...</EncryptedData>  
</Employee>
```

Encrypted XML Document

XML Encryption Structure

```
<EncryptedData Id? Type? MimeType? Encoding?>
```

```
<CipherData>
```

```
<CipherValue?>
```

```
<CipherReference URI?>?
```

```
</CipherData>
```

```
<KeyInfo>
```

```
<EncryptedKey>
```

```
<AgreementMethod>
```

```
<ds:*>
```

```
</KeyInfo>
```

```
<EncryptionMethod/>
```

```
<EncryptionProperties>
```

```
</EncryptedData>
```

Encrypted Value

Reference to
Encrypted Value

Key Information
(extends KeyInfo of
Digital Signature)

Additional Metadata

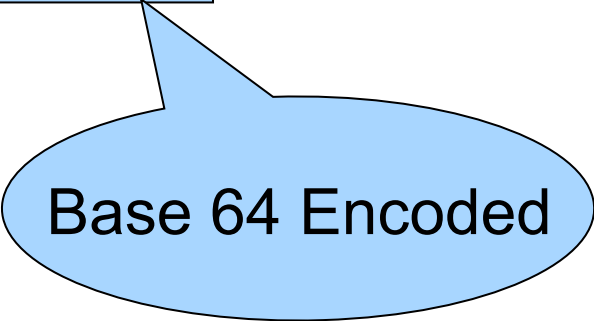
<EncryptedData> Element

- ❑ The <EncryptedData> container tag replaces the document elements that are sent in encrypted form
- ❑ Together with the encrypted elements <CipherData>, it contains metadata and attributes describing how to decrypt them <EncryptionMethod>, <KeyInfo>
- ❑ Attributes:
 - Type = (**element | content**). Determine whether the plaintext is an entire XML element or only the content has been encrypted.
 - MimeType. Optional attribute describing the type of the encrypted non-XML element
 - Encoding. How the non-XML has been encoded
- ❑ The <**EncryptionMethod**> specifies which algorithm has been used to encrypt the data. Currently supported are:
 - Triple-DES
 - AES (Advanced Encryption Standard) with 128, 256 (required) or 192 (optional) bit key

<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

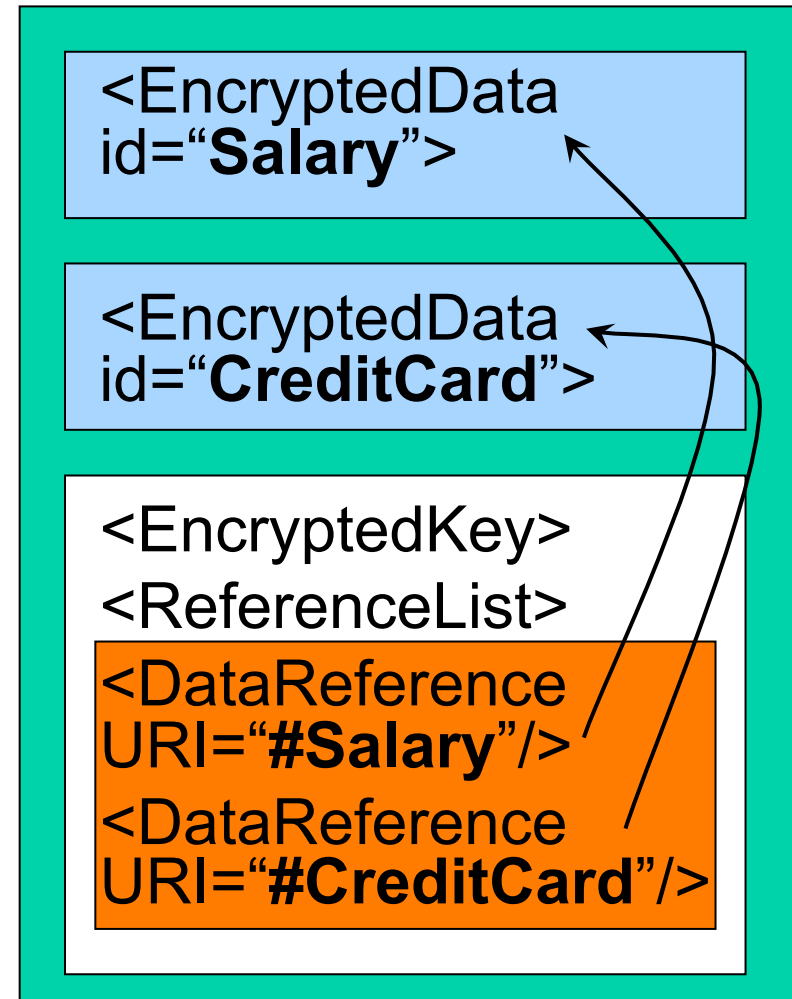
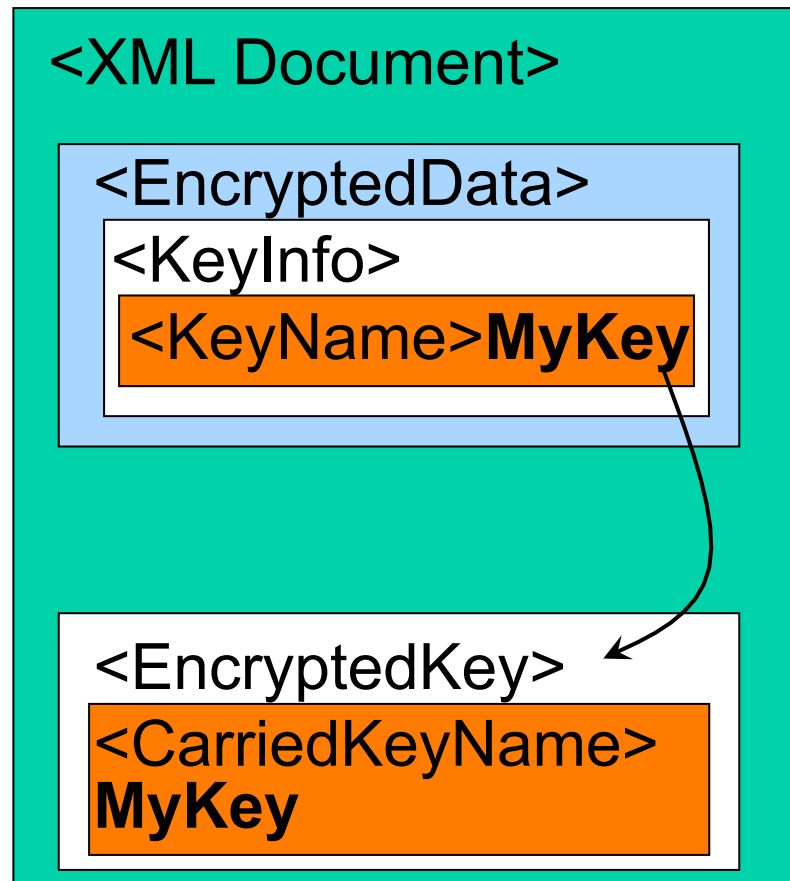
- <CipherReference>
reference to an URL of the encrypted data.
Can include a pipeline of Transform elements like XML
Signature, that specify how to filter the referenced data
before it is decrypted

<KeyInfo> Element

- ❑ 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!!
- ❑ 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

❑ 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)
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 Decryption

❑ 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 data
 - CipherValue (decode the embedded Base-64 byte stream)
 - CipherReference (dereference the URI and apply the specified Transforms before the data is decrypted)
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

```
<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

<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

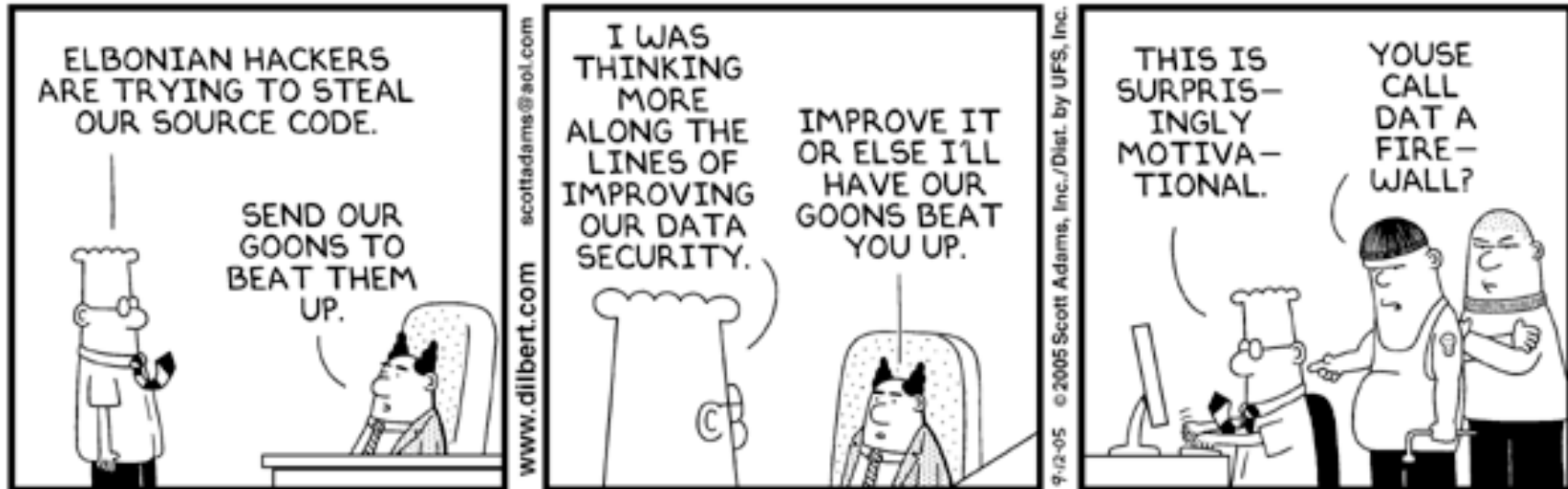
- ❑ When a message is received, it may not be clear in which order signature validation and decryption should be applied.
- ❑ To make the order of encryption and signature explicit, the Decrypt transform has been added to the XML signature standard
- ❑ This transform is used to distinguish whether the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

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

- ❑ 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



WS-Security Overview

- ❑ 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 features support for:
 - Multiple signature technologies
 - Multiple encryption technologies
 - Multiple security token formats
- ❑ OASIS standard, April 2004

Message Security vs. Transport Security



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)

Transport Security

Advantages

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

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

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

Envelope

Header

wsse:Security

Security Tokens

Signatures for Body
and for Tokens

Body

Encrypted Body

Security Tokens

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

Security Tokens and Identity

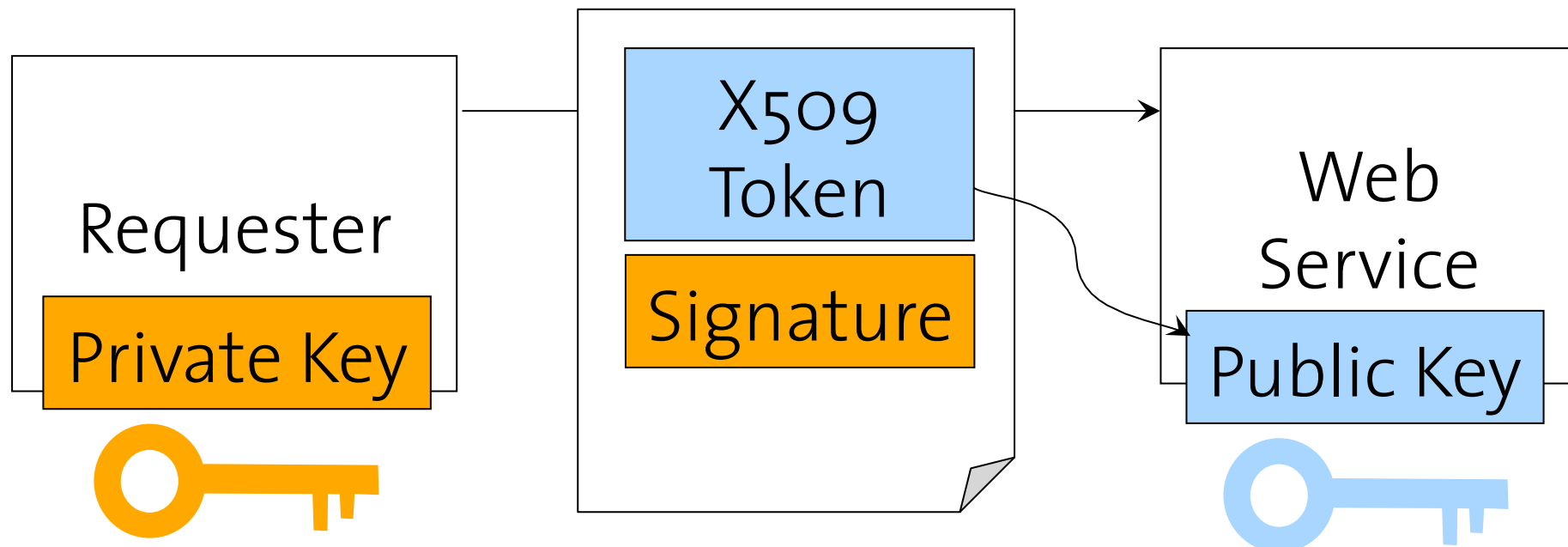
- ❑ A security token can be used to claim the identity of the source of a message
- ❑ Username/PasswordText is the simplest token used to convey identity but it is also **not secure** (SOAP messages should not contain passwords in clear)
- ❑ Username/PasswordDigest deals with this problem:

```
<UsernameToken>  
  <Username>Scott Tiger</Username>  
  <Password Type="PasswordDigest">XYZAAA9</Password>  
  <Nonce>123521</Nonce>  
  <Created>2005-11-24T15:00:00Z</Created>  
</UsernameToken>
```

- ❑ 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

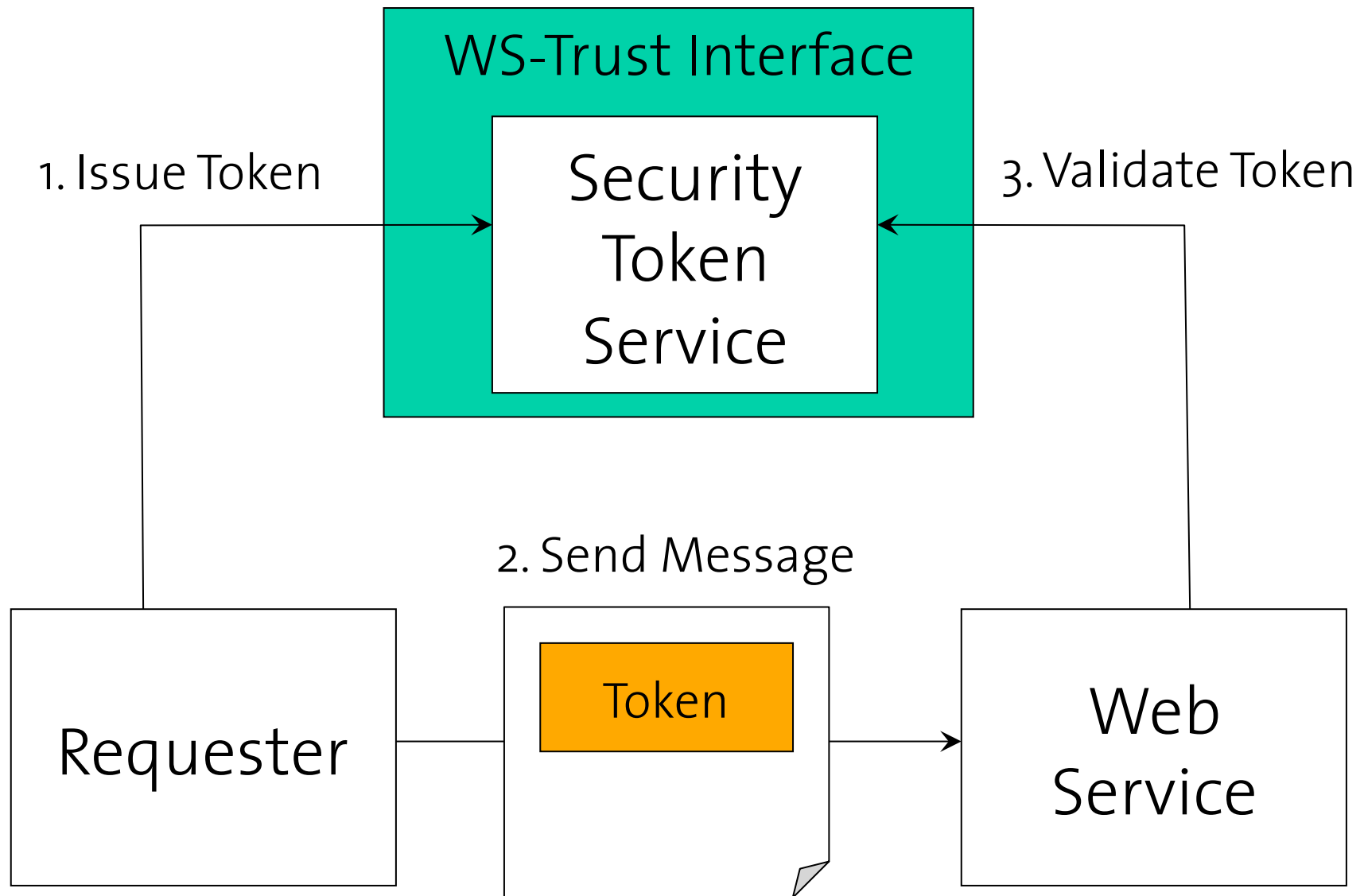
- ❑ A security token can be signed to authenticate a claim made by the sender of the message
- ❑ Signatures associated with tokens can be verified by the recipient to authenticate the identity of the sender.
- ❑ Example: X509 certificates (public keys) should be signed in order to provide authentication of the sender (proof of possession of the corresponding private key)



Federated Security Domains

- ❑ 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.
- ❑ 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

- ❑ 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
- ❑ 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

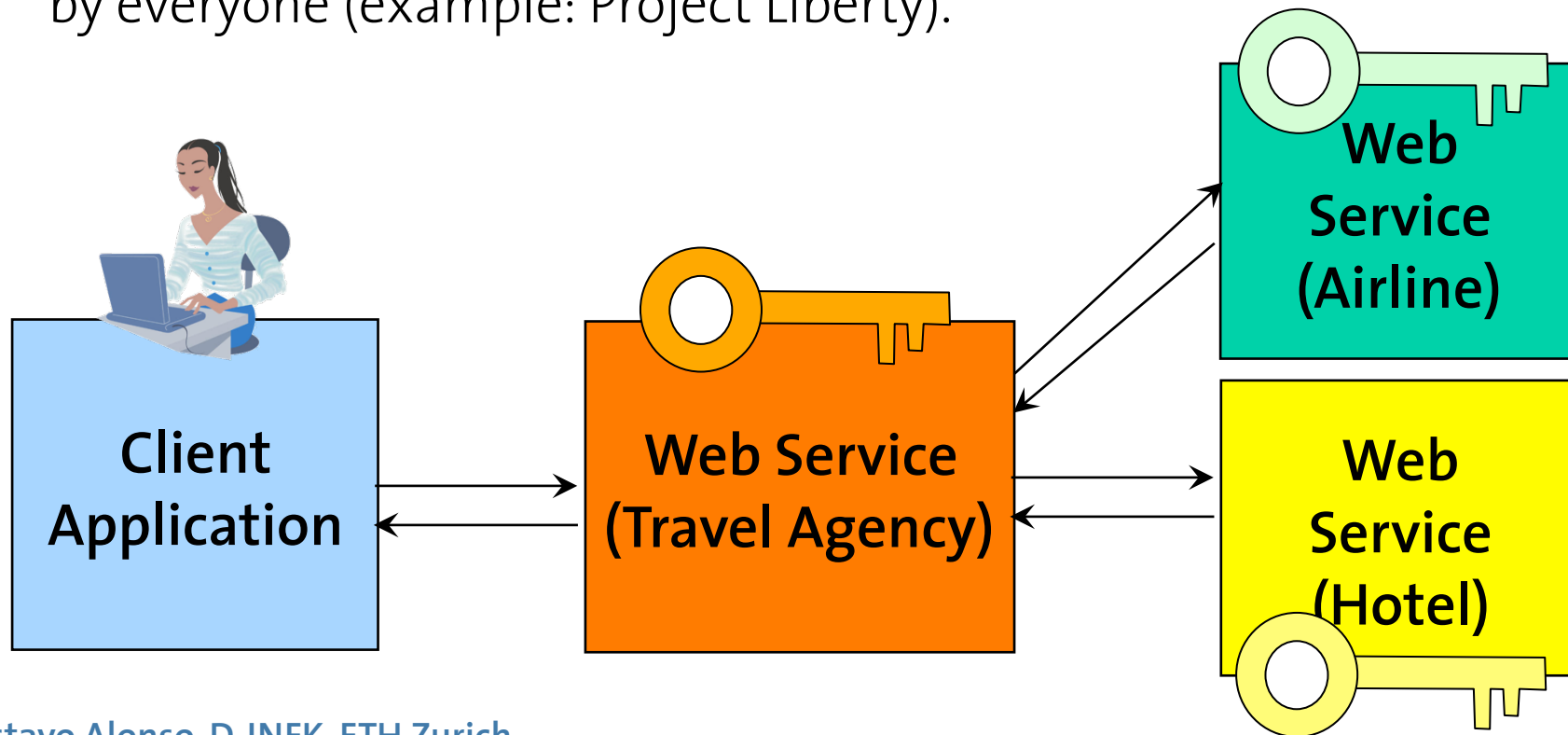
Security Assertion Markup Language

SAML Overview

- ❑ 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.
- ❑ 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

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



SAML Concepts

- ❑ 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.

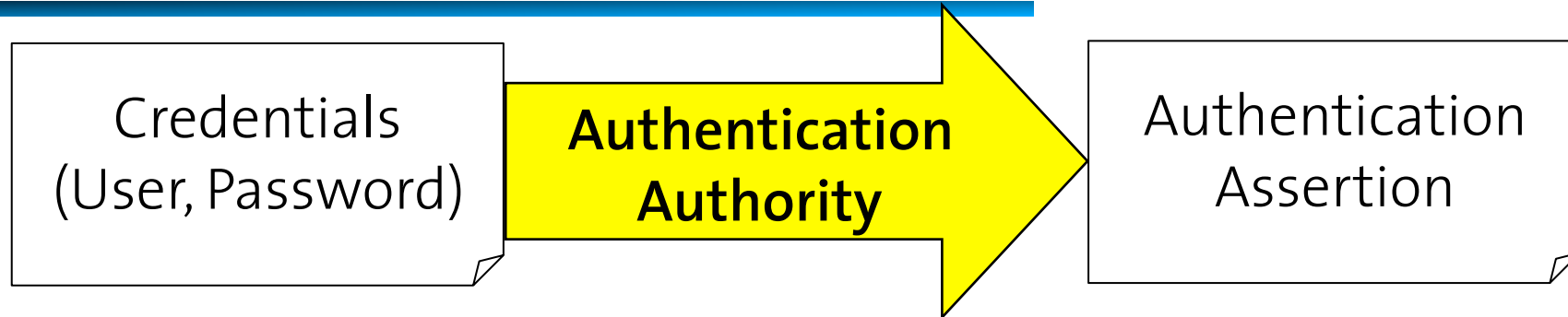
SAML Assertions and the corresponding protocols are used for:

- ❑ Authentication – verification of identity credentials.
- ❑ Attributes – information associated with subjects (e.g., the user address or its 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

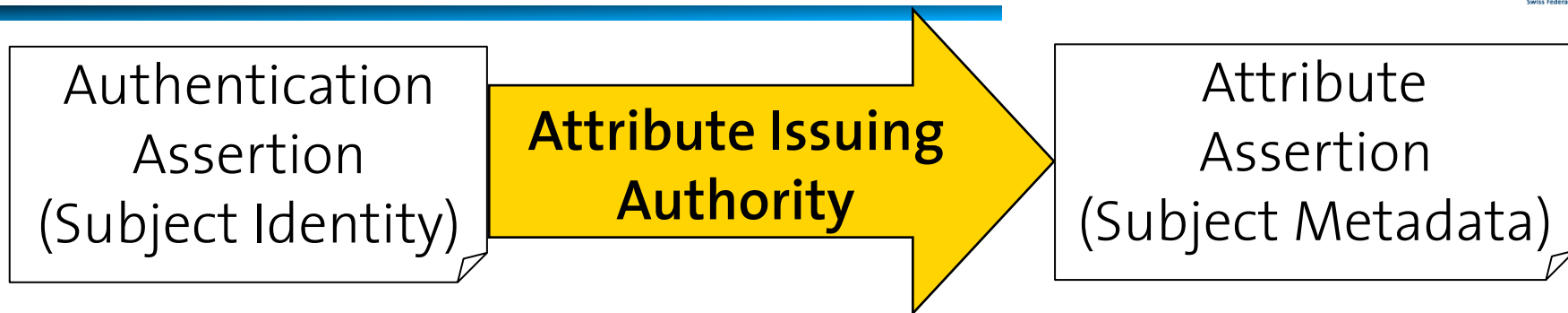


- ❑ An Authentication Assertion Statement is produced by an authentication authority (issuer) to claim that:
 - a subject (with some identification)
 - with a certain method (or context class)
 - at a certain time
- ❑ was successfully identified.
- ❑ Depending on the method, the authentication assertion can be trusted with a certain level of confidence to represent the digital identity of the subject for some period of time

Authentication Methods

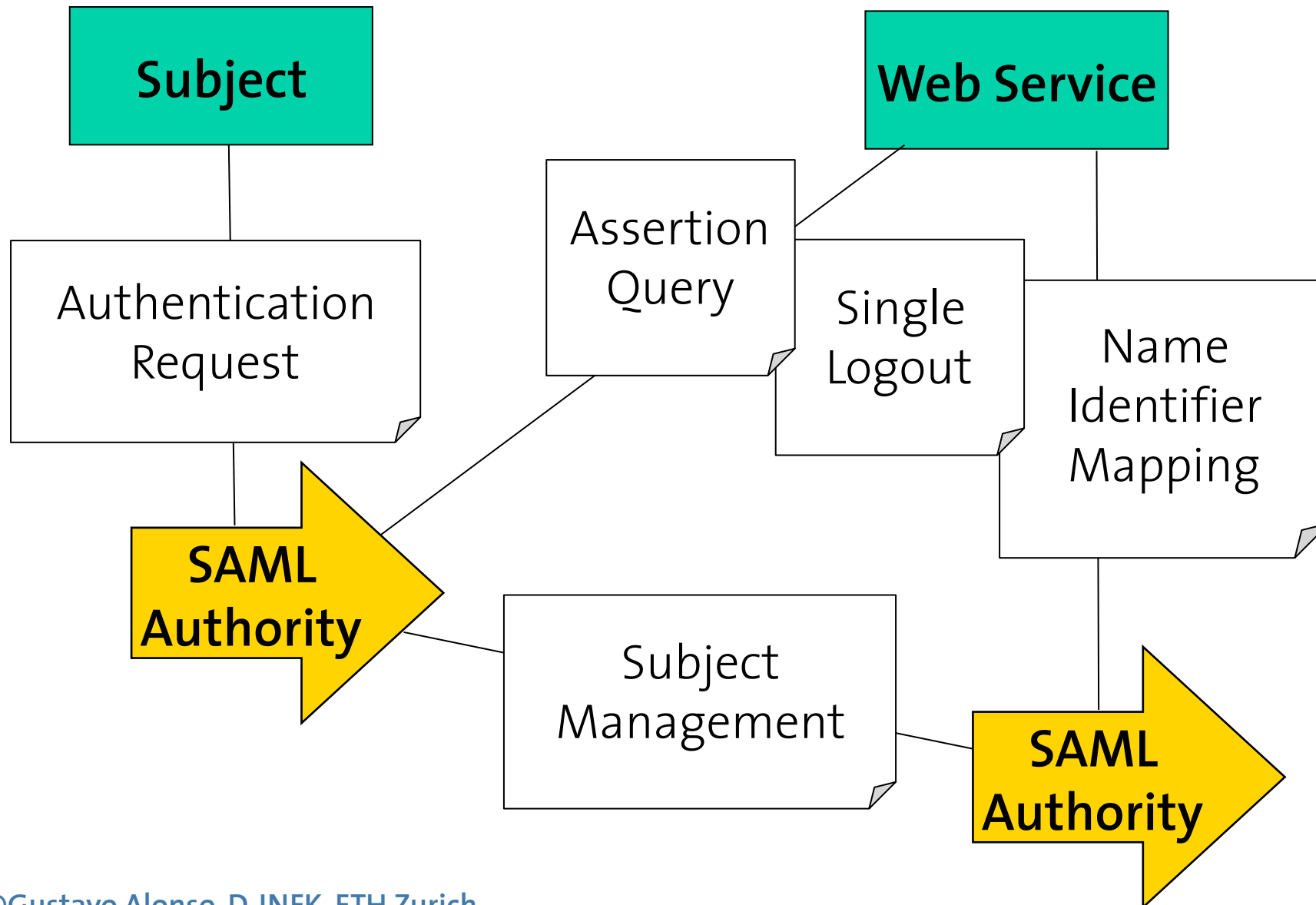
- ❑ To describe how a subject identity was authenticated, SAML 2.0 defines the following authentication context classes:
 - Internet Protocol Address
 - UserName/Password over HTTP or HTTPS
 - Secure Remote Password
 - IP Address and Username/Password
 - SSL/TLS Certificate Based Client Authorization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

Attribute Assertions

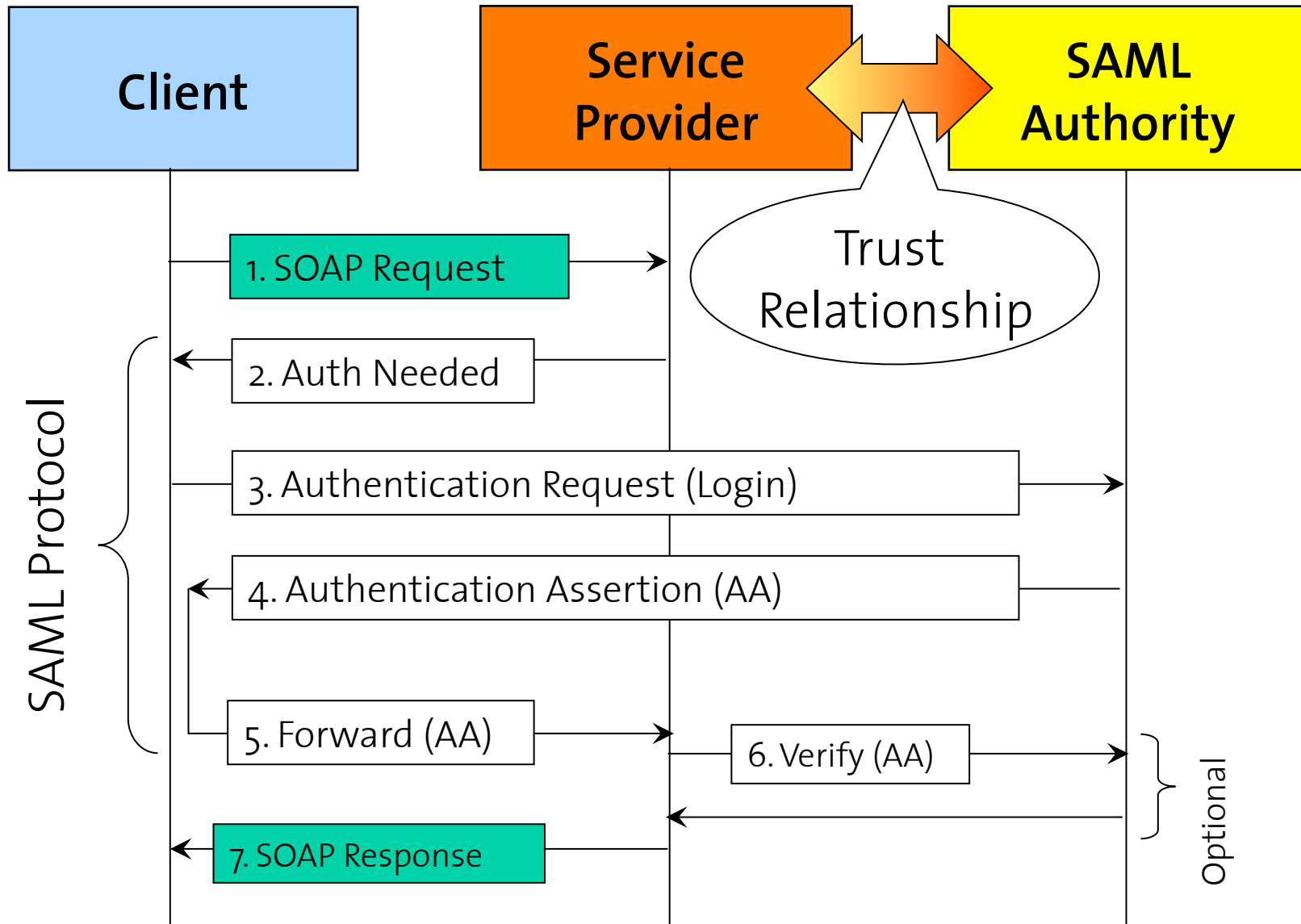


- ❑ An authority asserts that the subject is associated with the specified attributes:
 - SAML profiles show how to apply attributes to standardize access to directories of user attribute information:
 - LDAP/X.500
 - DCE PAC (Privilege Attribute Certificate)
 - XACML (eXtensible Access Control Markup 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





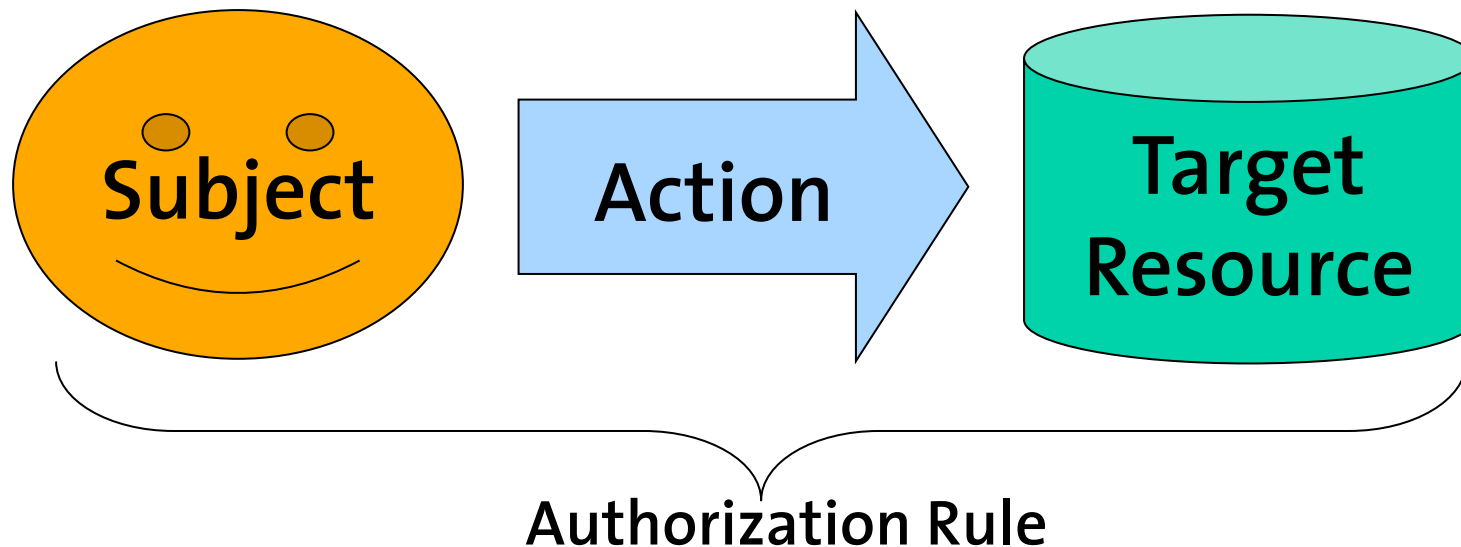
XACML eXtensible Access Control Markup Language

XACML Overview

- ❑ 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
- ❑ 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

