School of Electrical Engineering and Computing

# SENG2250/6250
# SYSTEM AND NETWORK SECURITY
(S2, 2020)

# DSS Part 2
# (Web Service Security)

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

# Outline

- Web Service Security
- Simple Object Access Protocol (SOAP)
- Security Assertion Markup Language (SAML)
- OAuth

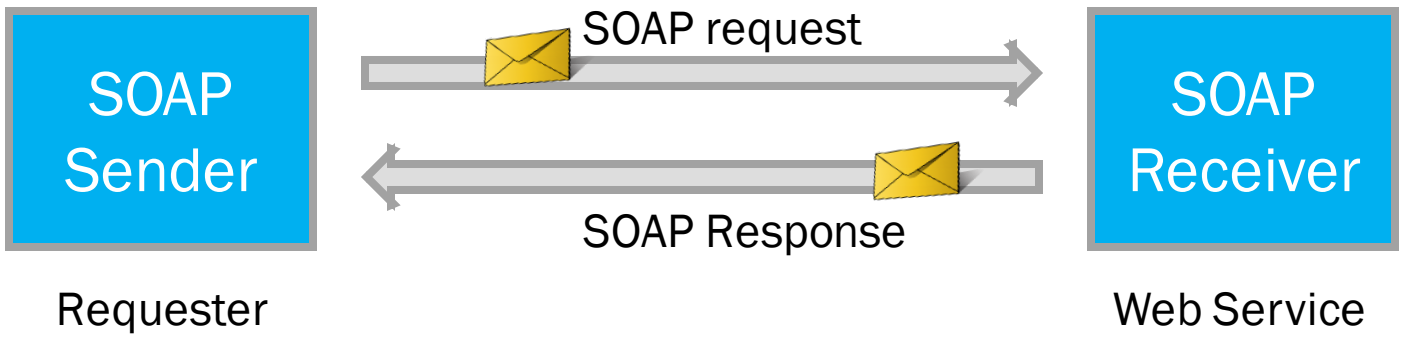
# Web Service – W3C Definition

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

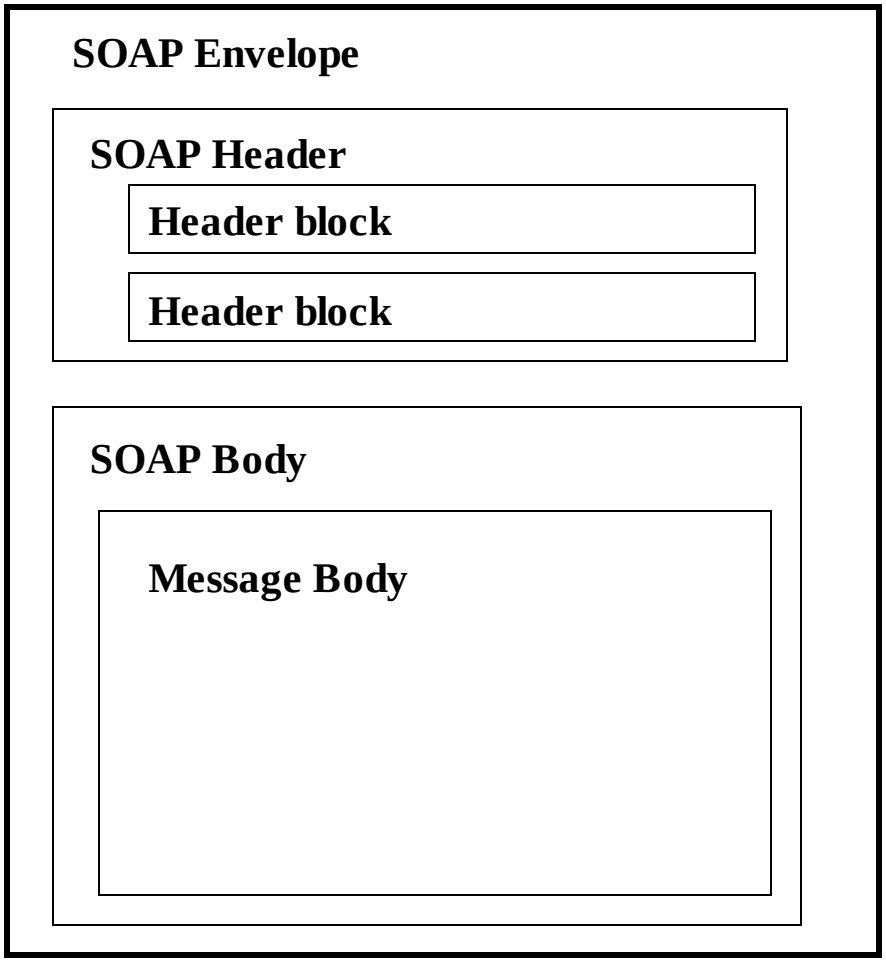
# Simple Object Access Protocol (SOAP)

- SOAP is a lightweight protocol which supports structured message (based on XML) exchange in a decentralised and distributed environment.
- It does not define application semantics, but to provide a mechanism/framework to express application message semantics.

# SOAP


SOAP request
SOAP Sender
SOAP Receiver
SOAP Response
Requester
Web Service

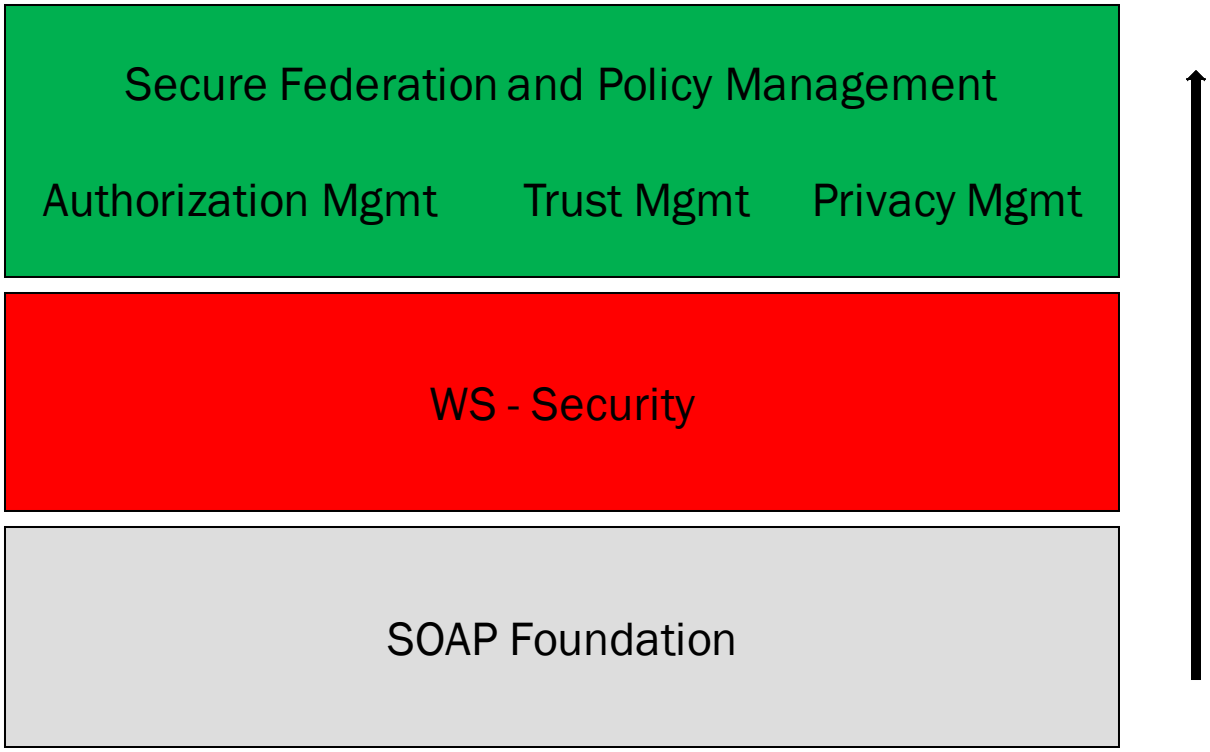
# SOAP Message Structure



# Secure Web Services

- Broad Security Objectives
  - *Authentication of Requests*
  - *Integrity of messages*
  - *Confidentiality of messages*
  - *Authorization of actions requested in messages*
  - *Accountability of actions*

# Securing Web Services



# Web Service Security

- Secure the SOAP to provide integrity and confidentiality.
- WS-Security is flexible and is designed to be used as the basis for the construction of a wide variety of security models including PKI, Kerberos, and SSL.
- The combination of security specifications, related activities, and interoperability profiles will enable customers to easily build interoperable secure Web services.

# Some Terminologies

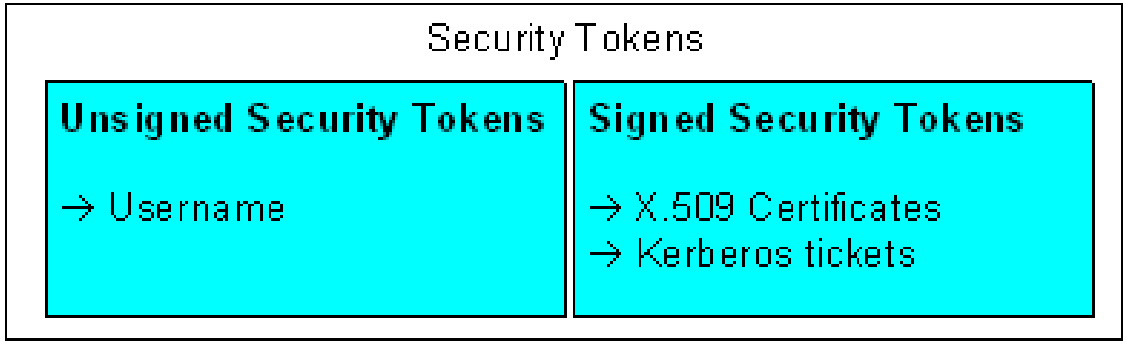
- **Claim:** A *claim* is a statement that a requestor makes (e.g. name, identity, key, group, privilege, capability, etc).
- **Security Token:** A *security token* represents a collection of claims.
- **Signed Security Token:** A *signed security token* is a security token that is asserted and cryptographically endorsed by a specific authority (e.g. an X.509 certificate or a Kerberos ticket).

# WS-Security

- Integrity - information is not modified in transit
  - *XML signature in conjunction with security tokens*
  - *Multiple signature, multiple actors, additional signature formats*
- Confidentiality - only authorized actors or security token owners can view the data
  - *XML encryption in conjunction with security tokens*
  - *Multiple encryption processes, multiple actors*

# WS-Security

- Authentication – you are whom you said you are
  - *Security Tokens*



# Username Token Element

- <UsernameToken Id="...">
- <Username>...</Username>
- <Password Type="...">...</Password>
- </UsernameToken>

Types

| | |
|---|---|
| wsse:PasswordText (default) | The actual password for the username |
| wsse:PasswordDigest | The digest of the password for the username. The value is a base64-encoded SHA1 hash value of the UTF8-encoded Password |

# Username Token Example

```
<wsse:Security>
    <wsse:UsernameToken>
        <wsse:Username>Zoe
        </wsse:Username>
            <wsse:Password>ILoveDogs
        </wsse:Password>
    </wsse:UsernameToken>
</wsse:Security>
```

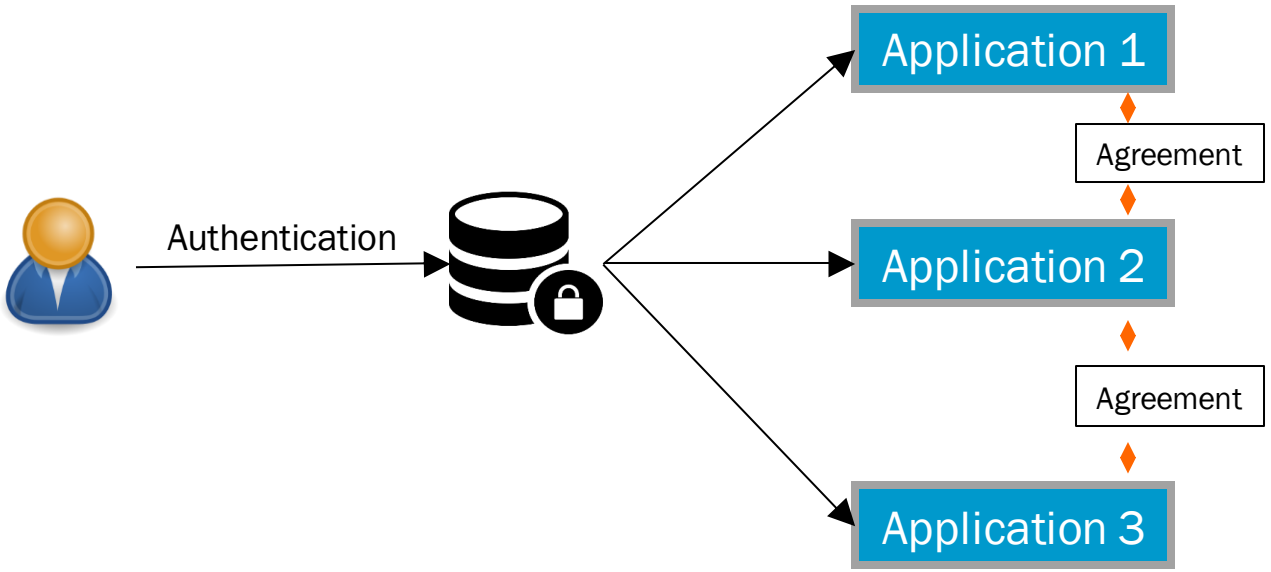
# Security Assertion Markup Language (SAML)

- Open standard (OASIS)
- SAML 1.0 was approved in November, 2002.
- SAML 2.0 was approved in March, 2005.
- XML-based standard for exchanging security information between entities.
  - *Enhance security of SOAP.*

# Why SAML is needed?

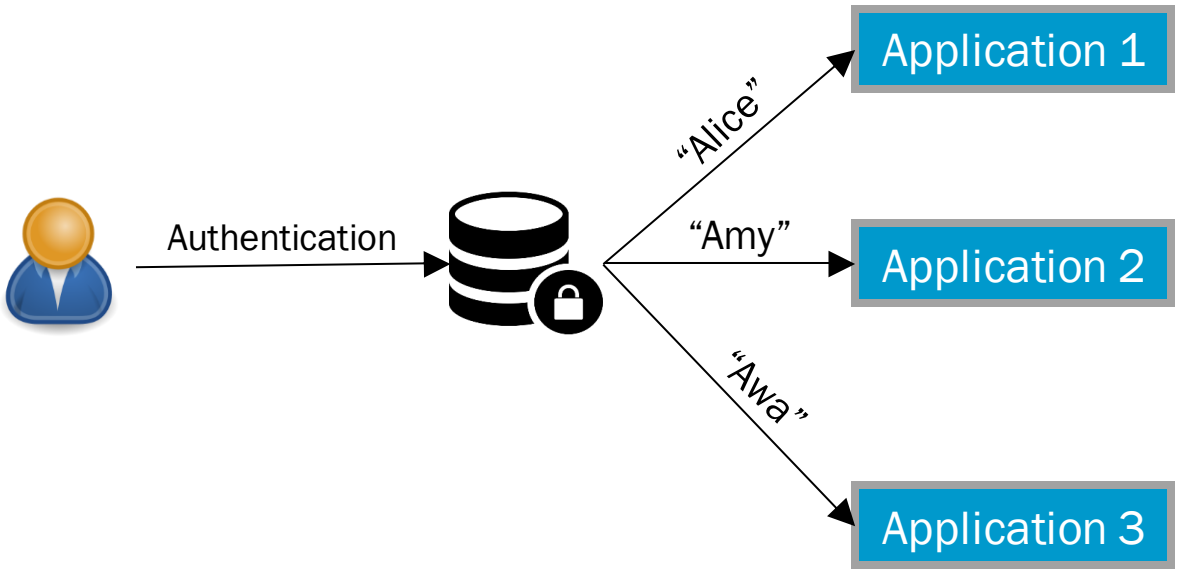
- SAML can provide interoperable solution for exchanging of authentication and authorization messages.
  - *Cross-Domain Single Sign-On (CDSSO)*
    - A standard vendor-independent protocol for transferring information across domains
  - *Federated Identity Management*
    - Shares user's identity information across different organisations.

# Single Sign-On (SSO)




- Single log in authentication for multiple applications and systems.
- No need for authentication at multiple places.

# Federated Identity Management

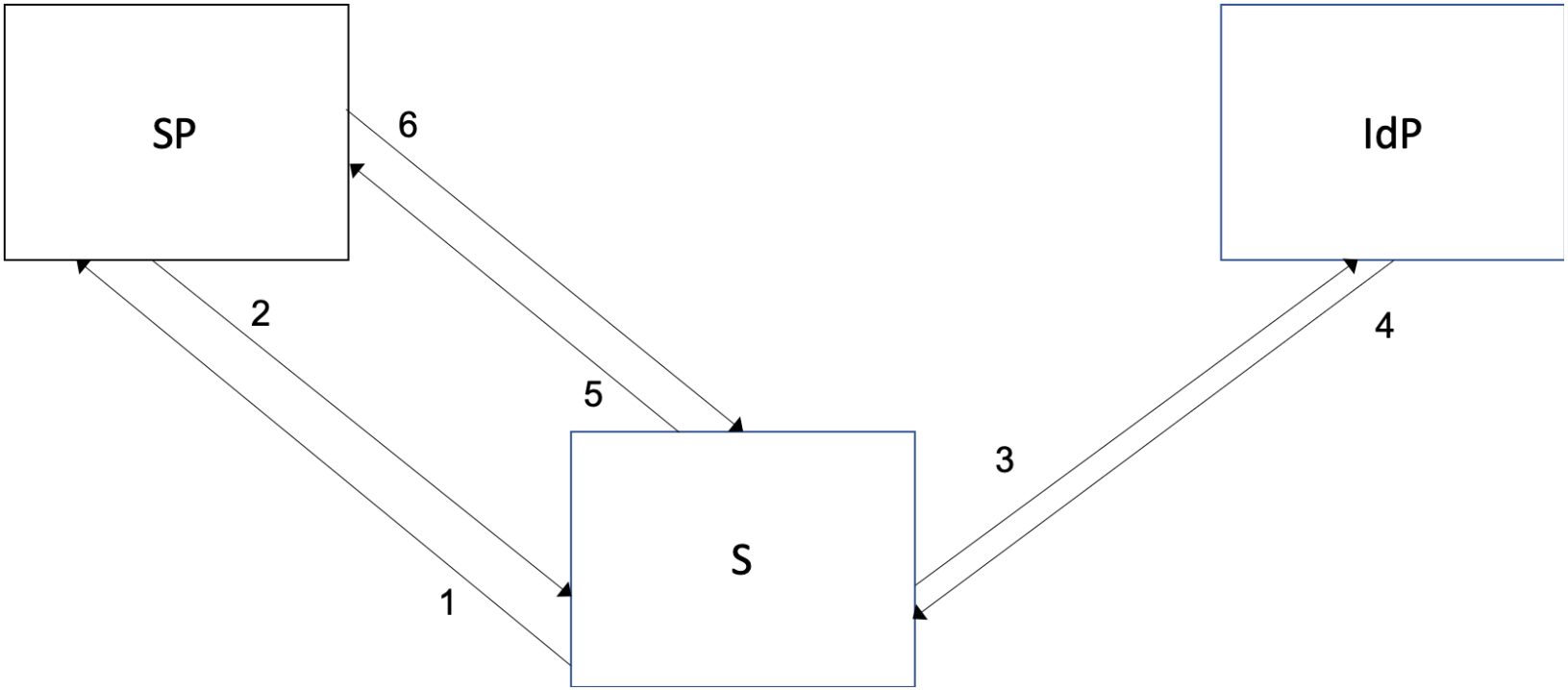



- Applications agrees/trusts on the same identity provider to refer a single user. (A user may be known by different names.)
- FIM gives you SSO, but SSO may not result in FIM.

# SAML Entities

- Service Provider (SP): A SAML-enabled service.
- Subject (S): An entity who request log in to the SP.
- Identity Provider (IdP): A SAML-enabled system that can authenticate S and make assertions about the subject's identity.

# SAML Authentication



# SAML Authentication

1. Subject requests login to the SP.
2. SP sends S's browser an authentication request.
3. Browser relays the authentication request to IdP.
4. IdP attempts to authenticate S, then returns the authentication response to the browser.
5. Browser relays the authentication to SP.
6. SP reads and checks the authentication response, if S is authorized, apply S's privileges as IdP specified.

# SAML Assertions

- An essential unit of SAML.
- Assertion is a claim, statement, or declaration of fact made by some SAML authority.
- To assert characteristics and attributes of a subject.
  - *Bob is a student of UON.*
  - *His email address is bob@Newcastle.edu.au.*

# SAML Assertions

- **Defined types of assertions:**
  - ***Authentication:*** *The assertion subject was authenticated by a particular means at a particular time.*
  - ***Attribute:*** *The assertion subject is associated with the supplied attributes.*
  - ***Authorization Decision:*** *A request to the assertion subject to access the specified resource has been granted or denied.*
- A SAML Token (authentication response from IdP) contains one or more SAML assertions.

# Assertion Statements (V2.0)

- An assertion statement consists of zero or more assertion statements of the following types.
  - *<AuthnStatement>: authentication statement*
  - *<AttributeStatement>: attribute statement*
  - *<AuthzDecisionStatement>: authorization statement*
    - MUST contain <Subject> element in assertion.
  - *<Statement>: custom statement type*

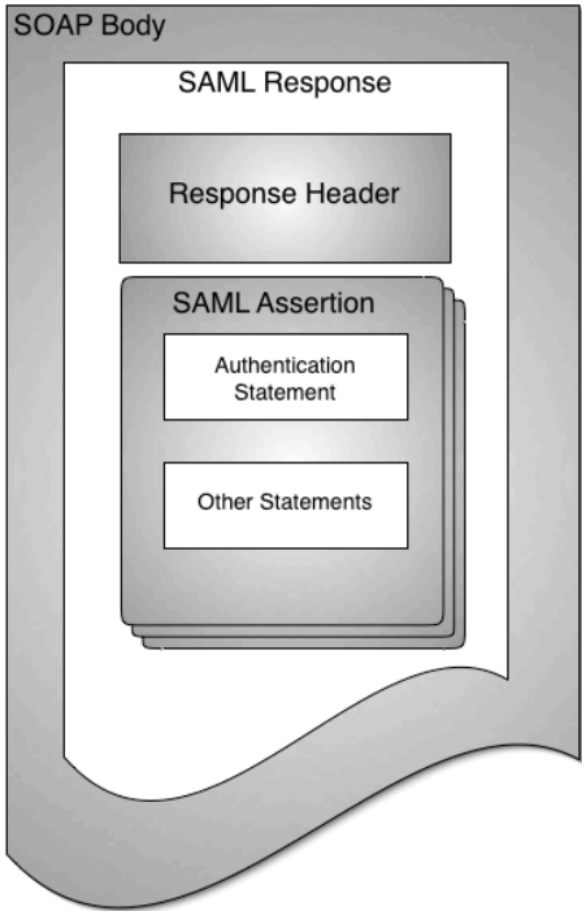
# Example

```
<saml:Assertion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema"
ID="_d71a3a8e9fcc45c9e9d248ef7049393fc8f04e5f75" Version="2.0" IssueInstant="2014-07-17T01:01:48Z">
    <saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer>
    <saml:Subject>
      <saml:NameID SPNameQualifier="http://sp.example.com/demo1/metadata.php" Format="urn:oasis:names:tc:SAML:2.0:nameid-
format:transient">_ce3d2948b4cf20146dee0a0b3dd6f69b6cf86f62d7</saml:NameID>
      <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
        <saml:SubjectConfirmationData NotOnOrAfter="2024-01-18T06:21:48Z" Recipient="http://sp.example.com/demo1/index.php?acs"
InResponseTo="ONELOGIN_4fee3b046395c4e751011e97f8900b5273d56685"/>
      </saml:SubjectConfirmation>
    </saml:Subject>
    <saml:Conditions NotBefore="2014-07-17T01:01:18Z" NotOnOrAfter="2024-01-18T06:21:48Z">
      <saml:AudienceRestriction>
        <saml:Audience>http://sp.example.com/demo1/metadata.php</saml:Audience>
      </saml:AudienceRestriction>
    </saml:Conditions>
    <saml:AuthnStatement AuthnInstant="2014-07-17T01:01:48Z" SessionNotOnOrAfter="2024-07-17T09:01:48Z"
SessionIndex="_be9967abd904ddcae3c0eb4189adbe3f71e327cf93">
      <saml:AuthnContext>
        <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password</saml:AuthnContextClassRef>
      </saml:AuthnContext>
    </saml:AuthnStatement>
    <saml:AttributeStatement>
      <saml:Attribute Name="uid" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue xsi:type="xs:string">test</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute Name="mail" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue xsi:type="xs:string">test@example.com</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute Name="eduPersonAffiliation" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue xsi:type="xs:string">users</saml:AttributeValue>
        <saml:AttributeValue xsi:type="xs:string">examplerole1</saml:AttributeValue>
      </saml:Attribute>
    </saml:AttributeStatement>
  </saml:Assertion>
```

# Example – Assertion Decision

```
<saml:Assertion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" … >
    <saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer>
    <saml:Subject>
      …
    <saml:AuthnStatement …
       …
    </saml:AuthnStatement>
    <saml:AttributeStatement>
       …
    </saml:AttributeStatement>
    <saml:AuthzDecisionStatement
      Resource="http://www.example-bookstore.com/services/getInfo"
      Decision="Permit">
        <saml:Action>Read</saml:Action>
    </saml:AuthzDecisionStatement>
  </saml:Assertion>
```

# SAML Assertion Containment



# SAML and WSS

- WS-Security is a framework for securing SOAP message.
- WSS supports different profiles for various security tokens, such as X.509 certificate and Kerberos ticket.
- SAML assertion is also acceptable by using SAML token profile.

# Security of SAML - Issues

- Mutual authentication
  - *Impersonation.*
- Message integrity
- Confidentiality
- Replay attack
- Man-in-the-middle attack

# Security of SAML - Solutions

- Encrypt security assertions
  - *XML Encryption.*
- Sign security assertions
  - *XML signature.*
- Can also use SSL/TLS connections for secure communication. (This topic will be discussed soon.)

# Encrypted and Signed Assertions

- <EncryptedAssertion>: assertion is encrypted
- <xenc:EncryptedData> [Required]: details are defined by the XML Encryption.
- <xenc:EncryptedKey> [Zero or More]: decryption keys, defined by XML Encryption.
- <ds:Signature> : assertion is signed by using XML signature.

# XML Signature

<Signature ID?>

*<SignedInfo>*

*<CanonicalizationMethod/>*

*<SignatureMethod/>*

*(<Reference URI? > (<Transforms>)?*

*<DigestMethod> <DigestValue> </Reference>)+*

*</SignedInfo>*

*<SignatureValue>*

*(<KeyInfo>)? (<Object ID?>)**

*</Signature>*

# XML Signature Example

```
<Signature Id="MyFirstSignature"
xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo> <CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
<SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
<Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-
20000126/">
<Transforms> <Transform
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
</Transforms>
<DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
<KeyInfo> <KeyValue> <DSAKeyValue>
<P>...</P><Q>...</Q><G>...</G><Y>...</Y>  </DSAKeyValue>
</KeyValue> </KeyInfo> </Signature>
```

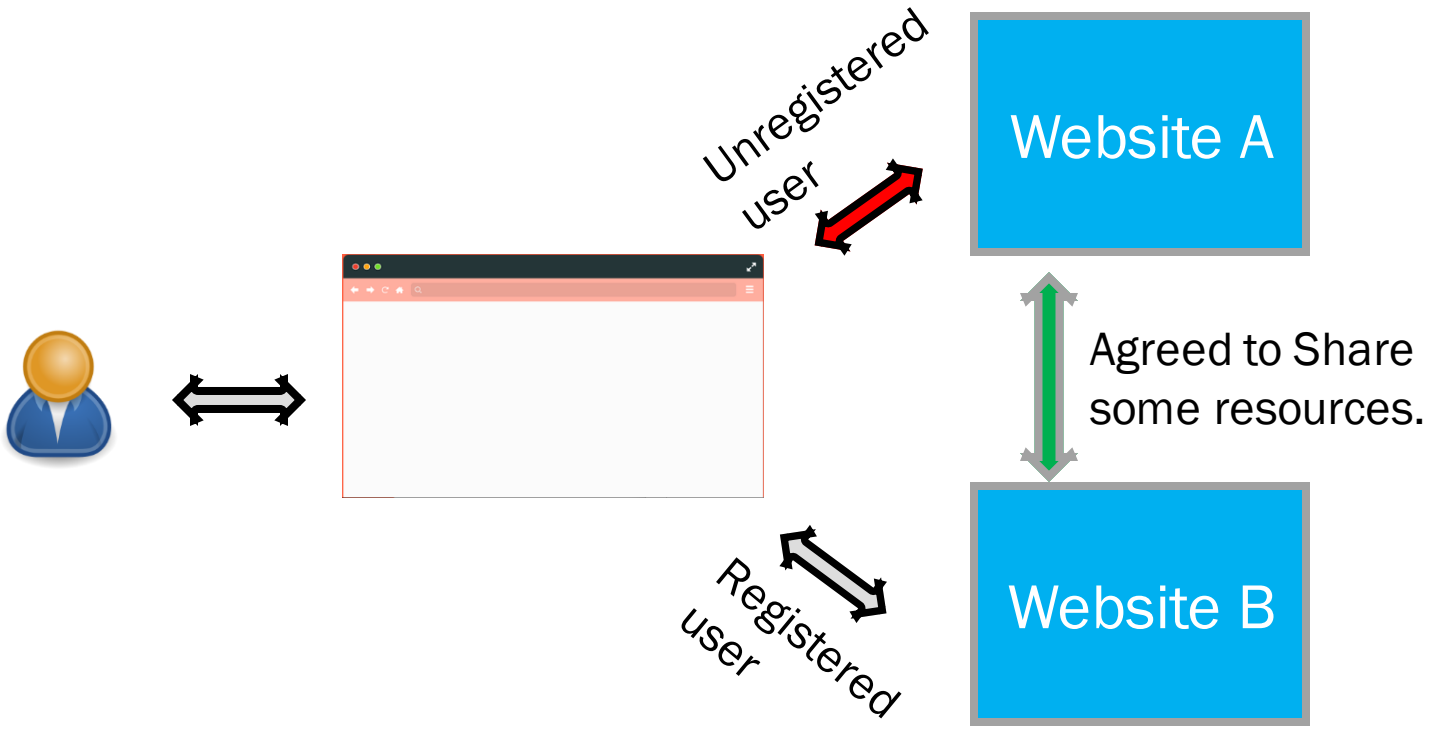
# XML Encryption

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

# OAuth

- OAuth is an authorization standard rather than an authentication standard, and its primary purpose is authorizing third-party applications to access APIs on a user's behalf.
- Enforce least privilege.
- OAuth 2.0 is not compatible with OAuth 1.0.

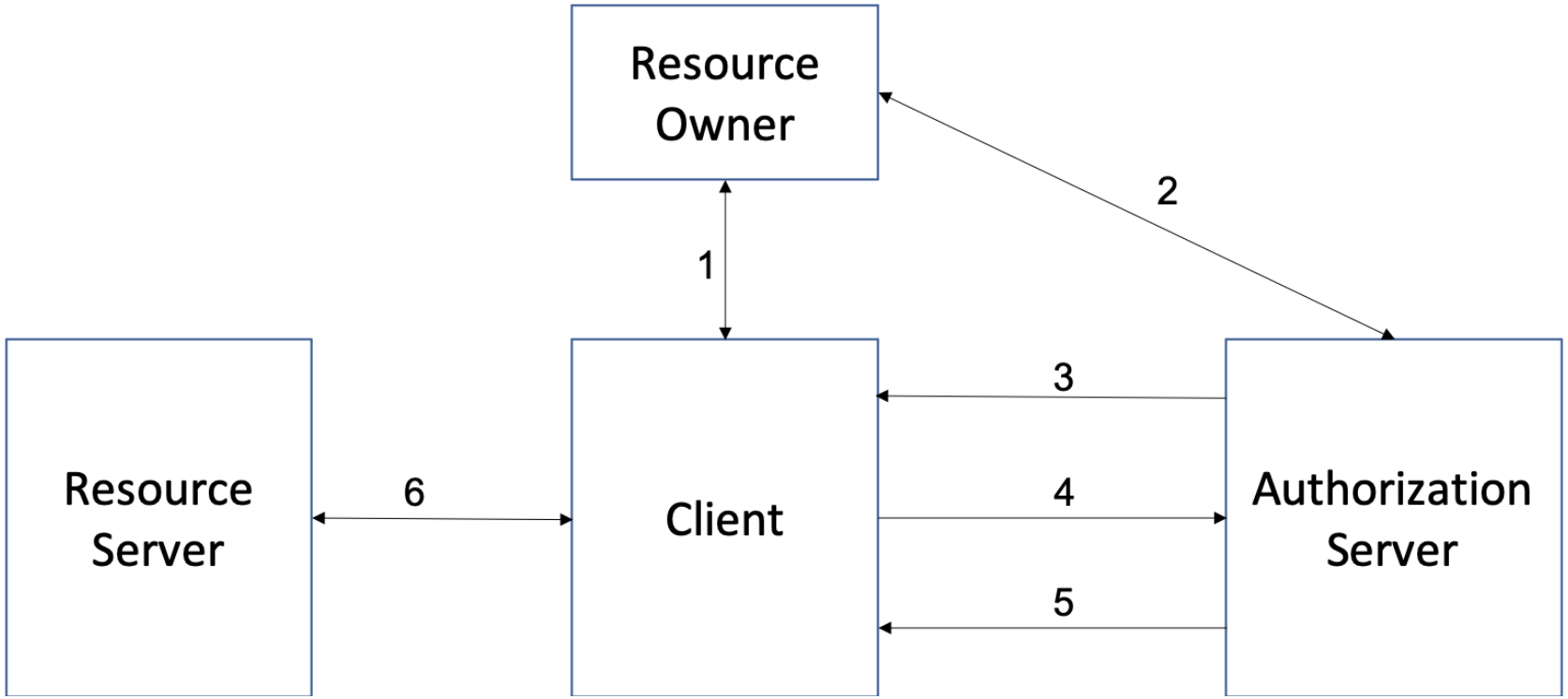
# A Scenario


Unregistered user
Website A
Agreed to Share some resources.
Registered user
Website B

# Roles of OAuth

- **Resource Owner:** the user with a password-protected online account, like SAML subject.
- **Resource Server:** the server on which the APIs reside.
- **Client:** the application that is attempting to access the account APIs, like SAML SP.
- **Authorization Server:** the server that can authenticate the resource owner and grant the client access to the resource server, like SAML IdP.

# OAuth Authorization Overview



# Authorization Flow – Step 0

- Client registers itself to the authorization server, usually by given the application's URL.
- Authorization server assigns client a unique identifier "Client ID" and a Client Secret to use for authentication.

# Authorization Flow – Step 1

- The client initiates the flow by directing the resource owner's user-agent (browser) to the authorization endpoint.
- Client prepares authorization parameters, e.g,
  - *client_id*
  - *redirect_uri*
  - *scope*
  - *access_type*

# Authorization Flow – Step 2

- The authorization server authenticates the resource owner (user) and determines whether the resource owner grants or denies the client's access request.
- This is where user inputs the password and accepts/denies the requested operations.

# Authorization Flow – Step 3

- Authorization Response
- The authorization server redirects the user-agent back to the client using the redirection URI provided earlier.
- Response generated by the authorization server contains the authorization code (an intermediate credential) and the state present in the request URI.

# Authorization Flow – Step 4

- Request access token.
- The client requests an access token from the authorization server's token endpoint by including the authorization code.
- The client includes the redirection URI used to obtain the authorization code for verification.

# Authorization Flow – Step 5

- The authorization server authenticates the client, validates the authorization code, and ensures the redirection URI received matches the URI used to redirect the client in Step 3.
- If verification is valid, the authorization server responds back with an access token and optionally, a refresh token.

# Authorization Flow – Step 6

- Client uses the access token to access user's (resource owner) protected resources.
- Refresh token could continue to function until the user or service provider deauthorizes the client.

# OAuth and SAML

- SAML is designed for web browsers, while OAuth works with native applications.
- SAML may particularly require encryption and digital signature for assertion content.
- OAuth uses TLS/SSL connections to provide confidentiality and integrity.

# OAuth Protocol Example

- Alice has a Gmail account with hundreds of contacts in her contact list
- She joins Facebook and would like to see which of her Gmail contacts she can befriend in the social network
- She can search Facebook for each one individually, but allowing Facebook to read her Gmail contacts directly would be much easier
- OAuth Protocol helps to make this easier

# OAuth Protocol Example

- Resource Owner : Alice
- OAuth Client: Facebook
- Authorization Server: Gmail
- Resource Server: Gmail

# OAuth Protocol Example

- Alice (Resource Owner) logs into her Facebook account and selects an option on the Facebook website to import contacts from Gmail
- Facebook (the OAuth Client) sends a response to Alice's web request. The response goes to her web browser, redirects the browser to Gmail (the Authorization Server) and passes the request to it
- Gmail prompts Alice to accept the requested authorization. She sees this authorization prompt in the browser. (It is the response to her original request). Prompt might say: Facebook is requesting access to read your contact list in Gmail. The Prompt will come from Gmail and will ask Alice to log in to Gmail to approve the request. The Client (Facebook) is not involved in this step

# OAuth Protocol Example

- Alice accepts the authorization request. She might have to login or might already be logged into Gmail in the browser. Typically she would say “OK” or “Accept”
- Gmail sends a response to Alice’s web browser that contains the authorization code and the response redirects the web browser back to Facebook (Alice does not see this)
- Facebook sends the authorization code directly to the authorization server (behind the scenes). Gmail responds to Facebook with an access token (if all ok)
- Facebook uses the access token to contact the resource server (Gmail) and perform the service for Alice – that is, retrieve her Gmail contact list and import the contacts into her Facebook account.

# OAuth Protocol Example

- From Alice's viewpoint, the interaction has been simple
  - *She made a request from Facebook to access Gmail,*
  - *She sees a prompt from Gmail asking her to authorize it*
  - *Then she sees Facebook acknowledge that authorization was received, and*
  - *That the request has been handled and completed*

Exercise: Draw the protocol flow

# References


- https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice
- http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf