Web Service Security

Dominic Duggan

Stevens Institute of Technology

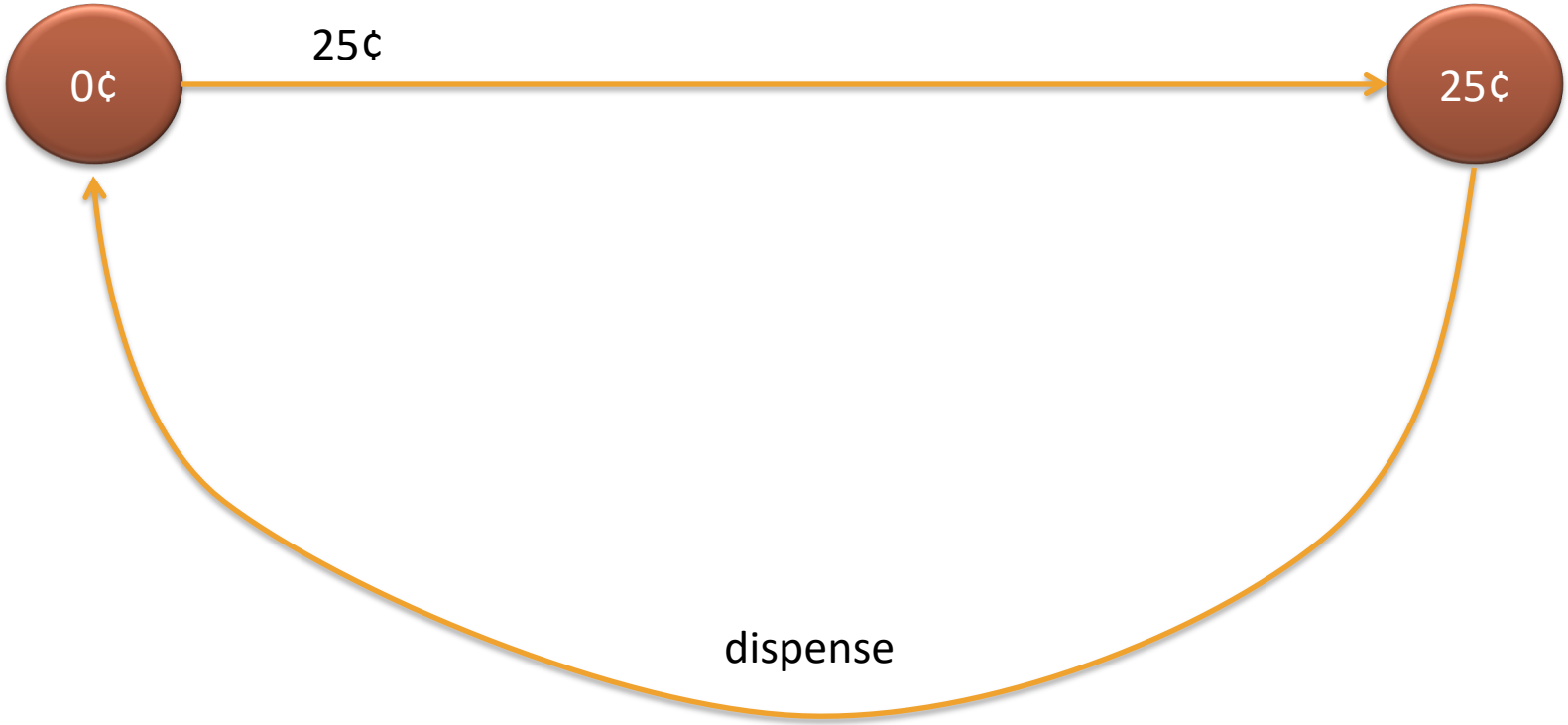
Based on materials by Gustavo Alonso, Eve Maier, Travis Spencer

REPRESENTATIONAL STATE TRANSFER (REST)

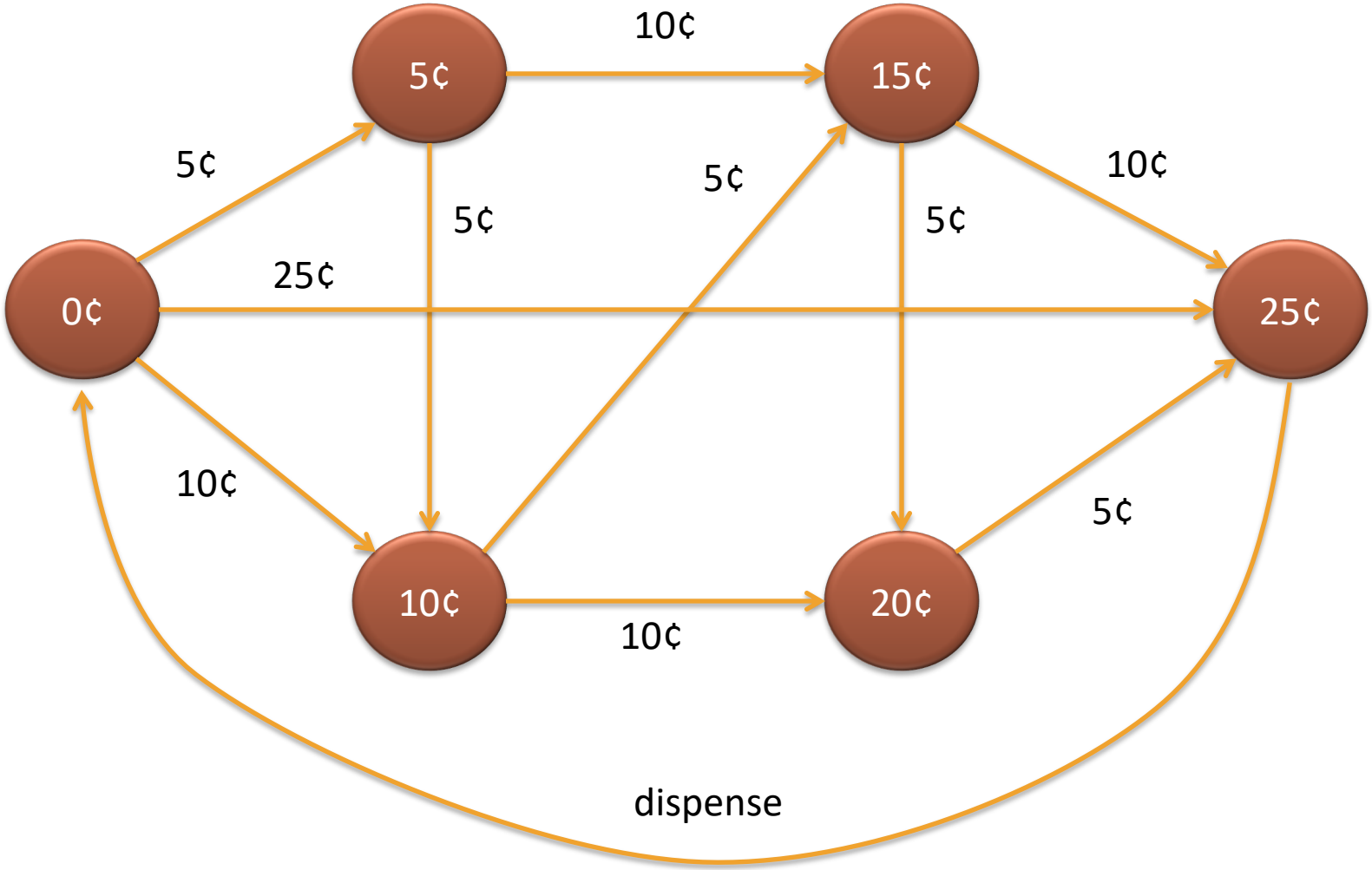
Representational State Transfer

- Software architecture for the Web
- Web browsing as navigation of hypermedia network

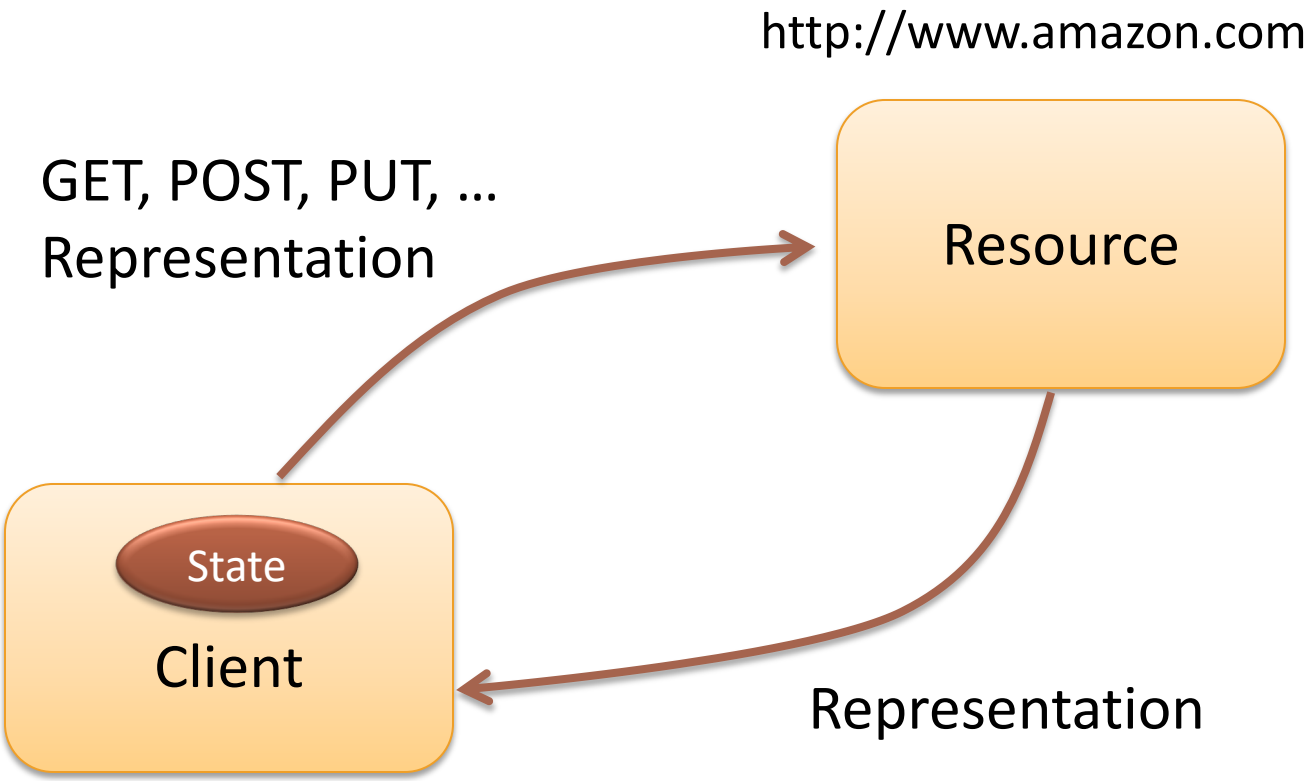
State Machine



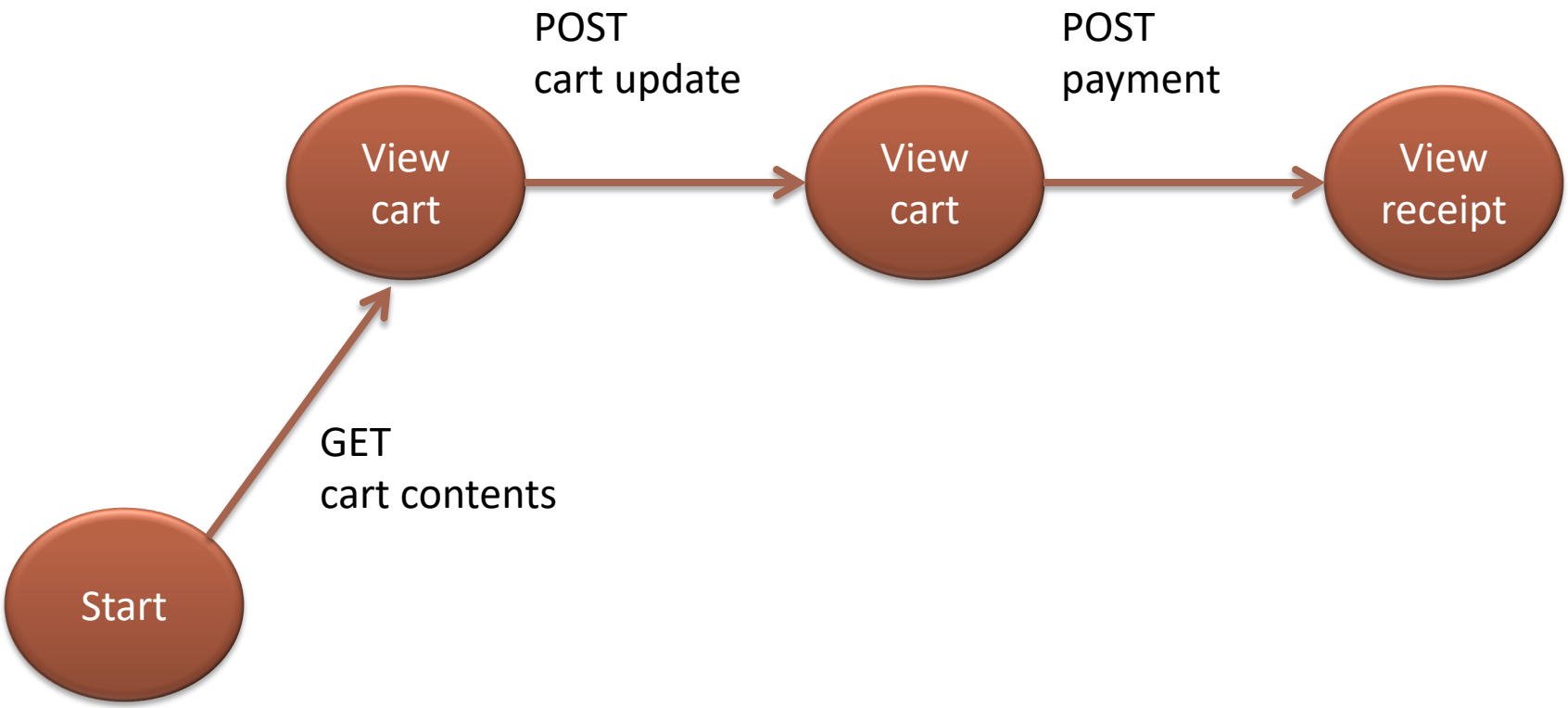
State Machine



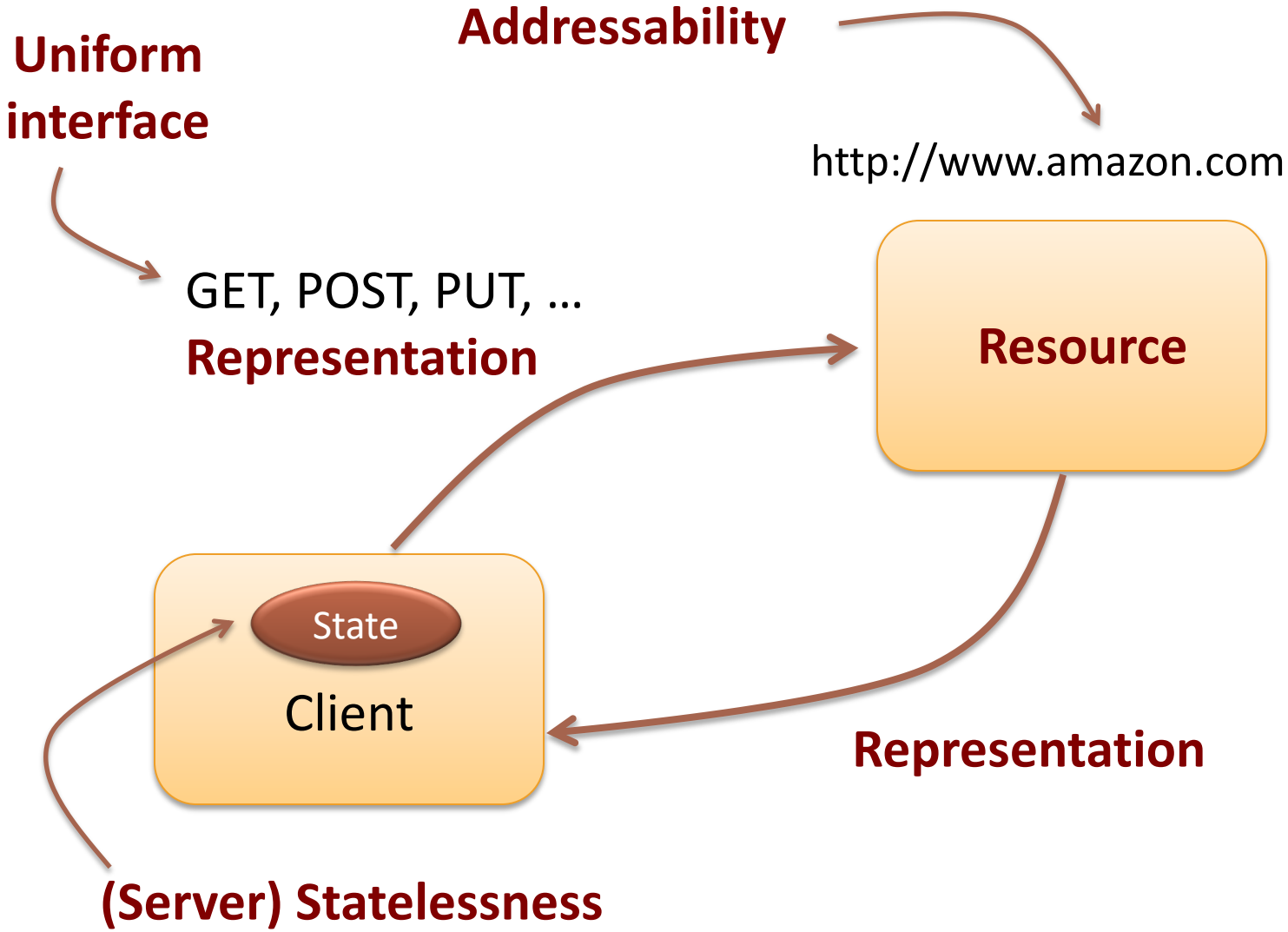
Representational State Transfer



Client as State Machine



Representational State Transfer



Representational State Transfer

- Originally a software architecture for the Web
- Emerged as an alternative architecture for Web services
  - Resource-oriented architecture

SOAP / WSDL	**REST**
Service (operation) oriented	Resource oriented
One endpoint URL	URL for each individual resource
Application-defined verbs	Fixed set of HTTP verbs

REST Verbs

- Retrieve: HTTP GET
- Create:
  - HTTP PUT for new URI or
  - HTTP POST for existing URI (server decides result URI)
- Modify: HTTP PUT to existing URI
- Delete: HTTP DELETE

- Merge updates: HTTP PATCH
- Retrieve metadata only: HTTP HEAD
- Check which methods are supported: HTTP OPTIONS

- No other operations besides these

Example: Amazon Simple Storage Service (S3)

- S3 is based on two concepts
  - Buckets
    - Named container
  - Objects
    - Named piece of data, with metadata
    - Stored in buckets

S3 RPC Interface

- Object-oriented interface to S3
  - `CreateBucket`
  - `ListAllMyBuckets`
- Getter/setter methods on bucket and object "objects"
  - `S3Object.name()`
  - `S3Object.setValue()`
  - `S3Bucket.getObjects()`

S3 REST Interface

- Three types of resources
  - List of your buckets

    `https://s3.amazonaws.com`
  - A particular bucket (virtual host)

    `https://`*`name-of-bucket`*`.s3.amazonaws.com`
  - A particular s3 object inside a bucket

    `https://`*`name-of-bucket`*`.s3.amazonaws.com/`*`name-of-object`*

S3 REST Interface

- Example:
  - A particular bucket

    `https://jeddak.s3.amazonaws.com`
  - A particular s3 object inside a bucket
    - Object names:
      docs/manual.pdf, docs/security.pdf, talks/snt.pdf
    - Resource URIs:
      `https://jeddak.s3.amazonaws.com/docs/manual.pdf`
      `https://jeddak.s3.amazonaws.com/docs/security.pdf`
      `https://jeddak.s3.amazonaws.com/talks/snt.pdf`

S3 REST Interface

- Use HTTP methods as verbs

Verb	Bucket list	Bucket	Object
GET	List buckets	List bucket objects	Get value and metadata
HEAD			Get metadata
PUT		Create bucket	Set object value and metadata
DELETE		Delete bucket	Delete object

HTTP

HTTP Request

```
GET /index.html HTTP/1.1
Host: www.example.org

...request headers...
```

HTTP Response

```
HTTP/1.1 200 OK
Date: Mon, 1 May 2011 21:38:14 GMT
Server: Apache/1.3.34 (Debian) mod ssl/2.8.25
OpenSSL/0.9.8c ...
Last-Modified: Wed, 25 Nov 2009 12:27:01 GMT
ETag: "7496a6-a0c-4b0d2295"
Accept-Ranges: bytes
Content-Length: 2572
Content-Type: text/html
Via: 1.1 www.example.org
Vary: Accept-Encoding
...
```

Request Headers

- Accept: for content negotiation
  - Content-Type: response header e.g.
    ATOM (application/atom+xml)
    RDF (application/rdf+xml)
    XHTML (application/xhtml+xml)
    Form-encoded key-value pairs (application/x-www-form-urlencoded)
- Authorization: app-defined auth info
  - WWW-Authenticate: response header with status code of 401 (“Unauthorized”)

Request Headers

- Cookie: (non-standard)
  - Save-Cookie: to save cookie on client

Response Headers

- Last-modified: time of last modification
  - If-Last-Modified: request header for caching
- Etag: hash of metadata
  - If-None-Match: request header for caching
- Cache-Control: how long to cache
- Upgrade: upgrade protocol e.g. http to https
- Location: URI for newly created resource, redirection, …

Response Codes

- 1XX: for negotiation with Web server
  - E.g. 101 (“Switching protocols”) with Upgrade: response header
- 2XX: to signal success
  - E.g. 200 (“Success”), 201 (“Created”), …
- 3XX: redirect clients
  - E.g. 303 (“See other”), 307 (“Temporary redirect”)
- 4XX: client errors
  - E.g. 400 (“Bad request”), 404 (“Not found”), 401 (“Unauthorized”), 403 (“Forbidden”)
- 5XX: server errors
  - E.g. 500 (“Internal server error”)

JAX-RS API

JAX-RS

- RESTful Web services implemented as methods of objects
- @Path for class: base context root
- @Path for methods: extensions for resources
- @Get, @Post, @Put, etc for methods
- @Produces, @Consumes: MIME types
- @QueryParam: param from query string

Example

```
@Path("/HelloService")
public class HelloResource {
  @GET
  @Produces("text/plain")
  public String sayHello (
          @QueryParam("name") String name
  ) {
     return "Hello, " + name;
  }
}
```

GET http://host/HelloService?name=Joe

Passing Parameters to Service

- `@QueryParam`
- `@PathParam`
- `@MatrixParam`
- `@HeaderParam`
- `@CookieParam`
- `@FormParam`

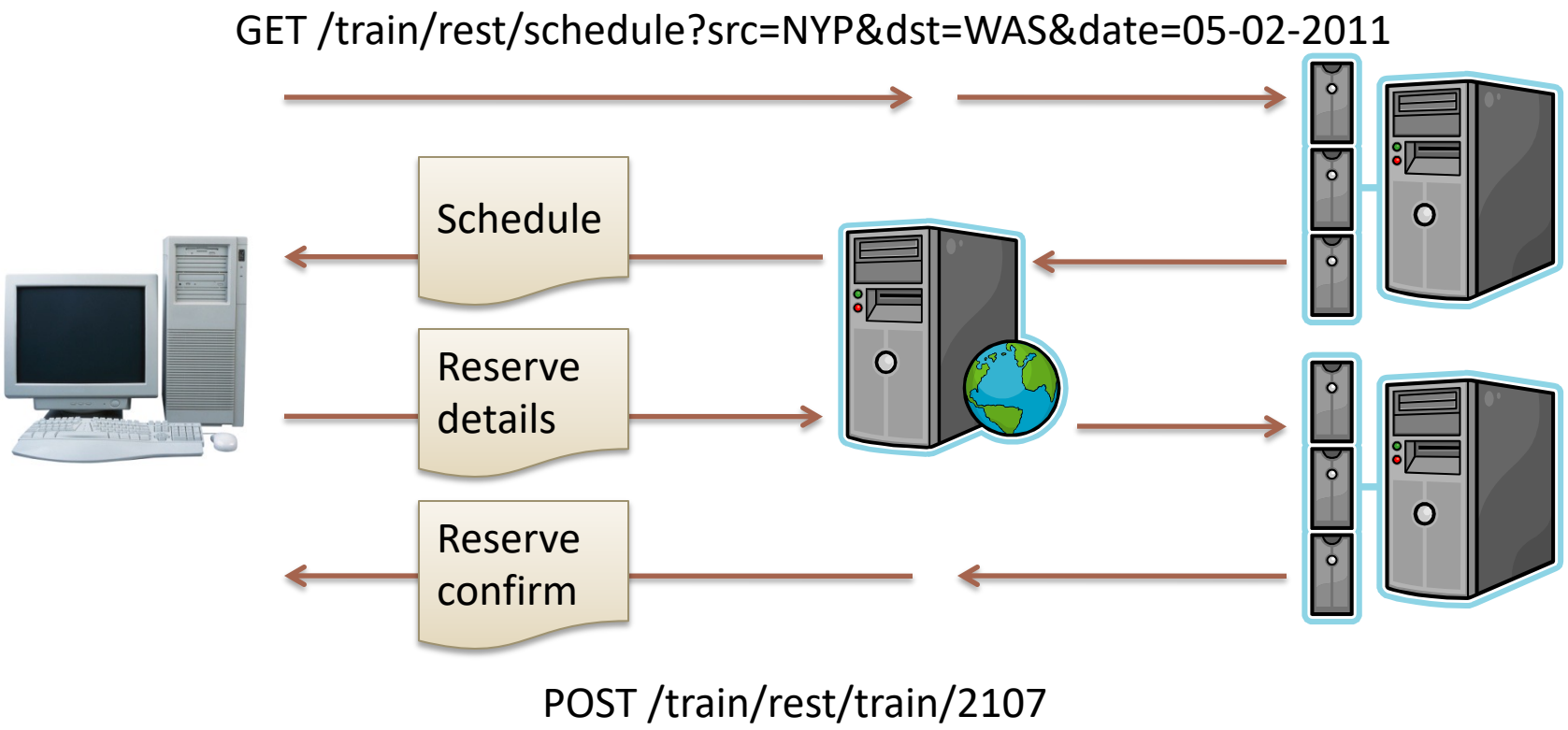
- `@BeanParam`: inject a bean with all parameters

Example with Bean parameter

```
public class Book {
   @FormParam(“isbn”)
   private String isbn;
   @FormParam(“title”)
   private String title;
   @FormParam(“author”)
   private String author;
   ...
}

@POST
@Consumes(MediaType.APPLICATION_FORM_URNENCODED)
public Response addBookToCart(@BeanParam Book book) {
   // Add book parameter to shopping cart
   return Response.ok().build();
}
```

EXAMPLE: TRAIN RESERVATION SERVICE

REST Web Service



Application Class

```
@ApplicationPath("/train/rest")
public class TrainApp extends Application {
  public Set<Class<?>> getClasses() {
      Set<Class<?>> s = new HashSet<Class<?>>();
      s.add(ScheduleResource.class);
      s.add(TrainResource.class);
      return s;
  }
}
```

GET /train/rest/schedule?src=NYP&dst=WAS&date=05-02-2011

Schedule Resource

```
@Path("/schedule")
public class ScheduleResource {

   ...

  @GET
  @Produces("application/vnd.trains+xml")
  public Response getSchedule (
                    @QueryParam("src") String start,
                    @QueryParam("dst") String destination,
                    @DefaultValue("today")
                    @QueryParam("date") String travelDate)
```

GET
/train/rest/schedule?src=src&dest=dest&date=travDate

Schedule Resource

```
@Path("/schedule")
public class ScheduleResource {

   ...

   @GET
   @Produces("application/vnd.trains+xml")
   public Response getSchedule (
                     @QueryParam("src") String start,
                     @QueryParam("dst") String destination,
                     @DefaultValue("today")
                     @QueryParam("date") String travelDate)
```

GET
/train/rest/schedule?src=src&dest=dest&date=travDate

```
@Path("/schedule")
public class ScheduleResource {

  @Context UriInfo uriInfo;

  @Inject ISchedule scheduleService;

  @GET
  @Produces("application/vnd.trains+xml")
  public Response getSchedule (
                        @QueryParam("src") String start,
                        @QueryParam("dst") String destination,
                        @DefaultValue("today")
                        @QueryParam("date") String travelDate)
    {
      ScheduleRepresentation schedule =
              scheduleService.get(start, destination,
                                  travelDate, uriInfo);
      return Response.ok().entity(schedule).build();
    }
  }
}
```

REST Response

```
<tr:Schedule
    xmlns:tr="http://www.example.org/
    schemas/train">
<tr:train>
  <tr:uri>
    http://www.example.org
    /train/rest/train/2103
  </tr:uri>
  <tr:time>0600</tr:time>
</tr:train>
<tr:train>
  <tr:uri>
    http://www.example.org
    /train/rest/train/2107
  </tr:uri>
  <tr:time>0700</tr:time>
</tr:train>
<tr:train>
  <tr:uri>
    http://www.example.org
    /train/rest/train/183
  </tr:uri>
  <tr:time>0717</tr:time>
</tr:train>
<tr:train>
  <tr:uri>
    http://www.example.org
    /train/rest/train/2109
  </tr:uri>
  <tr:time>0800</tr:time>
</tr:train>
...
</tr:Schedule>
```

REST CLIENT API

Jersey Client API

- Create a shopping cart:

```
Client client = ClientBuilder.newClient();

UriBuilder uriBase =
  UriBuilder.fromURI
  ("http://www.jeddak.org/rest/shoppingcart/joe");
URI resourceUri = uriBase.build();
WebTarget cart = client.target(resourceUri);

cart.request().put(Entity.text(“”));
```

Jersey Client API

- Add a film to the shopping cart:

```
Form f1 = new Form();
f1.add("product","Lawrence of Arabia");
f1.add("amount","24.95");

URI addUri = uriBase.path("add").build();
WebTarget cartAdd = c.target(addUri);

Entity<Form> f1Entity = Entity.entity(f1,
   MediaType.APPLICATION_FORM_URLENCODED_TYPE);
cartAdd.request().post(f1Entity);
```

Jersey Client API

- Add a book and review the shopping cart:

```
Form f2 = new Form();
f2.add("product","A Fistful of Dollars");
f2.add("amount","19.95");

Entity<Form> f2Entity = Entity.entity(f2,
   MediaType.APPLICATION_FORM_URLENCODED_TYPE);
cartAdd.request().post(f2Entity);

cart.request(MediaType.APPLICATION_JSON_TYPE);
    .get(CartEntity.class);
```

Catching Errors

```
Response response = cartAdd.request().post();

if (response.getStatusInfo() ==
    Response.Status.NO_CONTENT) {

   response = cart.get();
   if (response.getStatusInfo() == Response.Status.OK)
   {

       CartEntity cartEntity =
             response.getEntity(CartEntity.class);
       ...
   }
}
```

JAVA EE SECURITY

Java SE Security Mechanisms

- Java Authentication and Authorization Service (JAAS)
  - PAM (pluggable login framework)
  - Principal-based access control
- Java Generic Security Services (Java GSS-API)
  - Kerberos
- Java Cryptography Extension (JCE)
- Java Secure Sockets Extension (JSSE)
- Simple Authentication and Security Layer (SASL)
  - Use for LDAP-based authentication

Java EE Security Mechanisms

- **Key Issue: how to integrate JAAS**
- Java Authentication Service Provider Interface for Containers (JASPIC)
  - Authentication
  - Standardizing what servlet implementations do
  - Influenced by SOAP security
- Java Authorization Contract for Containers (JACC)
  - Authorization
- Java EE Security API
  - `HttpAuthenticationMechanism`
  - Identity Store abstraction

JASPIC Server Auth Module

```
public interface ServerAuthModule extends ServerAuth {

  /**
   * Initialize this module with request and response message policies
   * to enforce, a CallbackHandler, and any module-specific configuration
   * properties.
   */
  void initialize(MessagePolicy requestPolicy, MessagePolicy responsePolicy, CallbackHandler handler, Map options) throws AuthException;

  /**
   * Get the one or more Class objects representing the message types
   * supported by the module.
   */
  Class[] getSupportedMessageTypes();

  /**
   * Authenticate a received service request.
   */
  AuthStatus validateRequest(MessageInfo messageInfo, Subject clientSubject, Subject serviceSubject) throws AuthException;

  /**
   * Secure a service response before sending it to the client.
   */
  AuthStatus secureResponse(MessageInfo messageInfo, Subject serviceSubject) throws AuthException;

  /**
   * Remove method specific principals and credentials from the subject.
   */
  void cleanSubject(MessageInfo messageInfo, Subject subject) throws AuthException;
}
```

HTTP Auth Module

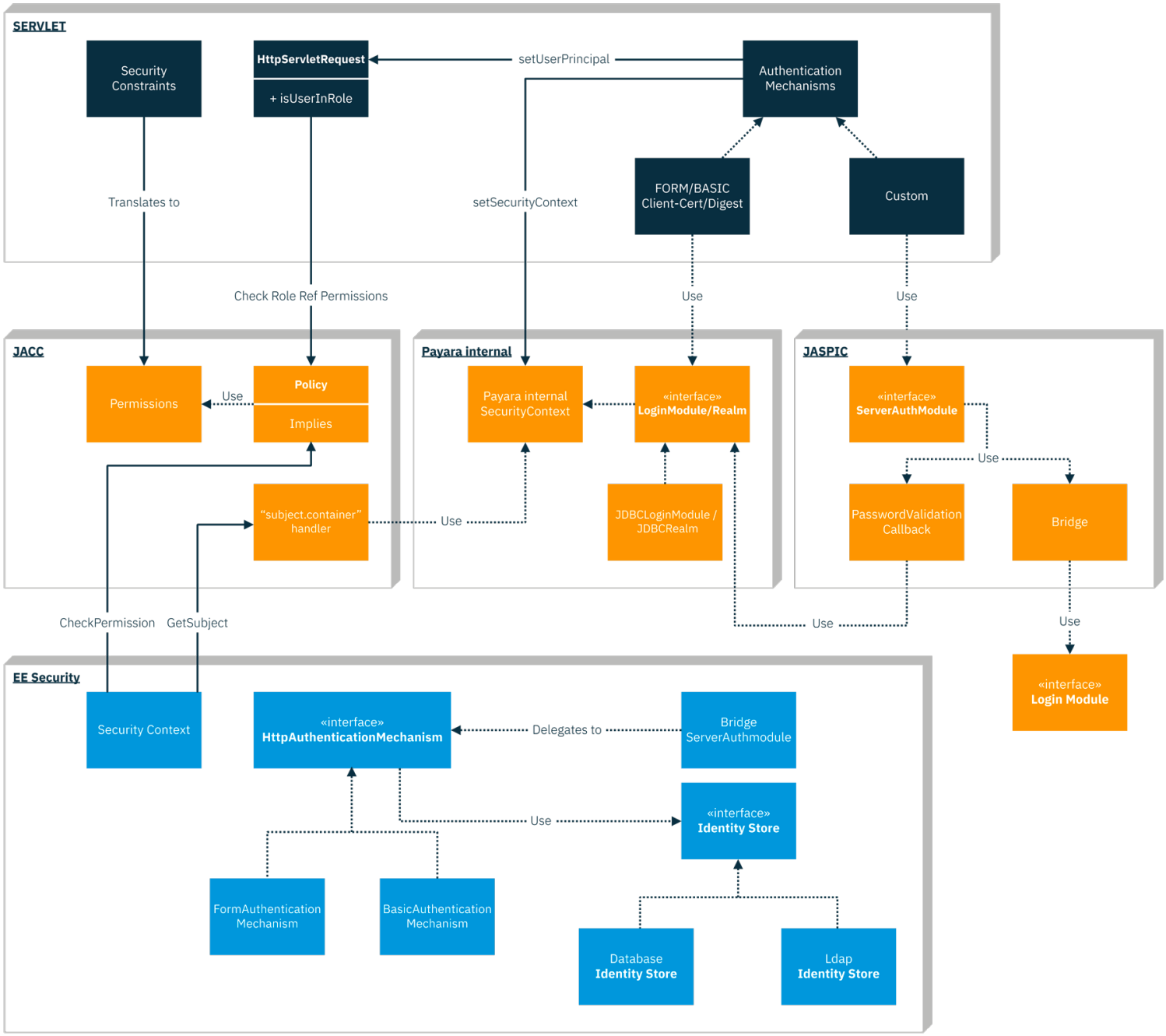
```
public interface HttpAuthenticationMechanism {

 /**
    * Authenticate an HTTP request.
    *
    * <p>
    * This method is called in response to an HTTP client request for a resource, and is always invoked
    * <strong>before</strong> any {@link Filter} or {@link HttpServlet). Additionally this method is called
    * in response to {@link HttpServletRequest#authenticate(HttpServletResponse)}
    */
 AuthenticationStatus validateRequest(HttpServletRequest request, HttpServletResponse response, HttpMessageContext httpMessageContext) throws Auth

 /**
    * Secure the response, optionally.
    */
 default AuthenticationStatus secureResponse(HttpServletRequest request, HttpServletResponse response, HttpMessageContext httpMessageContext) thro
 return SUCCESS;
 }

 /**
    * Remove mechanism specific principals and credentials from the subject and any other state the mechanism
    * might have used.
    */
 default void cleanSubject(HttpServletRequest request, HttpServletResponse response, HttpMessageContext httpMessageContext) {
 httpMessageContext.cleanClientSubject();
 }
}
```

JAAS Login Module

```
public interface LoginModule {

    /**
     * Initialize this LoginModule.
     */
    void initialize(Subject subject, CallbackHandler callbackHandler,
                    Map<String,?> sharedState,
                    Map<String,?> options);

    /**
     * Method to authenticate a { @code Subject} (phase 1).
    */
    boolean login() throws LoginException;

    /**
     * Method to commit the authentication process (phase 2).
    */
    boolean commit() throws LoginException;

    /**
     * Method to abort the authentication process (phase 2).
     */
    boolean abort() throws LoginException;

    /**
     * Method which logs out a { @code Subject}.
    */
    boolean logout() throws LoginException;
}
```

Identity Store

```
public interface IdentityStore {

    /**
    * Validates the given credential.
    */
    CredentialValidationResult validate(Credential credential);

   /**
    * Returns groups for the caller , who is identified by the { @link CallerPrincipal}
    * (and potentially other values) found in the { @code validationResult} parameter.
    */
    Set<String> getCallerGroups(CredentialValidationResult validationResult);

    /**
    * Determines the order of invocation for multiple { @link IdentityStore}s.
    */
    int priority();

    /**
    * Determines the type of validation the { @link IdentityStore} should be used for.
    */
    Set<ValidationType> validationTypes();
}
```


SERVLET
Security Constraints
HttpServletRequest
+ isUserInRole
setUserPrincipal
Authentication Mechanisms
Translates to
setSecurityContext
FORM/BASIC Client-Cert/Digest
Custom
Check Role Ref Permissions
Use
Use
JACC
Permissions
Use
Policy
Implies
Payara internal
Payara internal SecurityContext
«interface» LoginModule/Realm
JASPIC
«interface» ServerAuthModule
Use
"subject.container" handler
Use
JDBCLoginModule / JDBCRealm
PasswordValidation Callback
Bridge
CheckPermission
GetSubject
Use
Use
«interface» Login Module
EE Security
Security Context
«interface» HttpAuthenticationMechanism
Delegates to
Bridge ServerAuthmodule
Use
«interface» Identity Store
FormAuthentication Mechanism
BasicAuthentication Mechanism
Database Identity Store
Ldap Identity Store

Forms of Authentication

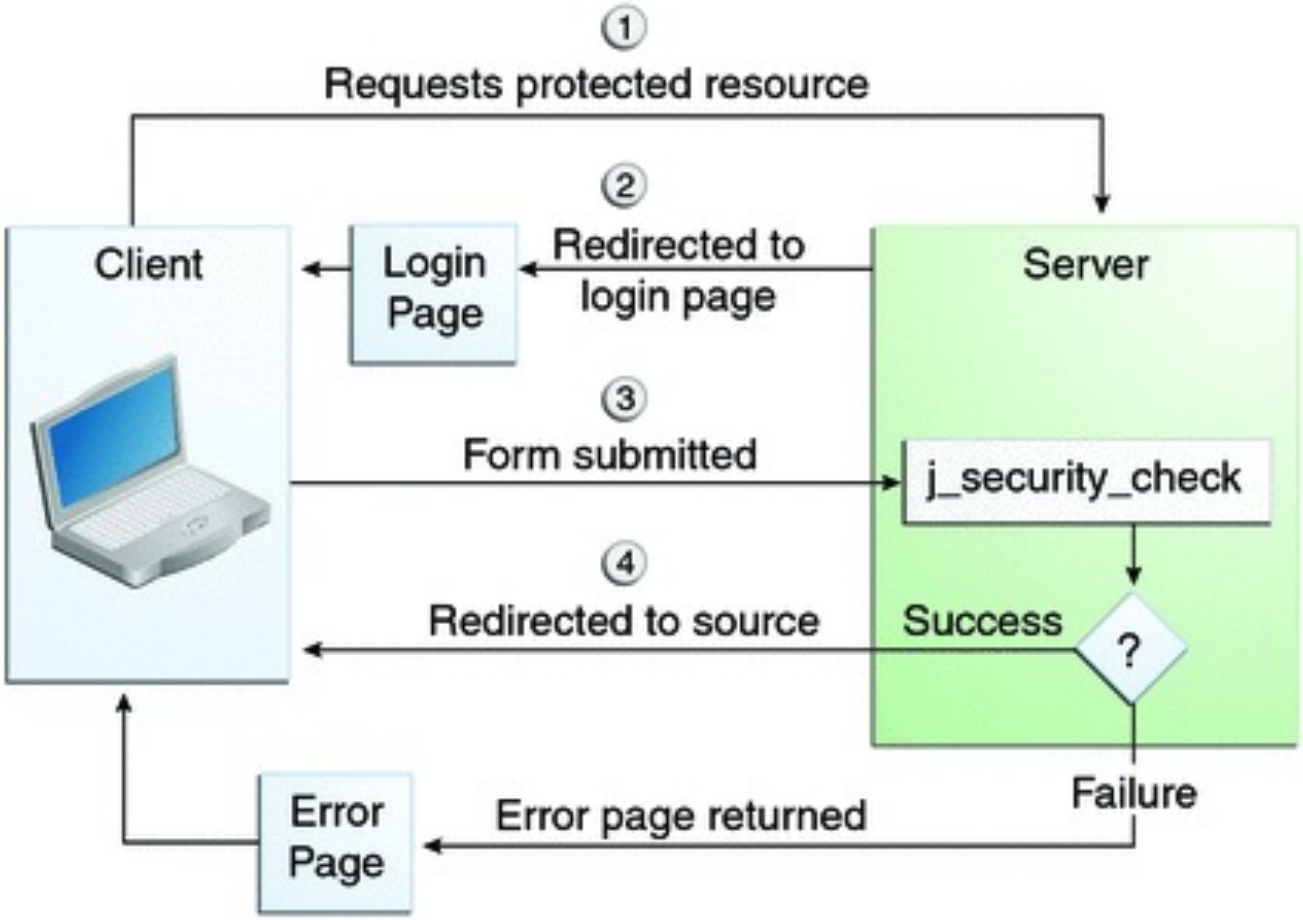
- Basic
  - Simple userid/password
- Form
  - Login session
- Digest
  - Hashed password
- Client Certificate
  - X509
- Mutual

Security Realm

- **Realm:** "A security policy domain defined for a web or application server. A realm contains a collection of users, who may or may not be assigned to a group."
- Example realms:
  - file
  - certificate
  - ldap
  - jdbc

```
<login-config>
   <auth-method>FORM</auth-method>
   <realm-name>ChatRealm</realm-name>
   ...
</login-config>
```

Form-Based Authentication



Forms-Based Authentication

- Example Configuration

```
@FormAuthenticationMechanismDefinition(
  loginToContinue = @LoginToContinue(
    loginPage = "/login.html",
    errorPage = "/login-error.html"))
@ApplicationScoped
public class AppConfig{ }
```

- Option of customizing login and login error pages

Custom Forms-Based Authentication

- Example Configuration

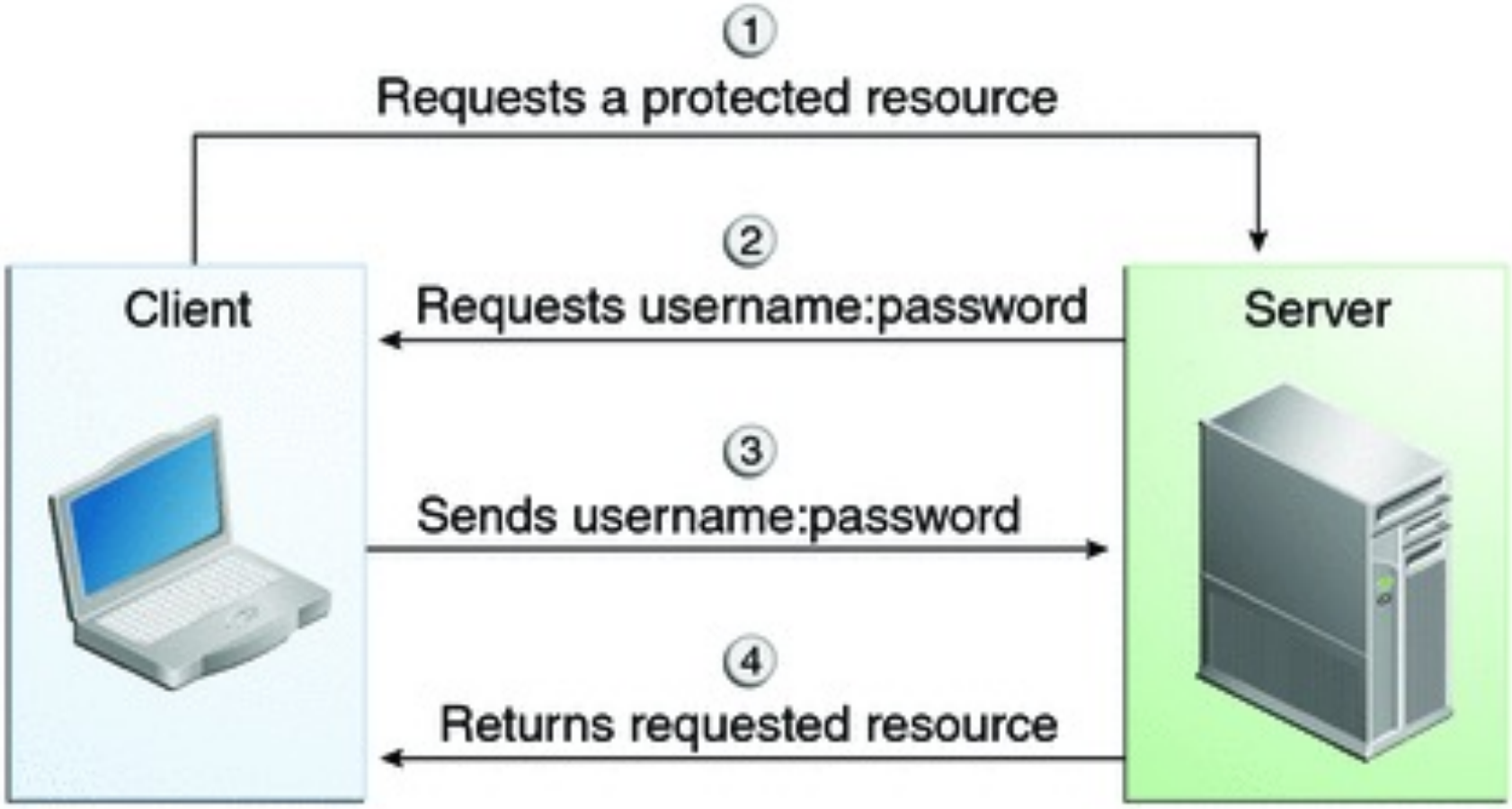
```
@CustomFormAuthenticationMechanismDefinition(
  loginToContinue = @LoginToContinue(
    loginPage = "/login.html"))
@ApplicationScoped
public class AppConfig{}
```

- Customizing the login page
- Implementing backing authentication process

Database Identity Store

```
@DatabaseIdentityStoreDefinition(
  dataSourceLookup = ...,
  callerQuery = ...,
  groupsQuery = ...,
  priority=30)
@ApplicationScoped
public class AppConfig {
}
```

BASIC Authentication



BASIC Authentication

```
@ApplicationScoped
@ApplicationPath("/resources")
@BasicAuthenticationMechanismDefinition
public class MyRestApp extends Application {
}
```

BASIC Authentication

```
@ApplicationScoped
@ApplicationPath("/resources")
@BasicAuthenticationMechanismDefinition
@DatabaseIdentityStoreDefinition(...)
public class MyRestApp extends Application {
}
```

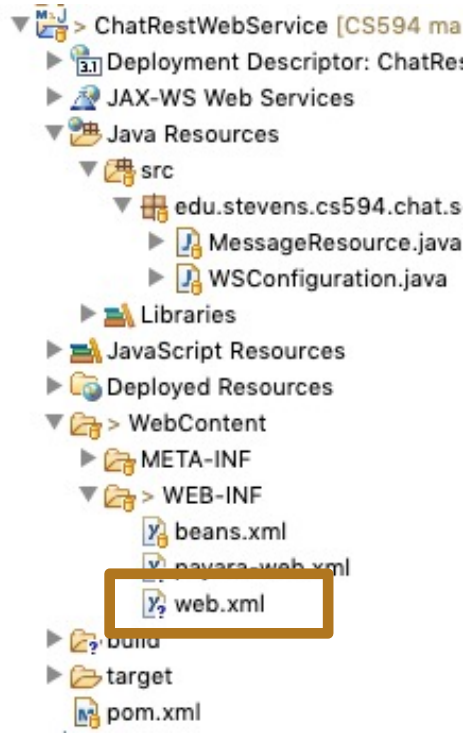
BASIC Authentication

```
@ApplicationScoped
@ApplicationPath("/resources")
@DeclareRoles({ "poster" })
@BasicAuthenticationMechanismDefinition
@DatabaseIdentityStoreDefinition(...)
public class MyRestApp extends Application {
}
```

BASIC Authentication

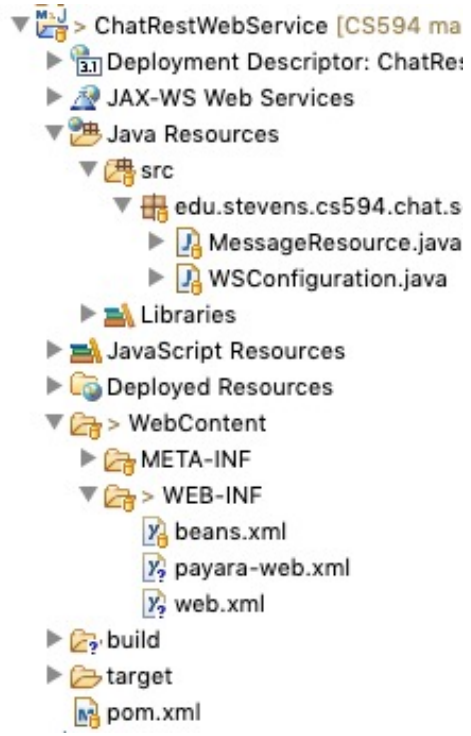
```
@Path("/resources")
@RequestedScoped
public class MessageResource {
    @POST
    @Path(“messages”)
    @Consumes(“application/xml”)
    @RolesAllowed(“poster”)
    public Response addMessage(
            MessageRepresentation rep) {
        ...
    }
}
```

Requiring HTTPS




```
<web-app>
   <security-constraint>
      <display-name>ChatClient</display-name>
      <web-resource-collection>
         <web-resource-name>chat</web-resource-n
         <description/>
         <url-pattern>/resources/forums/*</url-p
      </web-resource-collection>
      <user-data-constraint>
         <transport-guarantee>CONFIDENTIAL</transport-guarantee>
      </user-data-constraint>
   </security-constraint>
</web-app>
```

BASIC Authentication via




```
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>chat-realm</realm-name>
</login-config>

<security-constraint>
    <display-name>ChatClient</display-name>
    <web-resource-collection>
        <web-resource-name>chat</web-resource-name>
        <description/>
        <url-pattern>/resources/forums/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

BASIC Authentication via Servlet

```
<login-config>
   <auth-method>BASIC</auth-method>
   <realm-name>chat-realm</realm-name>
</login-config>

<security-constraint>
   <display-name>ChatClient</display-name>
   <web-resource-collection> ... </web-resource-collection>
   <auth-constraint>
      <description/>
      <role-name>poster</role-name>
   </auth-constraint>
   <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
   </user-data-constraint>
</security-constraint>
```

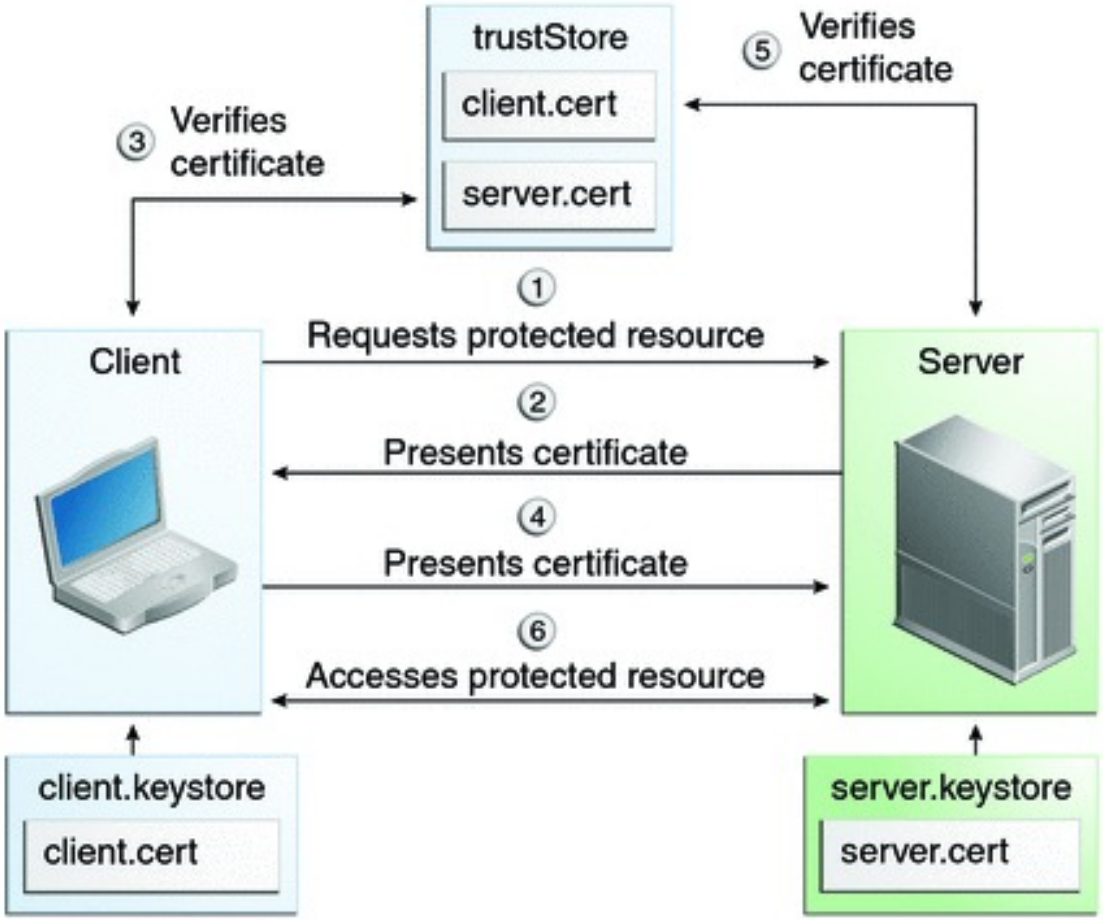
BASIC Authentication via Servlet

```
<login-config>
   <auth-method>BASIC</auth-method>
   <realm-name>chat-realm</realm-name>
</login-config>

<security-constraint> ... </security-constraint>

<security-role>
   <description/>
   <role-name>poster</role-name>
<security-role>
```

Map Roles To Groups: payara-web.xml

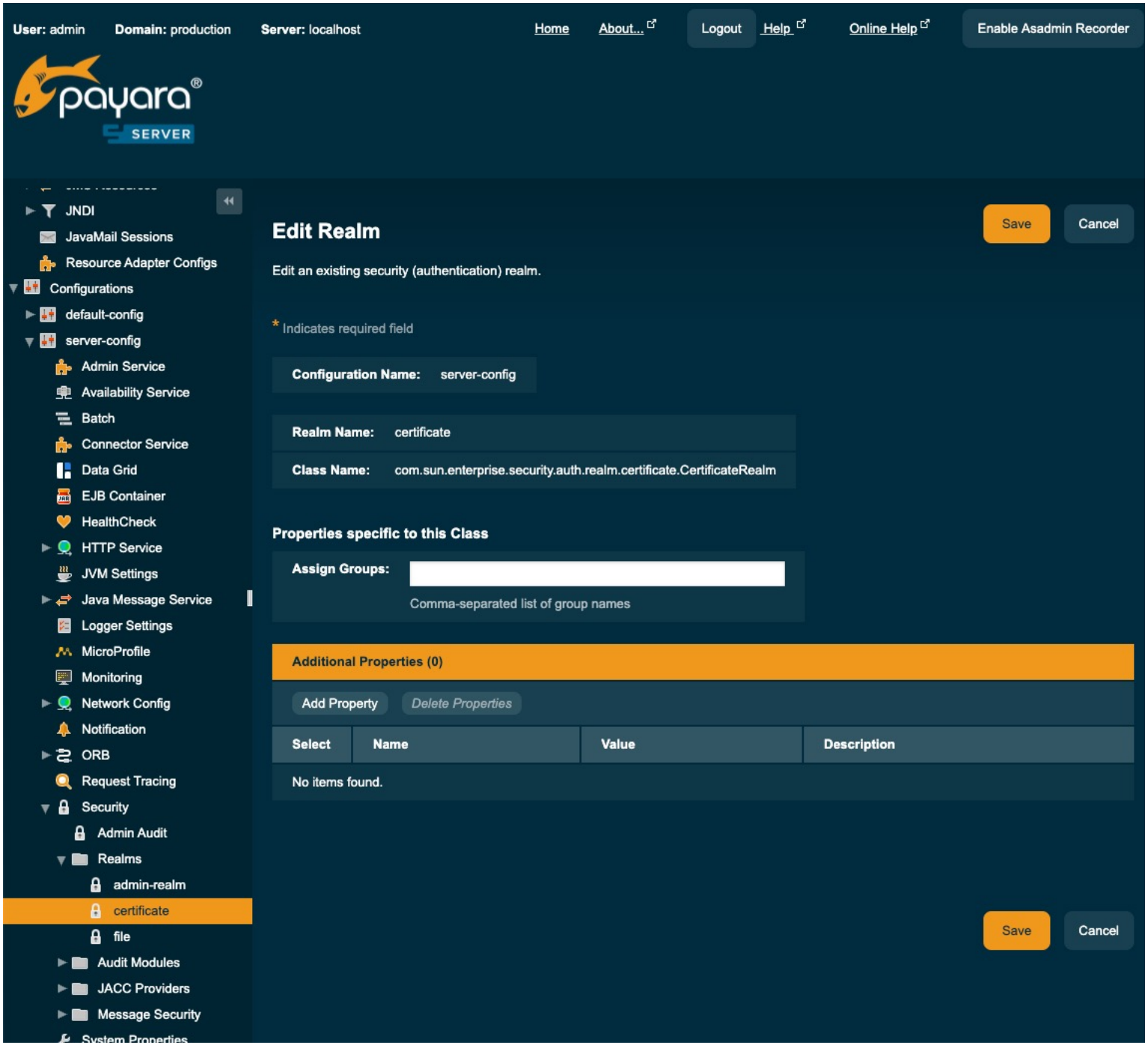
```
<payara-web-app error-url=””>
   <context-root>chat-rest</context-root>
   <security-role-mapping>
      <role-name>poster</role-name>
      <group-name>poster</group-name>
   </security-role-mapping>

   <!-- Default and required for other components. -->
   <class-loader delegate="true" />
   <parameter-encoding default-charset="UTF-8" />
</payara-web-app>
```

Mutual Authentication: Certificate



Certificate Identity Store

```
@ApplicationScoped
@ApplicationPath("/resources")
@DeclareRoles({ ”poster" })
@CertificateAuthenticationMechanismDefinition
@CertificateIdentityStoreDefinition(
        "certificate")
public class MyRestApp extends Application {
}
```


User: admin
Domain: production
Server: localhost
Home
About...
Logout
Help
Online Help
Enable Asadmin Recorder
payara SERVER
JNDI
JavaMail Sessions
Resource Adapter Configs
Configurations
default-config
server-config
Admin Service
Availability Service
Batch
Connector Service
Data Grid
EJB Container
HealthCheck
HTTP Service
JVM Settings
Java Message Service
Logger Settings
MicroProfile
Monitoring
Network Config
Notification
ORB
Request Tracing
Security
Admin Audit
Realms
admin-realm
certificate
file
Audit Modules
JACC Providers
Message Security
Edit Realm
Save
Cancel
Edit an existing security (authentication) realm.
* Indicates required field
Configuration Name: server-config
Realm Name: certificate
Class Name: com.sun.enterprise.security.auth.realm.certificate.CertificateRealm
Properties specific to this Class
Assign Groups:
Comma-separated list of group names
Additional Properties (0)
Add Property
Delete Properties
Select
Name
Value
Description
No items found.
Save
Cancel

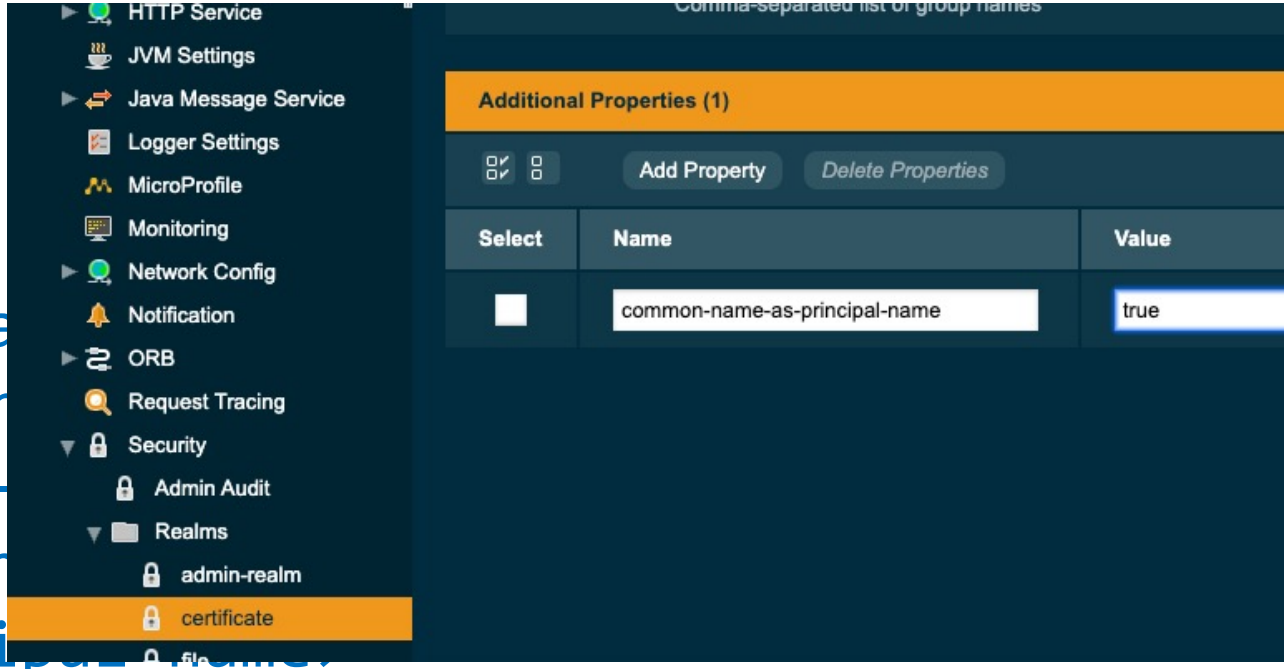
Map Roles To Groups: payara-web.xml

```
<payara-web-app error-url=””>
   <context-root>chat-rest</context-root>
   <security-role-mapping>
      <role-name>poster</role-name>
      <group-name>poster</group-name>
   </security-role-mapping>

   <!-- Default and required for other components. -->
   <class-loader delegate="true" />
   <parameter-encoding default-charset="UTF-8" />
</payara-web-app>
```

Mapping Principals to Roles (payara-web.xml)

```
<payara-web-app error-url="">
   <context-root>chat-rest</context-root>
   <security-role-mapping>
      <role-name>poster</role-name>
      <principal-name>
         C=US, ST=NJ, O=stevens, OU=cs, CN=foo
      </principal-name>
   </security-role-mapping>
</payara-web-app>
```

Map




```
<payara-web-a
    <context-r
    <security-
        <role-r
        <princi
            foo
        </principal-name>
    </security-role-mapping>
</payara-web-app>
```

Server Keystore and Truststore

- Web server is part of application server

- Default keystore: *domain*`/config/keystore.jks`
  - Key alias for admin server: `s1as`
  - Key alias for instance: `glassfish-instance`

- Default truststore: *domain*`/config/cacerts.jks`

Setting up Server Keystores

- Create (offline) root CA
  - Self-signed certificate
  - Offline keystore
- Create admin server and instance keys
  - Certs signed by root CA
  - Server keystore
- Truststore contains root CA certificate
  - Server truststore
  - Distribute to clients as well

Certificate Authentication via Servlet

```
<login-config>
   <auth-method>CLIENT-CERT</auth-method>
   <realm-name>certificate</realm-name>
</login-config>

<security-constraint>
   <display-name>ChatClient</display-name>
   <web-resource-collection>
      <web-resource-name>chat</web-resource-name>
      <description/>
      <url-pattern>/resources/forums/*</url-pattern>
   </web-resource-collection>
   <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
   </user-data-constraint>
</security-constraint>
```

Certificate Authentication via Servlet

```
<login-config>
   <auth-method>CLIENT-CERT</auth-method>
   <realm-name>certificate</realm-name>
</login-config>

<security-constraint>
   <display-name>ChatClient</display-name>
   <web-resource-collection> ... </web-resource-collection>
   <auth-constraint>
      <description/>
      <role-name>poster</role-name>
   </auth-constraint>
   <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
   </user-data-constraint>
</security-constraint>
```

Certificate Authentication via Servlet

```
<login-config>
   <auth-method>CLIENT-CERT</auth-method>
   <realm-name>certificate</realm-name>
</login-config>

<security-constraint> ... </security-constraint>

<security-role>
   <description/>
   <role-name>poster</role-name>
<security-role>
```

Client Side

- Server authentication
  - Based on server SSL cert
- Client authentication: two options
  - BASIC: user name & password in Authorize header
  - CLIENT-CERT: client certificate

Client Initialization: BASIC Auth

```
private Client client;

private URI baseUri;

public ChatClient(SSLContext sslContext,
                  String username, char[] password) {
   HttpAuthenticationFeature feature =
      HttpAuthenticationFeature.basic(username,
                                      new String(password));
   this.client = ClientBuilder.newBuilder()
                              .sslContext(sslContext)
                              .register(feature)
                              .build();
   this.baseUri = ...;
}
```

Client Initialization: BASIC Auth

- Authenticate server with truststore

```
String algorithm =
   TrustManagerFactory.getDefaultAlgorithm();
TrustManagerFactory tmf =
   TrustManagerFactory.getInstance(algorithm);
tmf.init(truststore);

SSLContext context = SSLContext.getInstance("TLS");
context.init(null, tmf.getTrustManagers(), null);
```

Client Authentication: Certs

- Server authentication
  - Based on server SSL cert
  - App server keystore
  - Client truststore
- Client authentication
  - CLIENT-CERT: client certificate
  - Client keystore

Client Initialization: BASIC Auth

- Authenticate to server with keystore

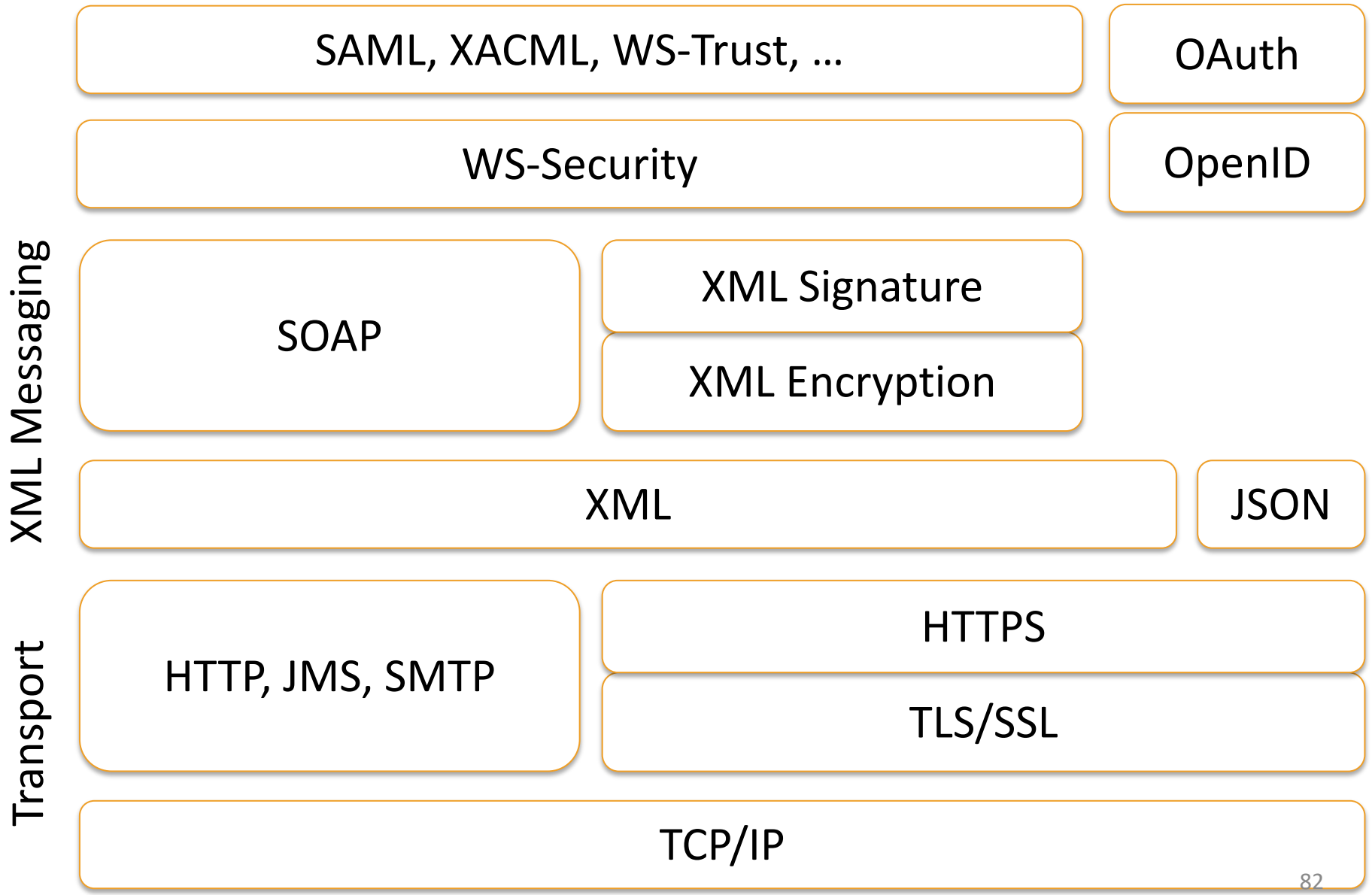
```
TrustManagerFactory tmf = ...;

KeyManager km =
   new AuthKeyManager(alias, password);
KeyManager[] keyManagers = { km };

SSLContext context = SSLContext.getInstance("TLS");
context.init(keyManagers,
             tmf.getTrustManagers(),
             null);
```

Client Initialization: Certs

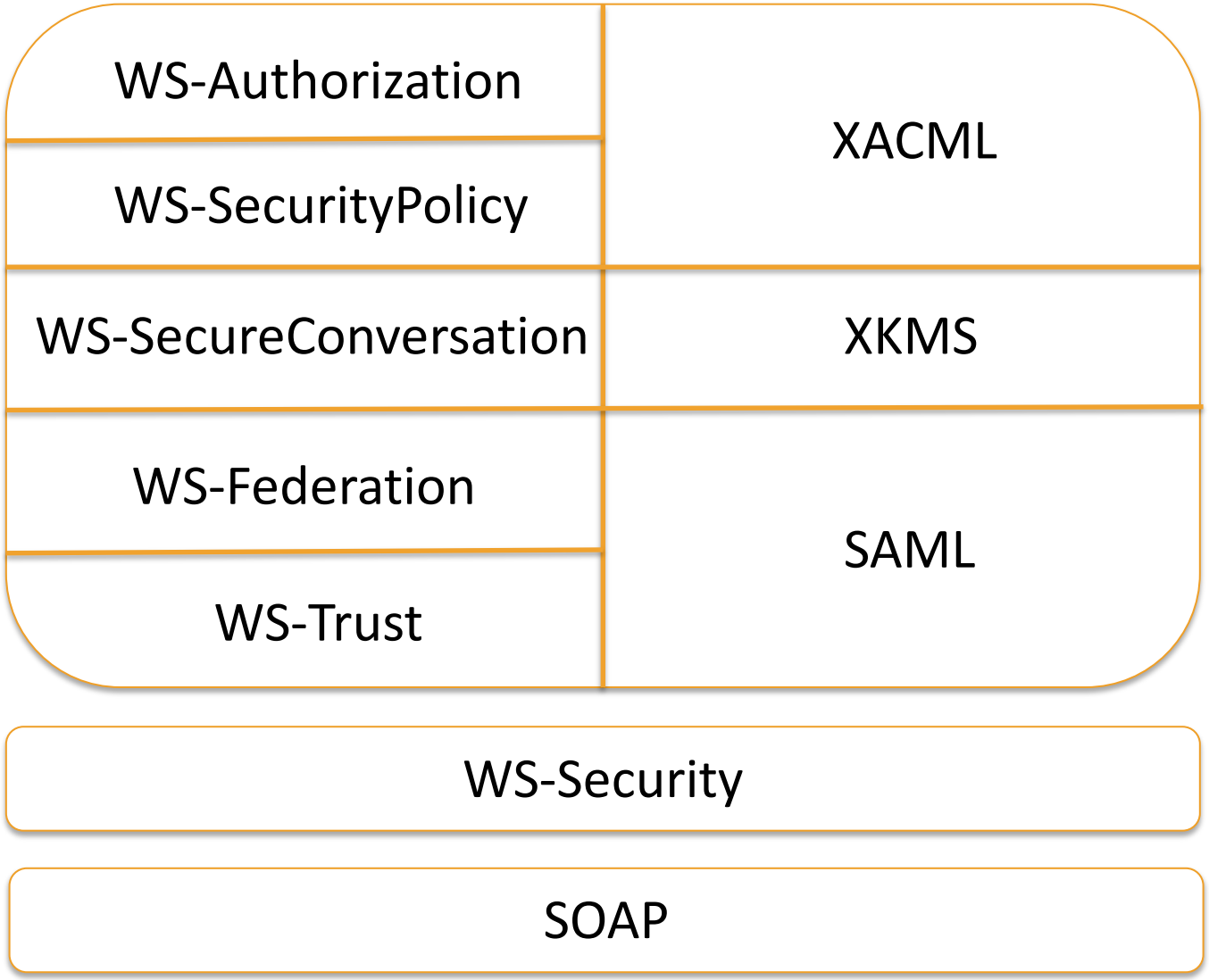
```
private Client client;

private URI baseUri;

public ChatClient(SSLContext sslContext) {
   this.client = ClientBuilder.newBuilder()
                              .sslContext(sslContext)
                              .build();
   this.baseUri = ...;
}
```

SOAP-BASED WEB SERVICE SECURITY

Web Service Security Standards



SOAP Security Stack



Main Security Specifications

- XML Signature (XMLDSIG)
  - Message Integrity and Sender/Receiver Identification
- XML Encryption (XMLENC)
  - Message Confidentiality
- WS-Security (WSS)
  - Securing SOAP Messages
- SAML & XACML
  - Security metadata exchange & access control
- OpenID & OAuth
  - RESTful SSO and 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

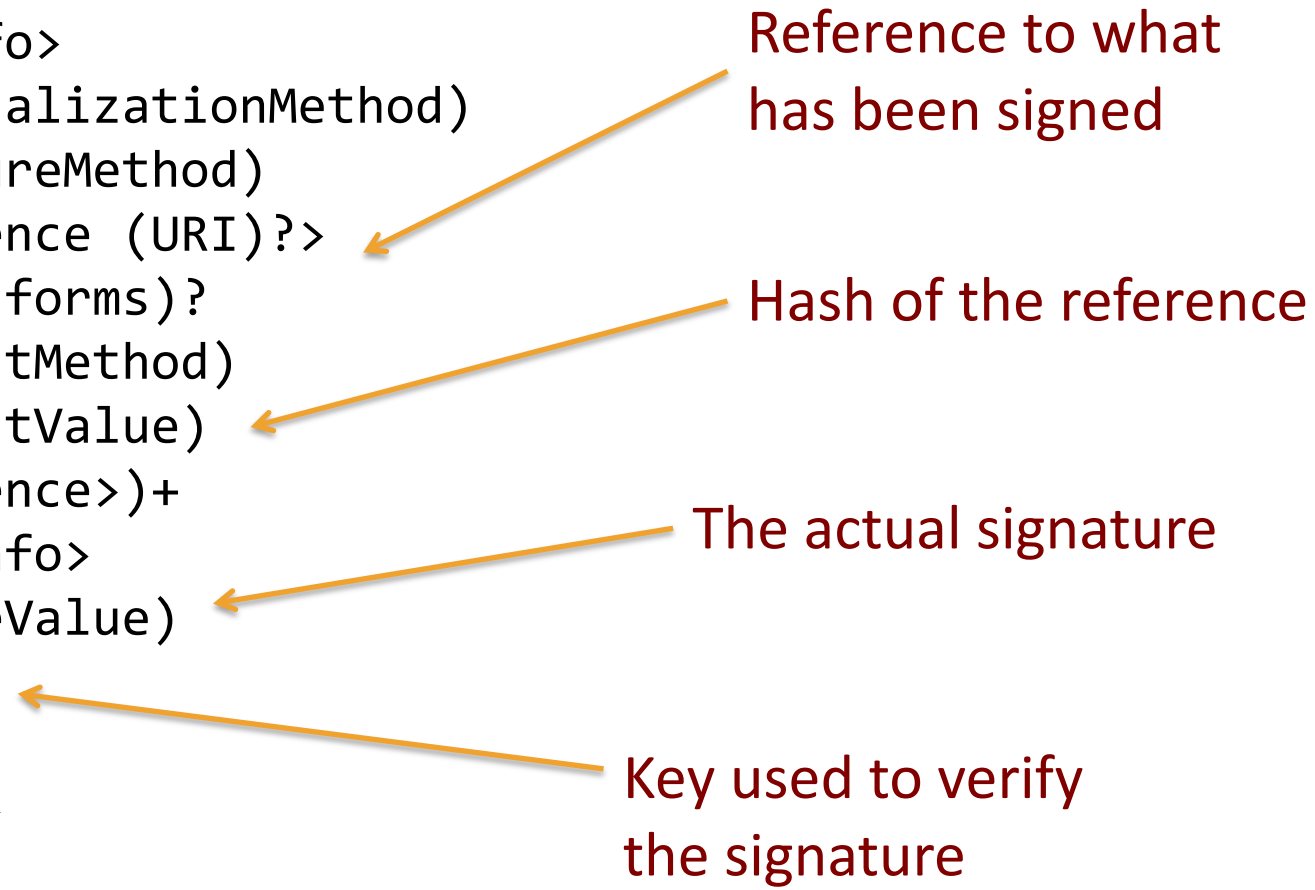
XML SIGNATURE

XML Signature

- 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

XML Signature Structure

```
<Signature>
  <SignedInfo>
    (CanonicalizationMethod)
    (SignatureMethod)
    (<Reference (URI)?>
      (Transforms)?
      (DigestMethod)
      (DigestValue)
    </Reference>)+
  </SignedInfo>
  (SignatureValue)
  (KeyInfo)?
  (Object)*
</Signature>
```



XML Signature Simplified Example

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

Getting the Signature

- Reference Generation
  - Dereference the <Reference URL> to access the content
  - Apply the Transforms
  - Compute the <DigestValue> applying the <DigestMethod> to the transformed content
  - Store the result in the <Reference> element
- Signature Generation
  - Create the <SignedInfo> element
  - Transform it to canonical form
  - Compute the <SignatureValue> applying a <SignatureMethod>
  - Bundle it all together with the <KeyInfo> and <Object> elements
- Note: what is actually signed is the <Reference>

Validating the Signature

- Reference Validation
  - Dereference the <Reference URL> to access the content
  - Apply the same Transforms
  - Compute a hash using the same <DigestMethod>
  - **Compare the <DigestValue> with the result.**
- Signature Validation
  - Canonicalize the <SignedInfo> element
  - Get the Key following the <KeyInfo> element
  - Compute the hash with the <SignatureMethod>
  - **Compare it with the <SignatureValue>**

XML Signature and Security

- Integrity of the message content/external resource:
  - Reference validation
- Integrity of the signature
  - Signature validation
- Identity of the source of the document
  - Signature validation
  - Warning: only if using a <SignatureMethod> based on public/private key (rather than a MAC)

XML Signature and Security

- Danger! Transforms modify and filter the data before it is signed
- Only what is signed is secure:
  - Perhaps only payload, not SOAP envelope
- Only what is seen should be signed:
  - Ex: If transform includes XSLT, that should be signed
  - “What you see is what you sign”
- “See” what is signed:
  - See output of transforms

XML ENCRYPTION

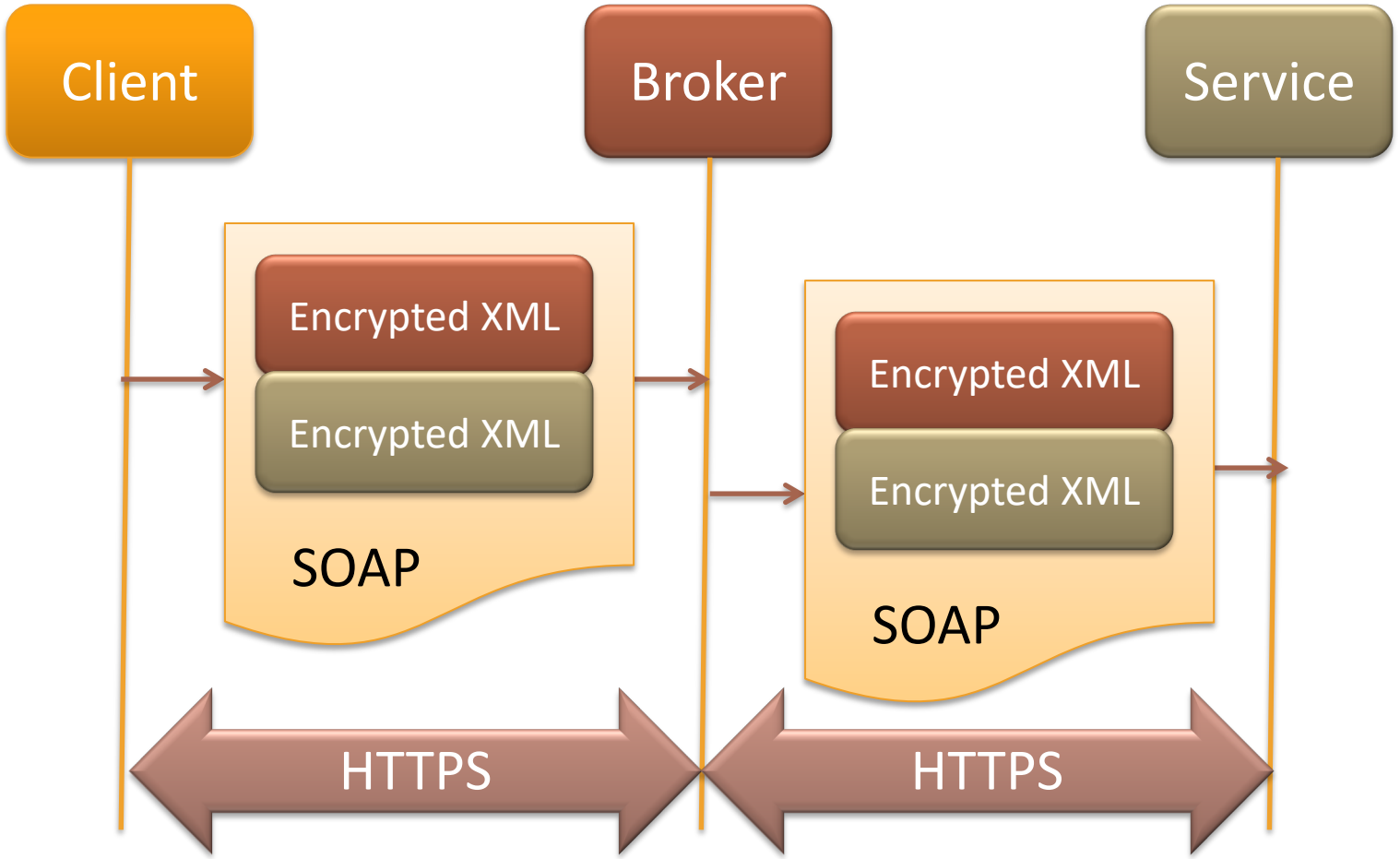
XML Encryption

- 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?

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



XML Encryption Example

Original XML Document

```
<Employee>
  <ID>123456789</ID>
  <Name>John Doe</Name>
  <Salary currency="USD">100000</Salary>
</Employee>
```

Encrypted XML Document

```
<Employee>
  <ID><EncryptedData>…</EncryptedData></ID>
  <Name>John Doe</Name>
  <EncryptedData>…</EncryptedData>
</Employee
```

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

Using XML Encryption

- Choose an algorithm (3DES, AES)
- 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
- Serialize the XML data to a byte stream
  - Element (with tags)
  - Content (tags omitted)
- Encrypt the byte stream
- Encode the result in the <CipherData> element
- Build the <EncryptedData> element with the information required to decrypt it

Using XML Decryption

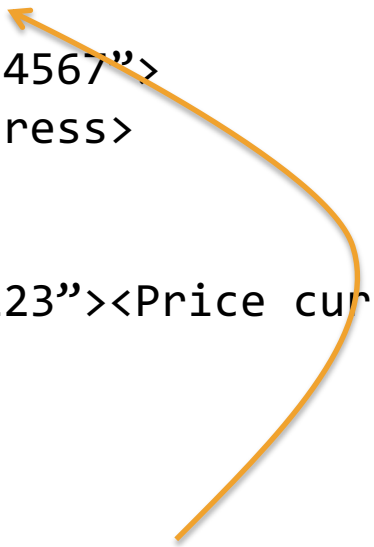
- Determine algorithm (3DES, AES)
- Determine key
  - Key and algorithm could be agreed upon in advance
  - If Key is encrypted, decrypt it (recursive)
- Decrypt data
  - CipherValue (decode the embedded Base-64 byte stream)
  - CipherReference (dereference the URI and apply the specified Transforms before the data is decrypted)
- Process XML content: parse the serialized XML and substitute the original <EncryptedData> element with the decrypted XML element (or content)
- Process non-XML content described by the MimeType and Encoding attributes of the <EncryptedData> element.

XML ENCRYPTION WITH XML SIGNATURE

- XML Encryption: Confidentiality
- XML Signature: Integrity
- Problem: in which order should they be applied?
  - If encryption metadata is sent in clear, could be corrupted to prevent decryption
  - If signatures are sent in the clear, attackers could strip them from a message

Example: Encrypt the Signed Data

```
<Document>
  <Order id="order">
    <Customer id="1234567">
      <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>
```



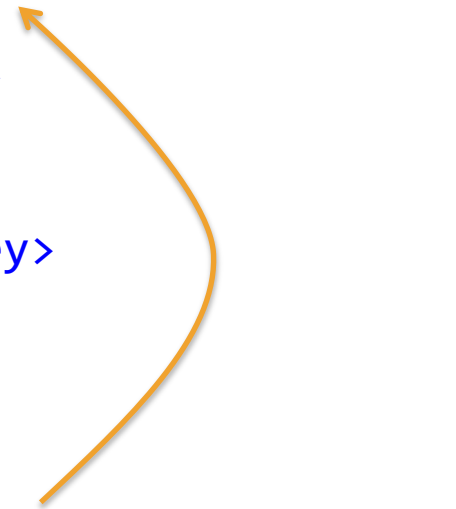
Example: Encrypt the Signed Data

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

- Order is clear: (1) decrypt (2) verify signature
- Problem: Encryption metadata is not protected

Example: Sign the Encrypted Data

```
<Document>
  <EncryptedData id="encryptedData">
    <CipherText>
      <CipherValue>…</CipherValue>
    </CipherText>
    <KeyInfo>
      <EncryptedKey>…</EncryptedKey>
    <KeyInfo>
  </EncryptedData>
  <Signature>
    <SignedInfo>
      <Reference URI="#encryptedData">…</Reference>
    </SignedInfo>
    <SignatureValue>…</SignatureValue>
    <KeyInfo><X509Data>…</X509Data></KeyInfo>
  </Signature>
</Document>
```



Decrypt Transform in XML Signature

- May not be clear in which order signature validation and decryption should be applied.
- Decrypt transform: distinguish whether the signature applies to the <EncryptedData> or to the decrypted data.

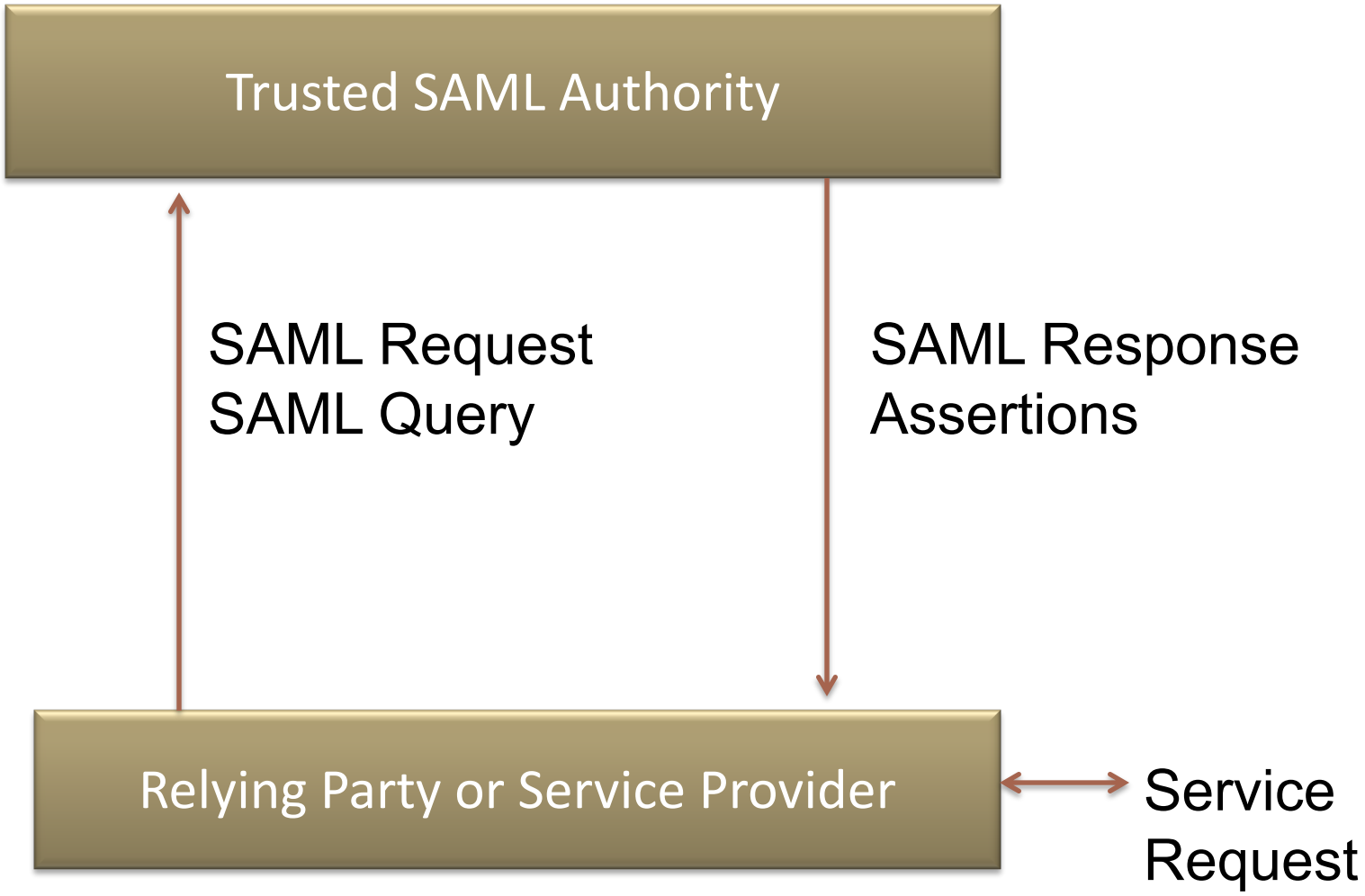
```
<Transform Algorithm="…decrypt#XML">
  <Except URI="#encryptedDataID">
</Transform>
```

- Default processing always applies decryption before signature verification
- Decrypt all referenced <EncryptedData> elements *except the one identified by the <Except> element.*

SAML: SECURITY ASSERTION MARKUP LANGUAGE

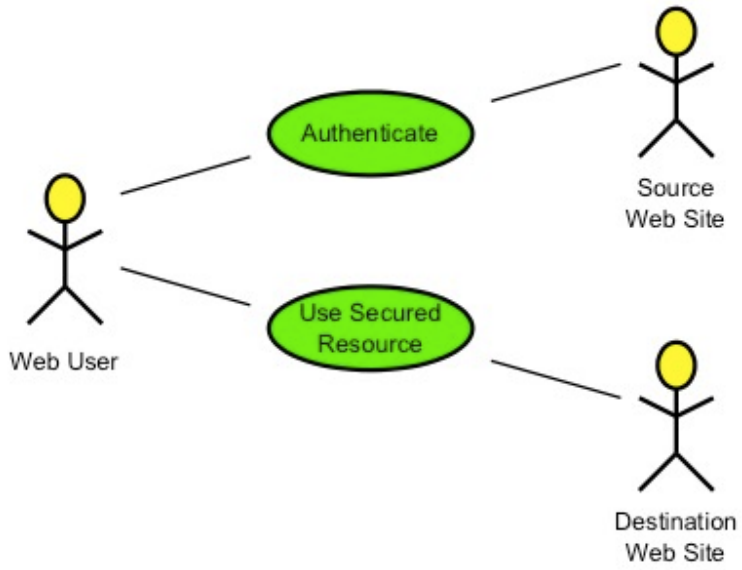
SAML Overview

- Goal: enable loosely coupled identity management.
- Solution: define a format and protocol for interoperable exchange of security information (or assertions)
- 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”.


Trusted SAML Authority
SAML Request
SAML Query
SAML Response
Assertions
Relying Party or Service Provider
Service
Request

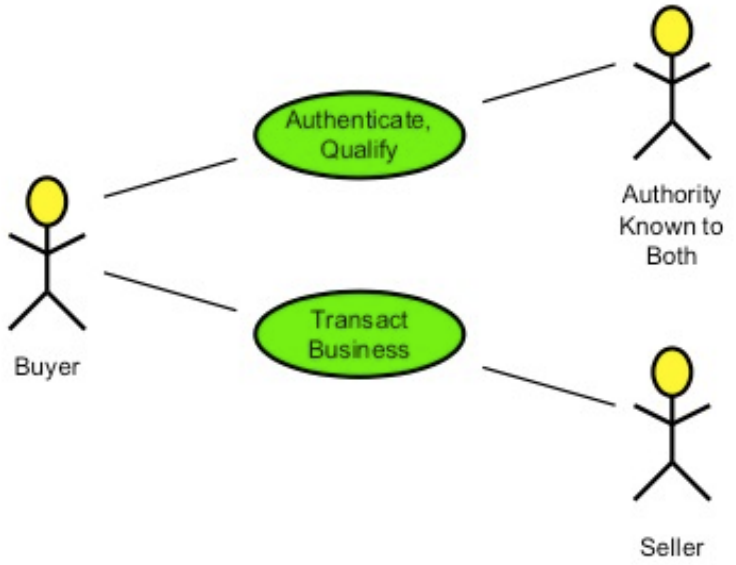
Web SSO Use Case

- Logged-in users of analyst research site SmithCo are allowed access to research produced by sister site JonesCo.
- The user is transferred to a partner's web page
  - The partner is a SAML Authority
- The SAML authentication query is passed as well
- If the SAML Authority is satisfied, then access granted All of this may be over SSL



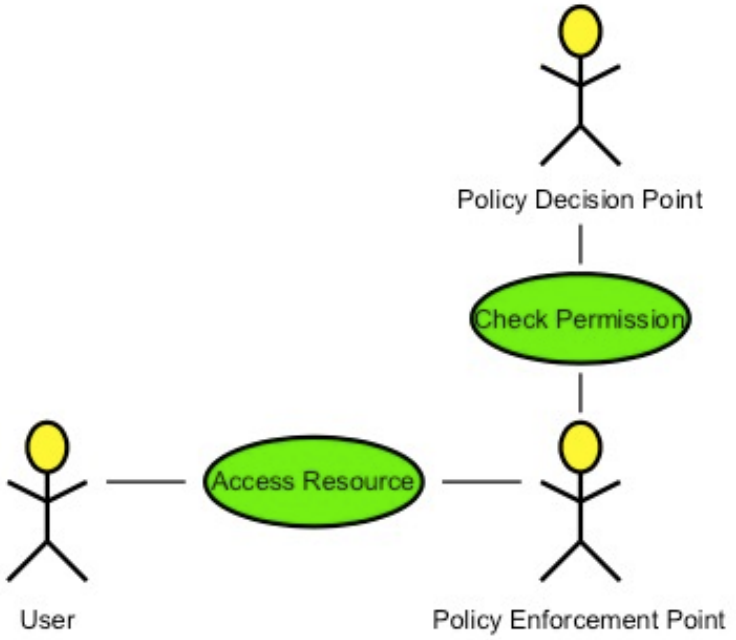
Business Transaction Use Case

- Employees at SmithCo are allowed to order office supplies from OfficeBarn if they are authorized to spend enough.
- The seller (OfficeBarn) may make inquiries on an authority known by both SmithCo and OfficeBarn.
- The authority may vouch for the employee and describe her qualifications.



Authorization Use Case

- A user attempts to access a resource.
- The security domain defines a Policy Enforcement Point and a Policy Decision Point.
- The Policy Enforcement Point makes calls on the Policy Decision Points to check permissions.
- These calls use SAML on a back channel.

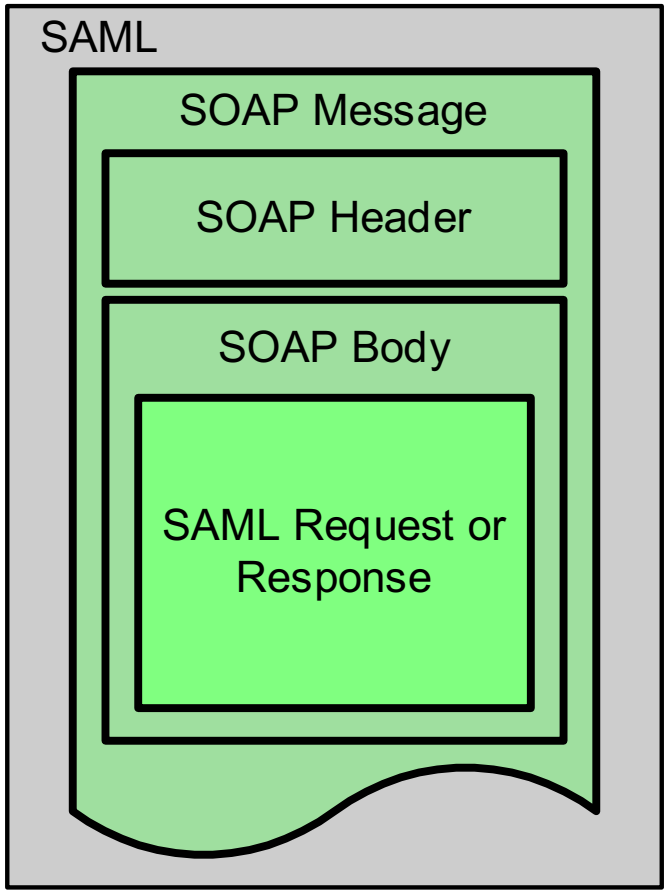


SAML Concepts

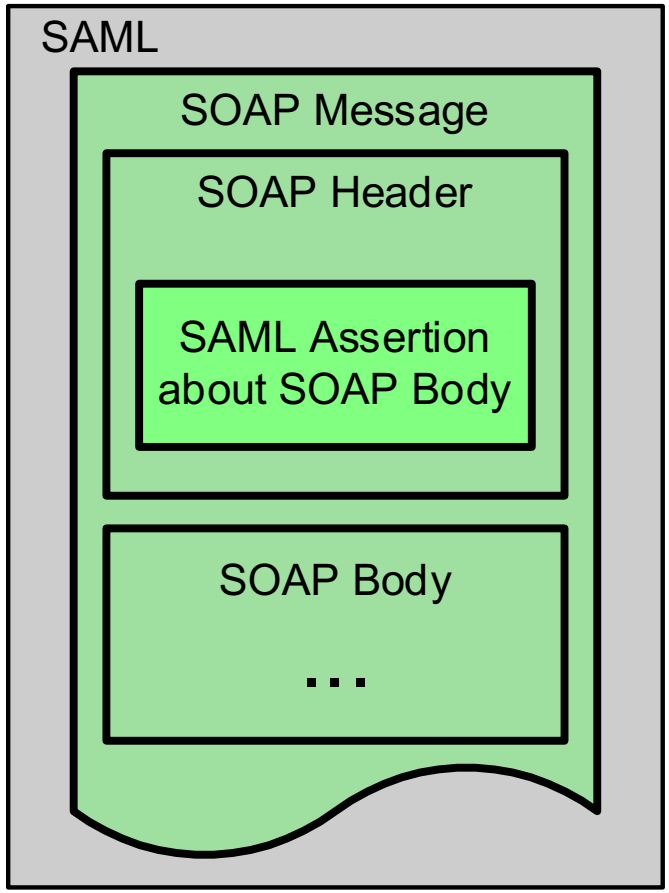
- Security assertions that can be understood across security domains
  - Authentication
  - Attributes
  - Authorization
- Standard protocols to exchange assertions
- SAML bindings:
  - HTTPS required
  - May be used inside SOAP for WS-Security tokens

SOAP Bindings

SOAP-over-HTTP



SOAP Profile



SAML Assertion Metadata Example

```
<Assertion Version="2.0" AssertionID="123042134"
        IssueInstant="2005-11-23..."> 
  <Issuer>saml.stevens.edu</Issuer>
  <Subject>
    <NameID Format="emailAddress">dduggan@stevens.edu</NameID>
    <SubjectConfirmation Method="holder-of-key">
      <SubjectConfirmationData>
        <ds:KeyInfo>…</ds:KeyInfo>
      </SubjectConfirmationData>
    </SubjectConfirmation>
  </Subject>
  <Conditions NotBefore="2011-10-28…"
              NotOnOrAfter="2011-10-29…"><OneTimeUse/>
  </Conditions>
  …statements…
</Assertion>
```

Authentication Assertions

- 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 assertion can be trusted to represent the digital identity of the subject for some period of time

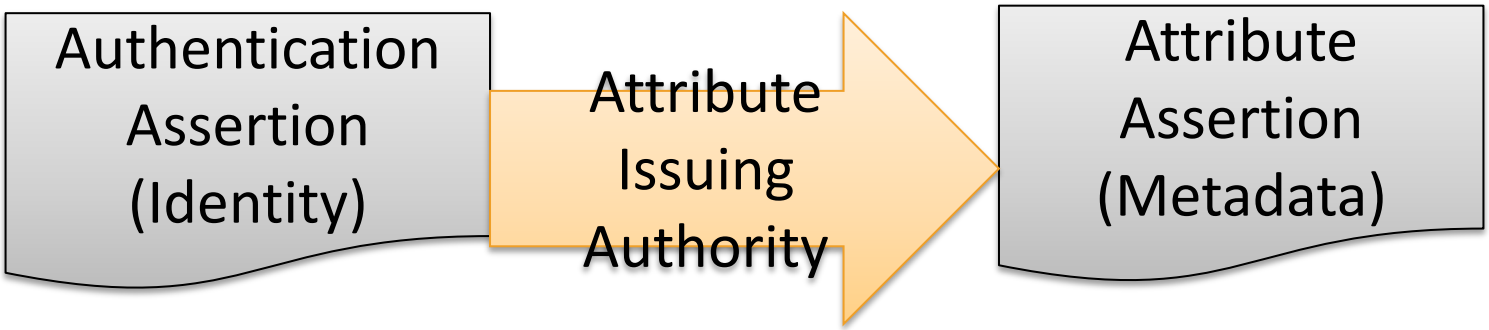
Authentication Assertions

- **Caution:** Actually checking or revoking of credentials is not in scope for SAML!
  - Password exchange
  - Challenge-response
  - Etc.
- It merely lets you link back to acts of authentication that took place previously

Authentication Methods

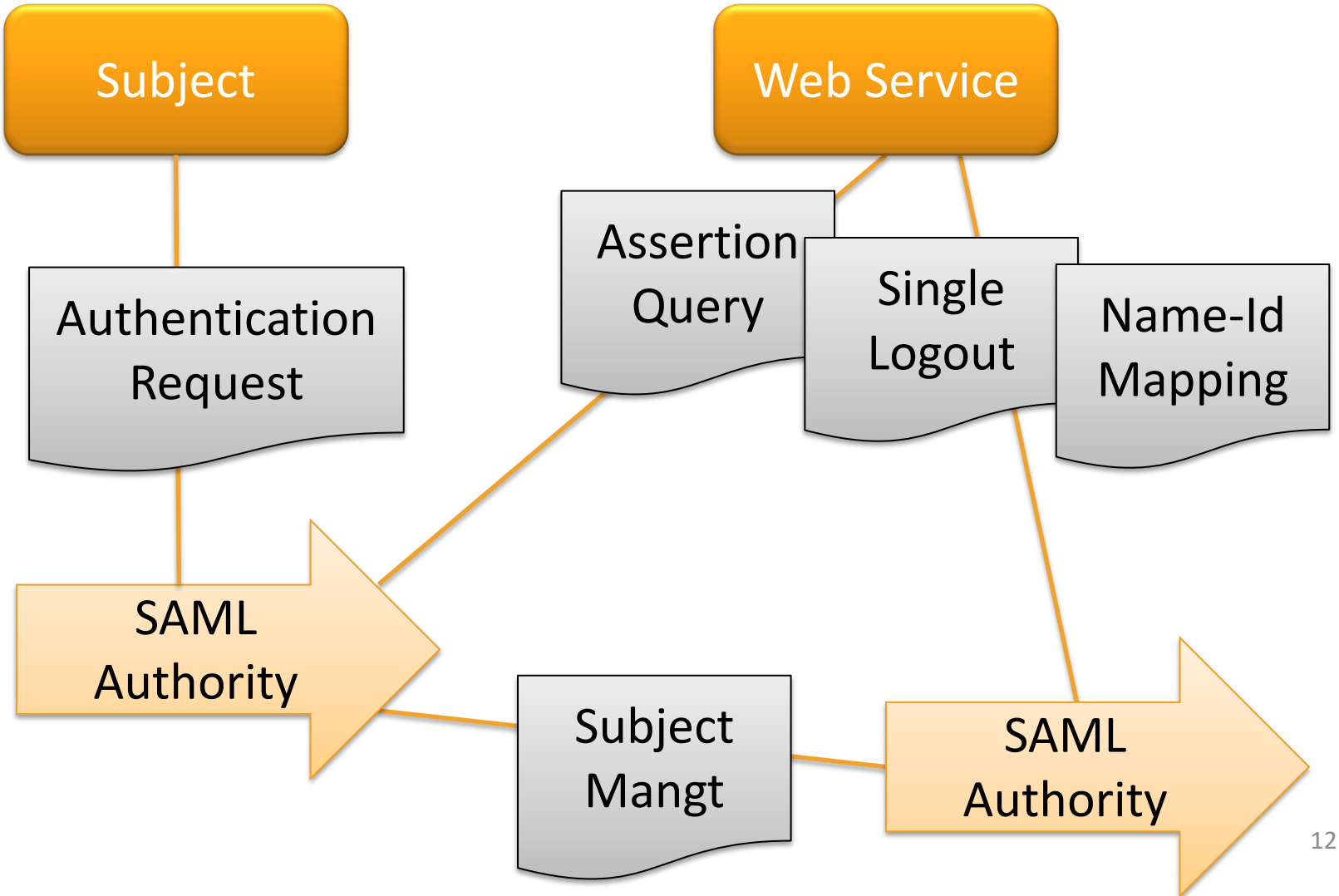
- Internet Protocol Address
- UserName/Password over HTTP or HTTPS
- Secure Remote Password
- IP Address and Username/Password
- SSL/TLS Certificate Based Client Authorization
- Kerberos Ticket
- Public Key (X.509, PGP, SPKI, XML Signature)
- Smartcard: One Factor, Two Factor
- Telephone Number
- Mobile: One Factor, Two Factor
- Previous Session
- Unspecified

Attribute Assertions

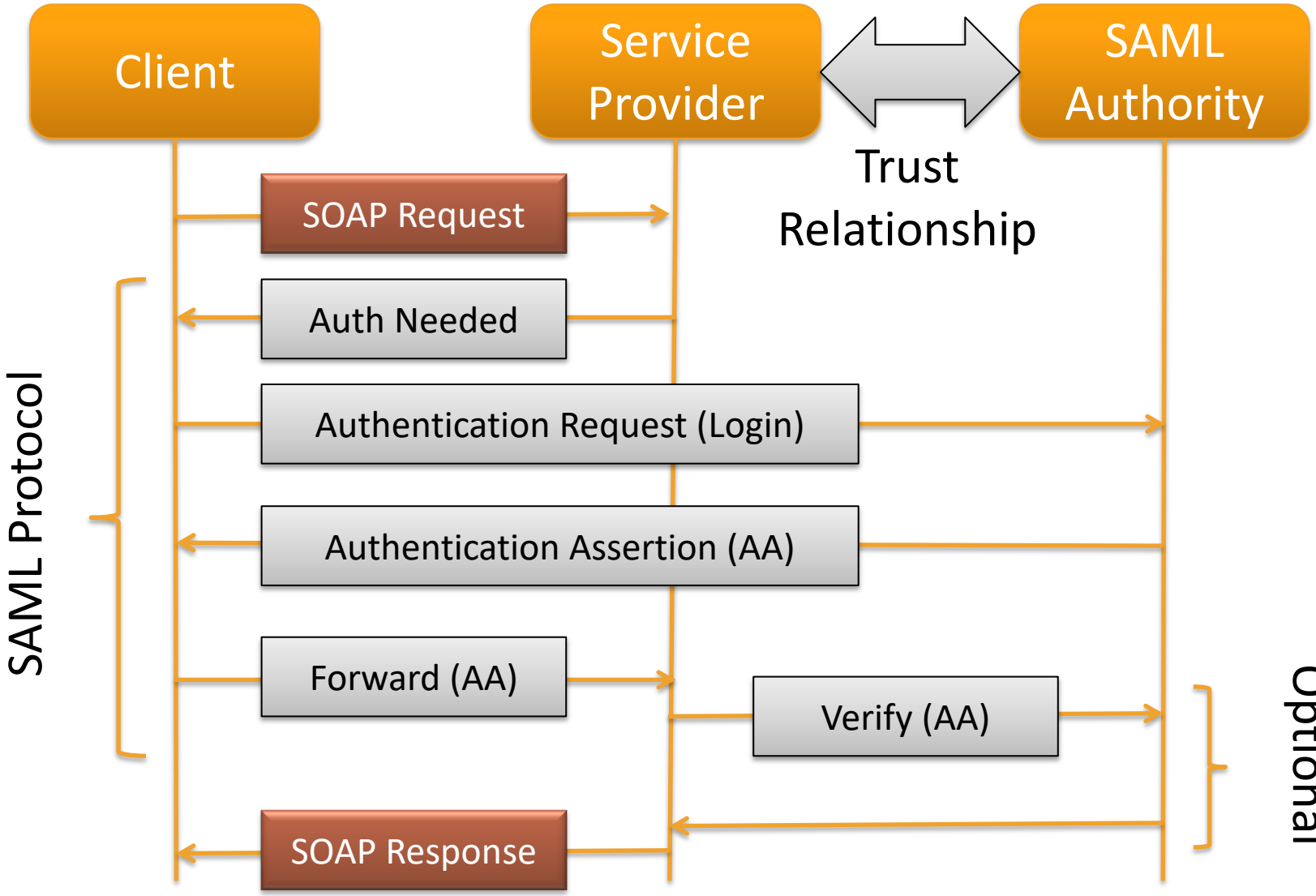


- An authority asserts that the subject is associated with the specified attributes
- SAML profiles standardize access to directories:
  - LDAP/X.500
  - DCE PAC (Privilege Attribute Certificate)
  - XACML (eXtensible Access Control Markup Language)
- Additionally, attributes can model accounting related information: remaining credit, payment status

SAML Protocols



Putting it all together



SAML PROFILES

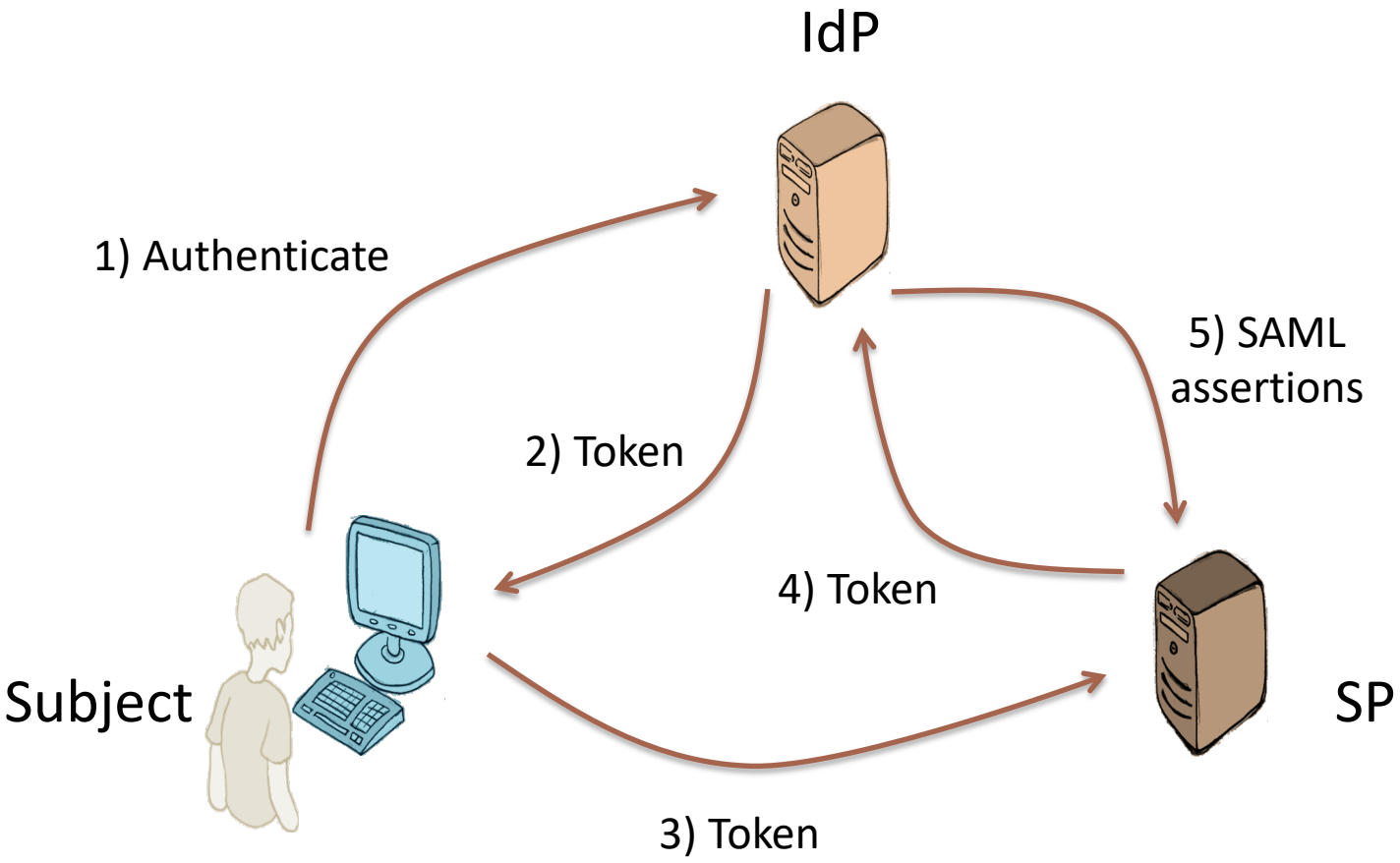
Lower level Use Cases

- Pull (IdP manages tokens)
  1. Subject authenticates with IdP and receives an 8 byte random token.
  2. Subject presents a request for service and the token to SP.
  3. SP passes the token to IdP and receives assertions about Subject.
  4. SP provides Subject with the service.

- Assume a back end channel and everything over SSL.

Lower Level Use Cases

- Pull (IdP manages tokens)



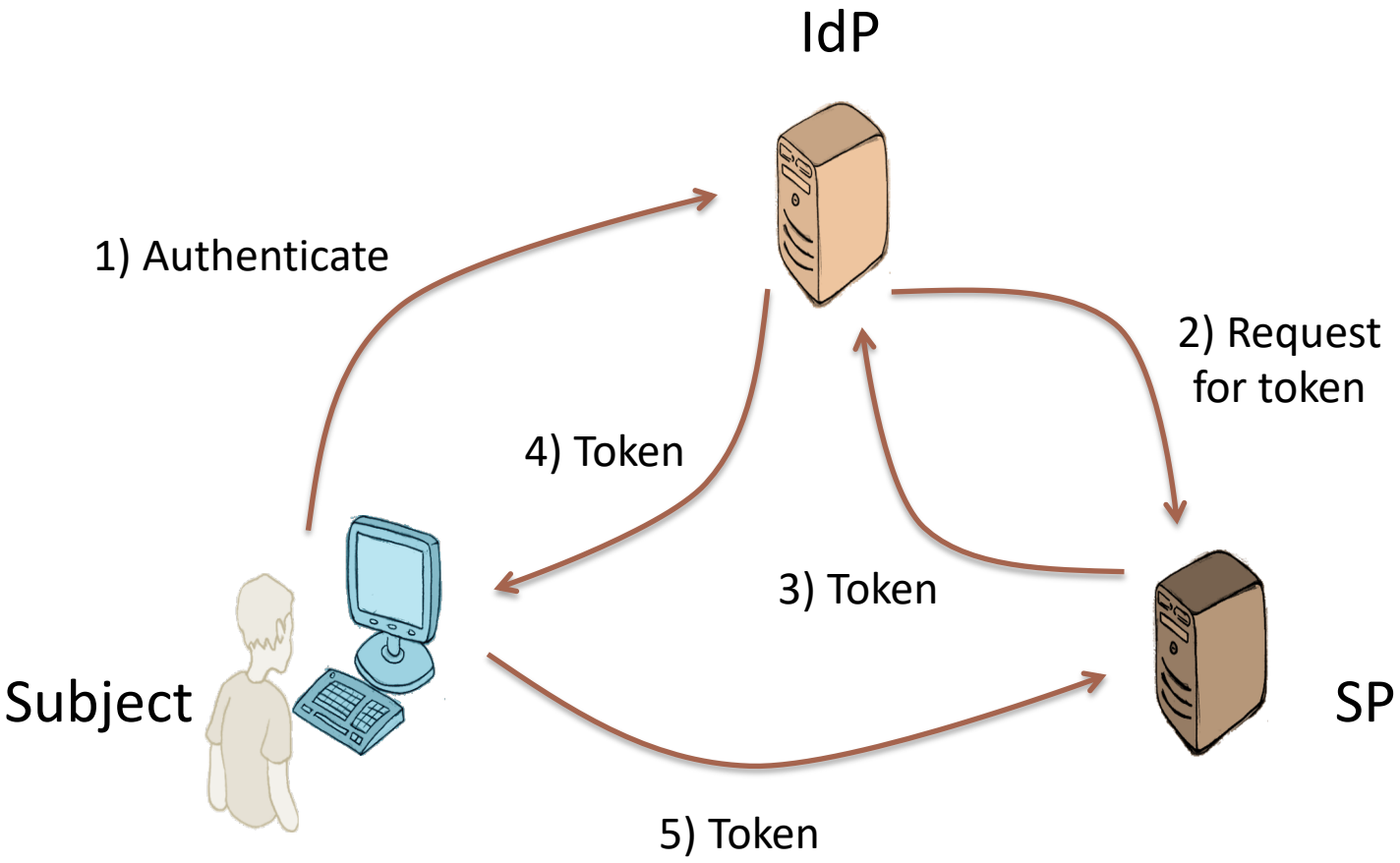
Lower Level Use Cases

- Push (SP manages tokens)
  1. Subject authenticates with IdP and IdP calls SP for SAML token
  2. SP responds with token. He knows she is authentic.
  3. IdP returns token to Subject.
  4. Subject calls SP with token.
  5. SP provides Subject with service.

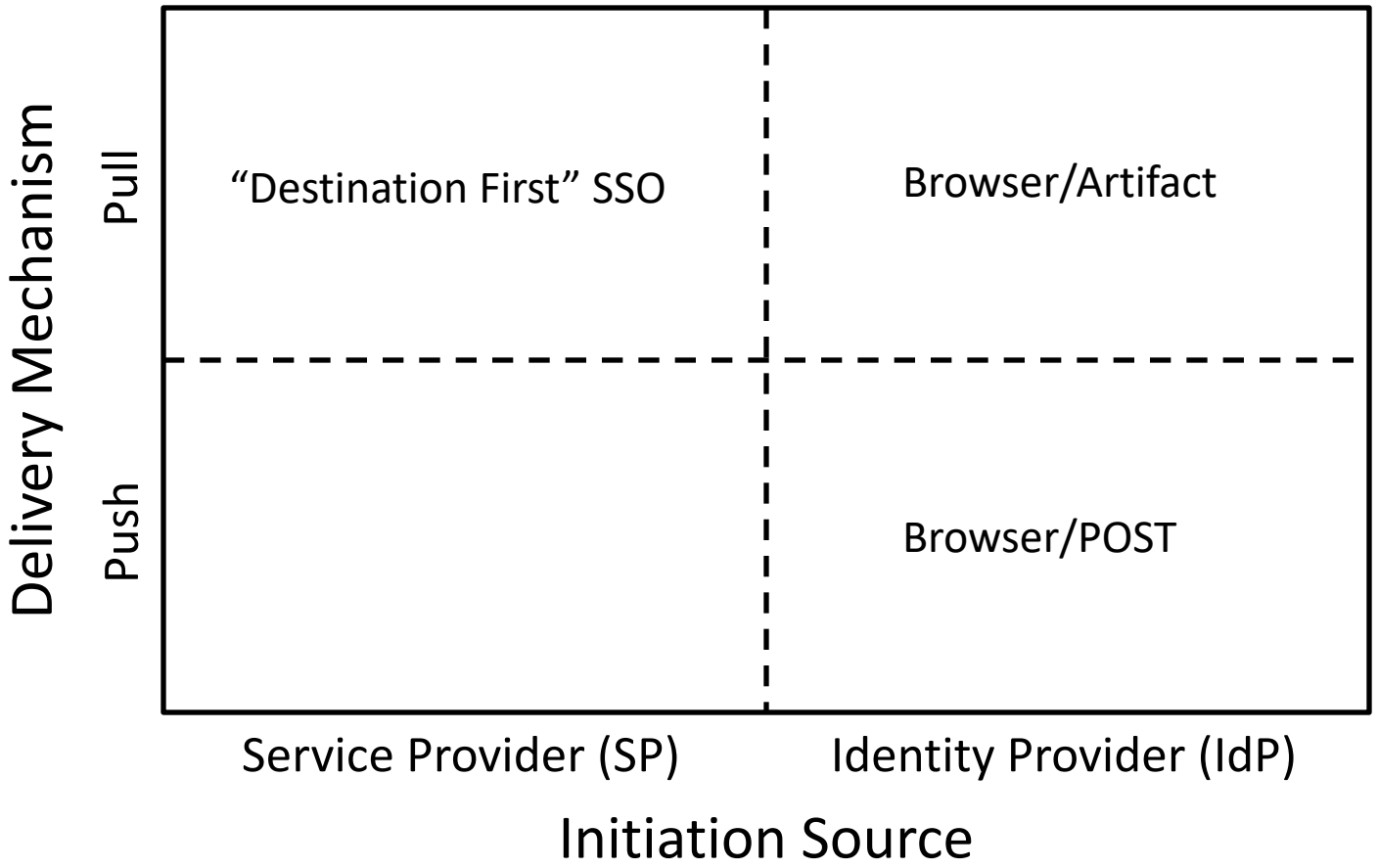
- SP need not handle authentication and may only provide tokens to IdP.

Lower Level Use Cases

- Push (SP manages tokens)



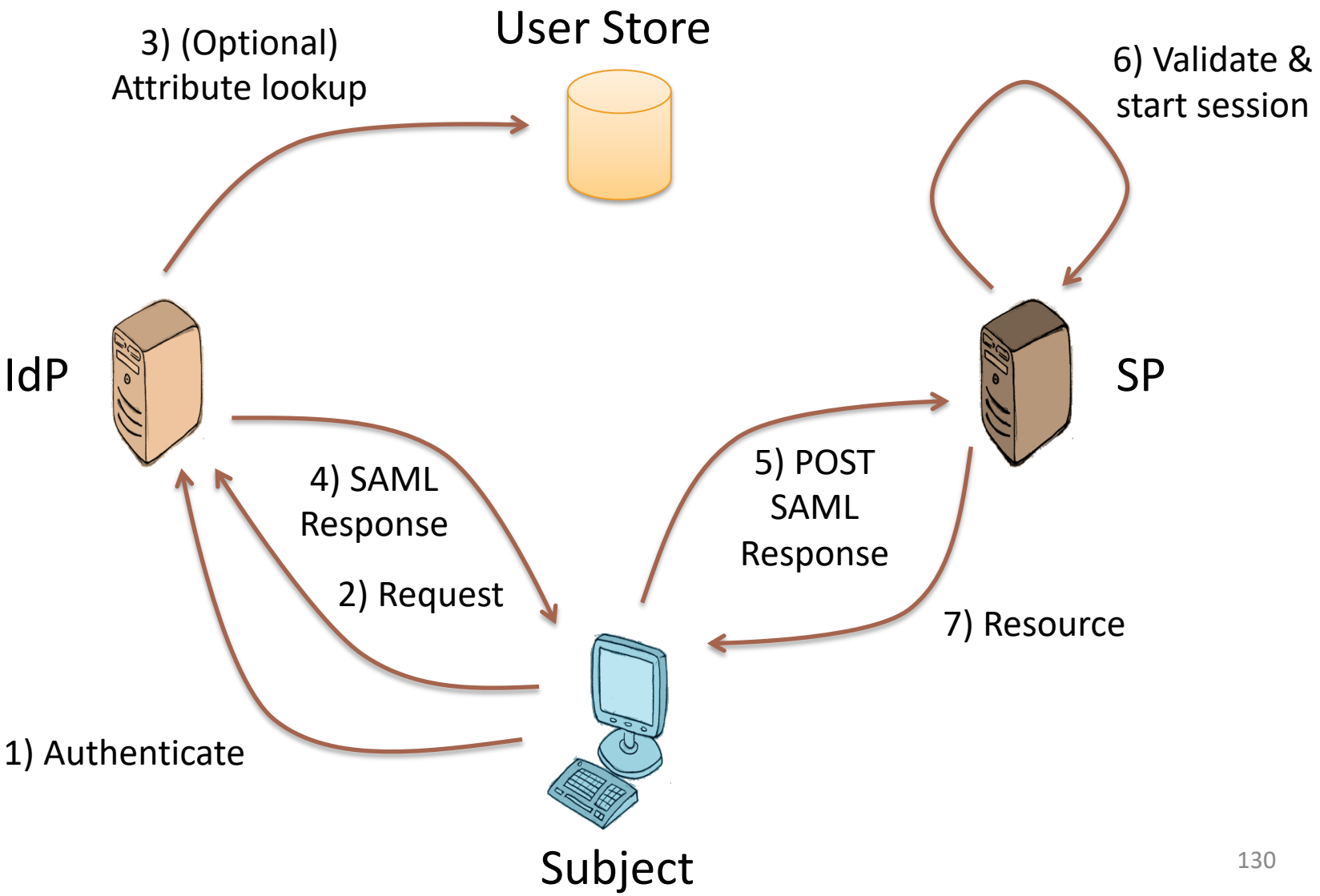
Profile Sources & Delivery Mechanisms (SAML 1.1)



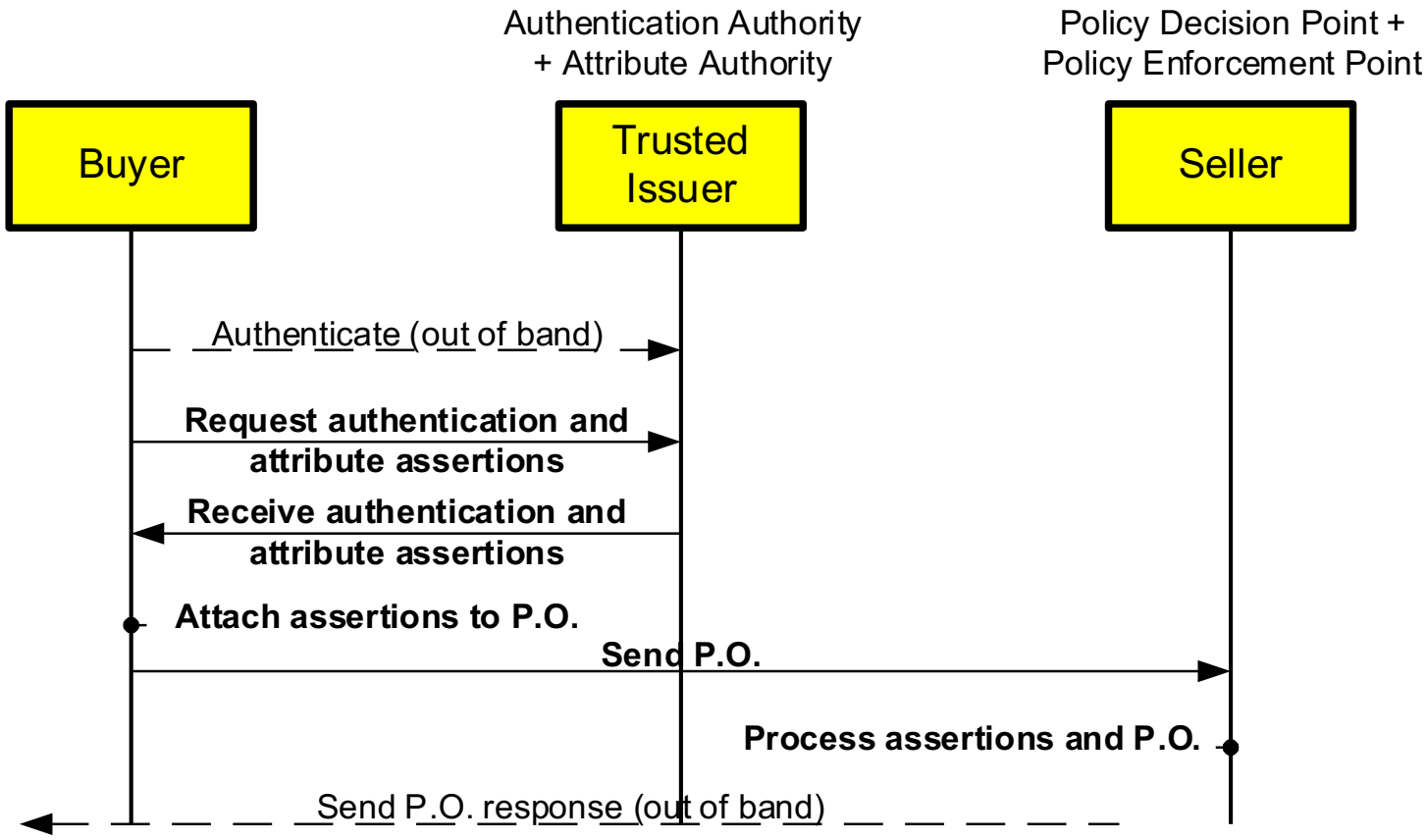
Web browser profiles

- These profiles assume:
  - A standard commercial browser and HTTP(S)
  - User has authenticated to a local source site
  - Assertion's subject refers implicitly to the user
- When a user tries to access a target site:
  - A tiny authentication assertion reference ("artifact") travels with the request so the real assertion can be dereferenced
  - Or the real assertion gets POSTed

SSO: Browser/POST



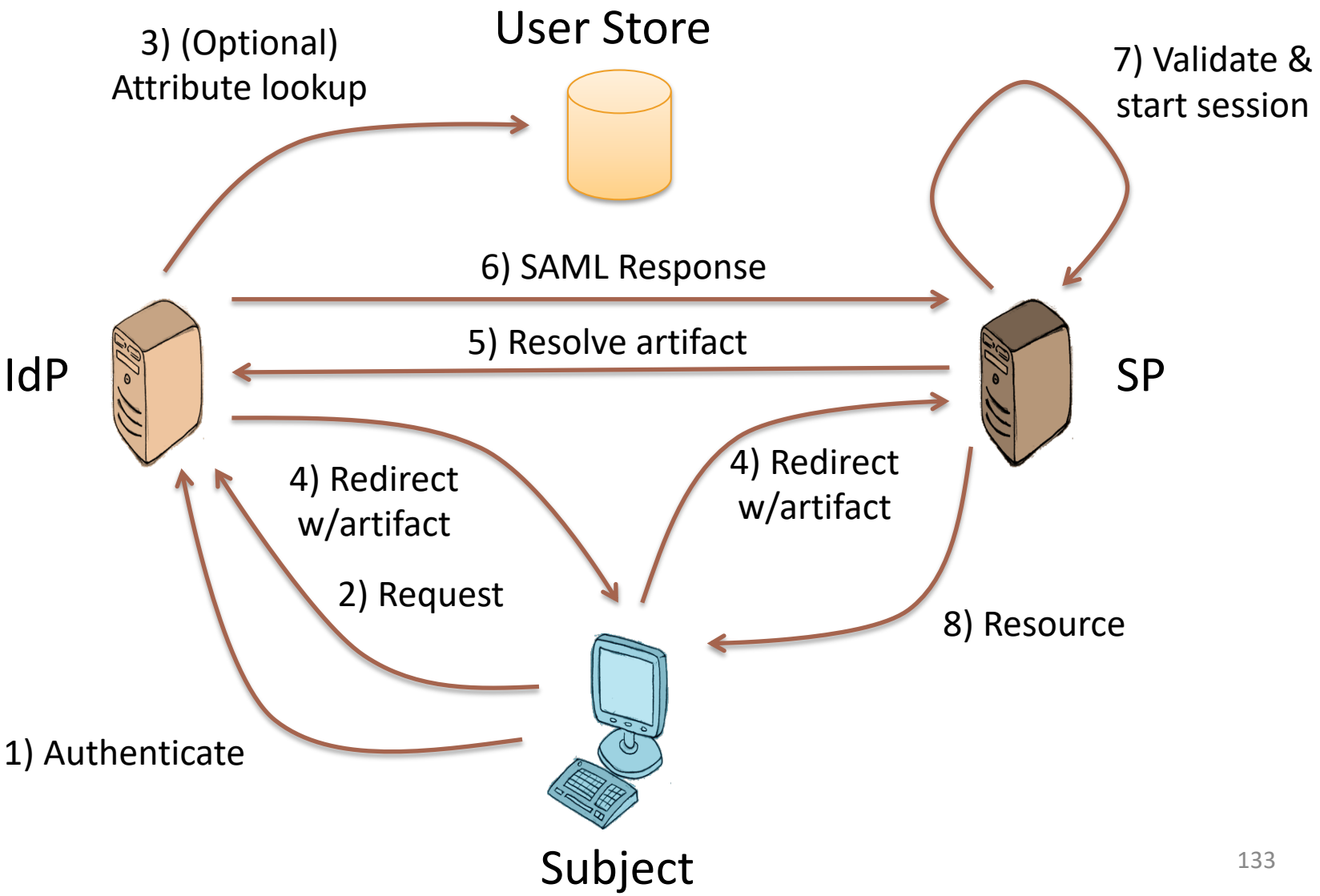
Back Office Transaction Scenario



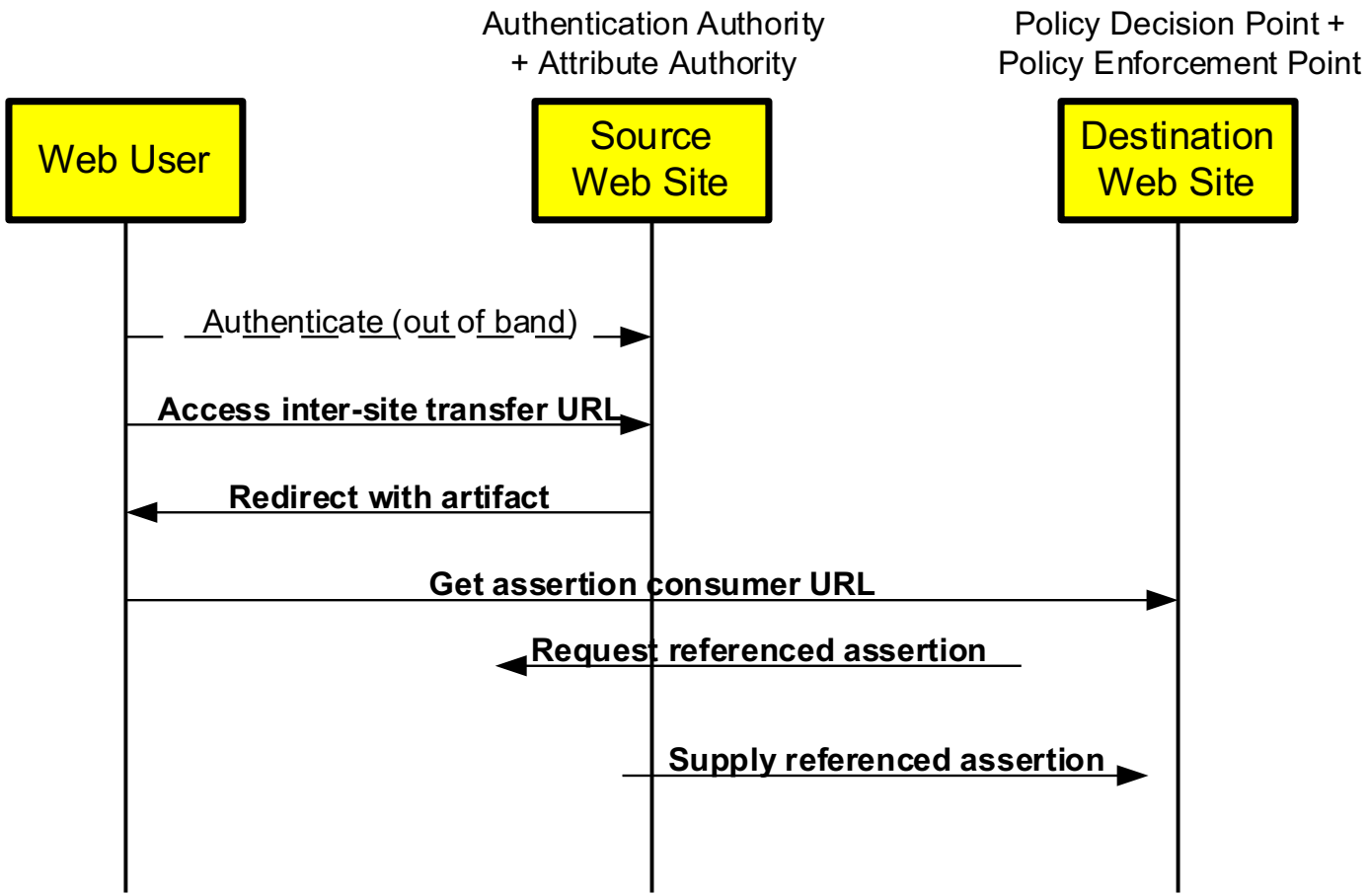
More on the Back Office Transaction Scenario

- An example of attaching SAML assertions to other traffic
- Asymmetrical relationship is assumed
  - Seller is already known to buyer, but buyer is not known to seller, a common situation
  - E.g., server-side certificates might be used to authenticate seller
- If it were symmetrical, additional SAML steps would happen on the right side too
  - This would likely be a different scenario

SSO: Browser/Artifact



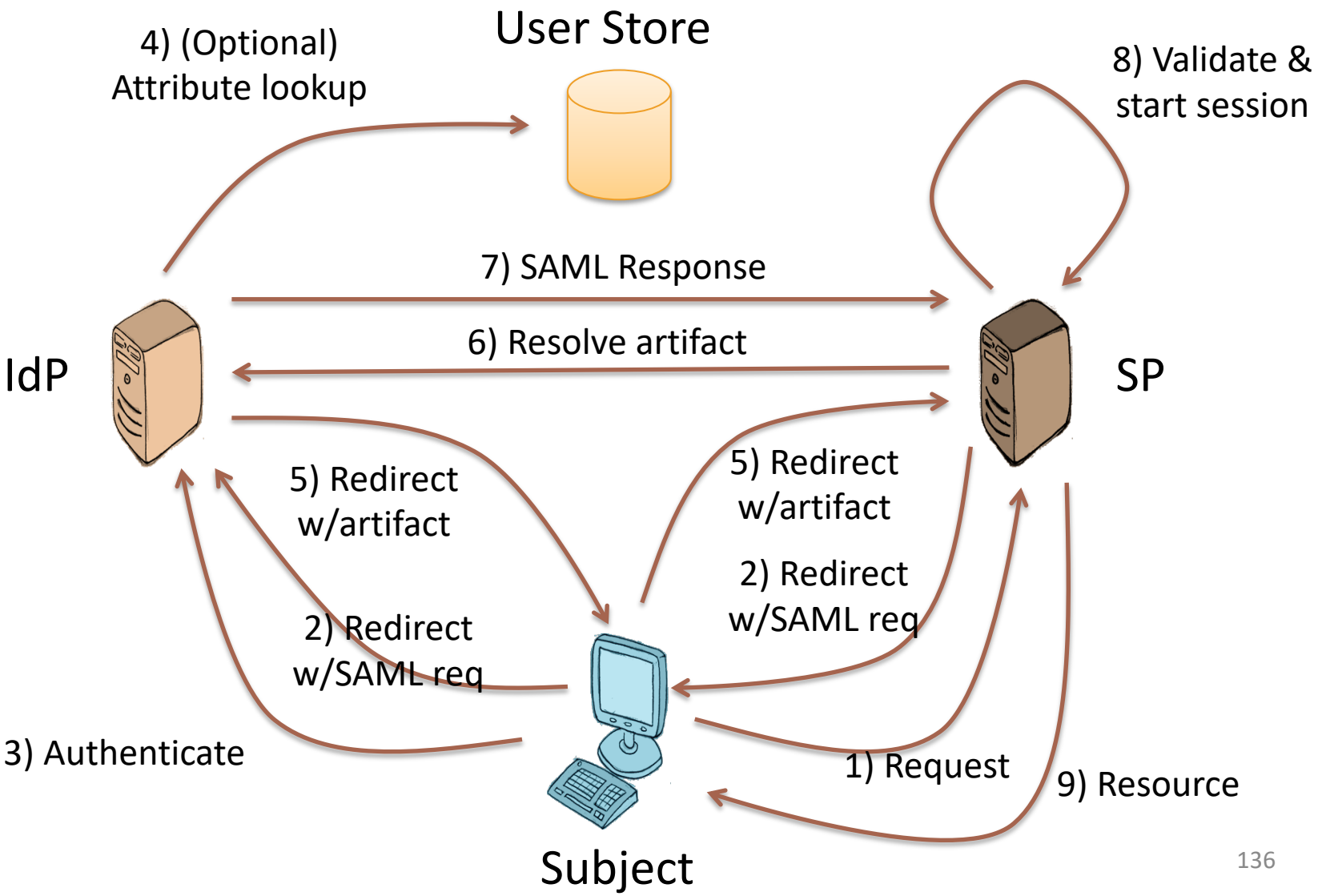
SSO pull scenario



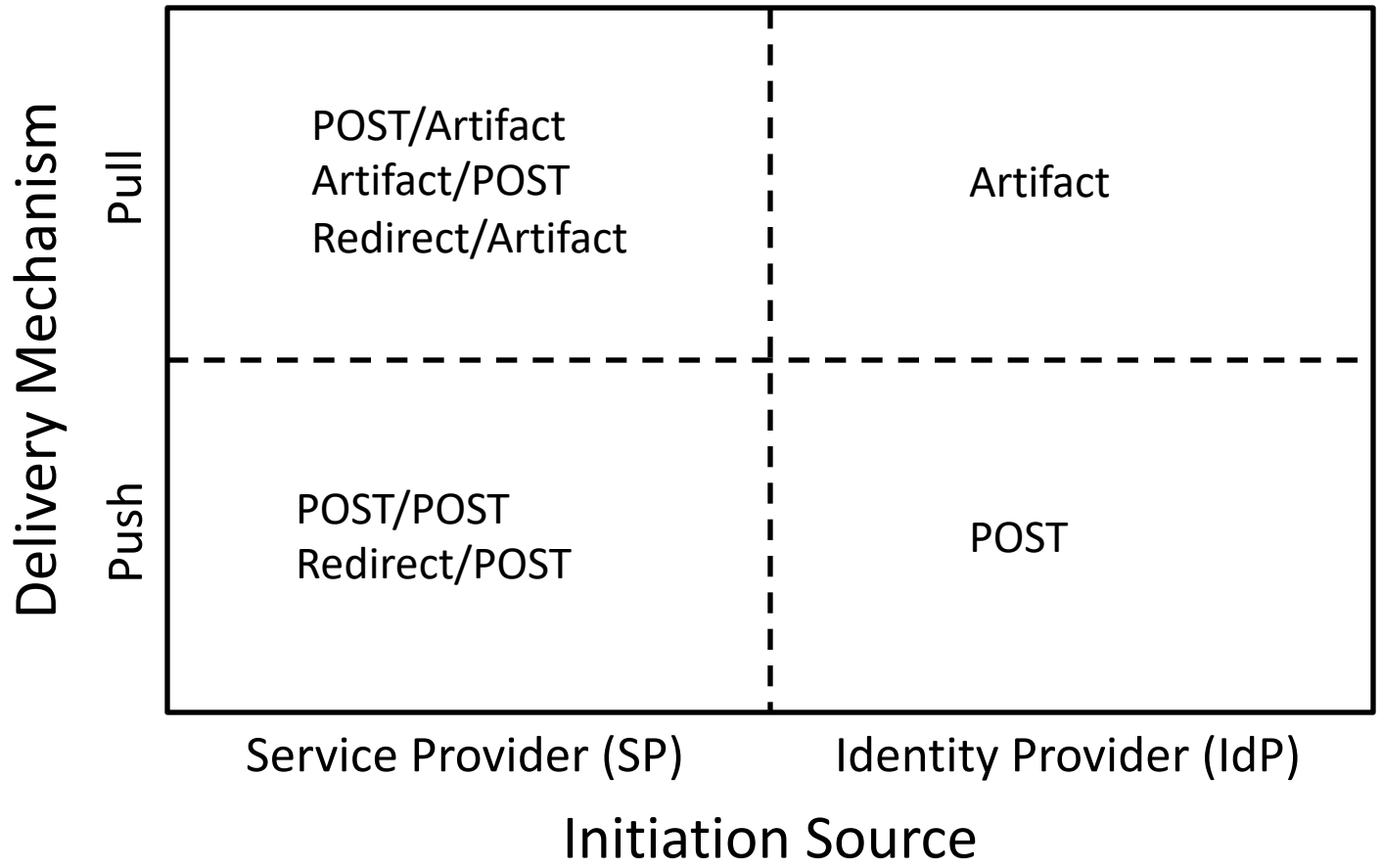
More on the SSO pull scenario

- "Access inter-site transfer URL" step:
  - User is at: `http://smithco.com`
  - Clicks on a link that looks like it will take her to `http://jonesco.com`
  - It really takes her to inter-site transfer URL: `https://source.com/intersite?dest=jonesco.com`
- "Redirect with artifact" step:
  - Reference to user's authentication assertion is generated as a SAML "artifact" (8-byte base64 string)
  - User is redirected to assertion consumer URL, with artifact and target attached: `https://jonesco.com?SAMLart=<artifact>`

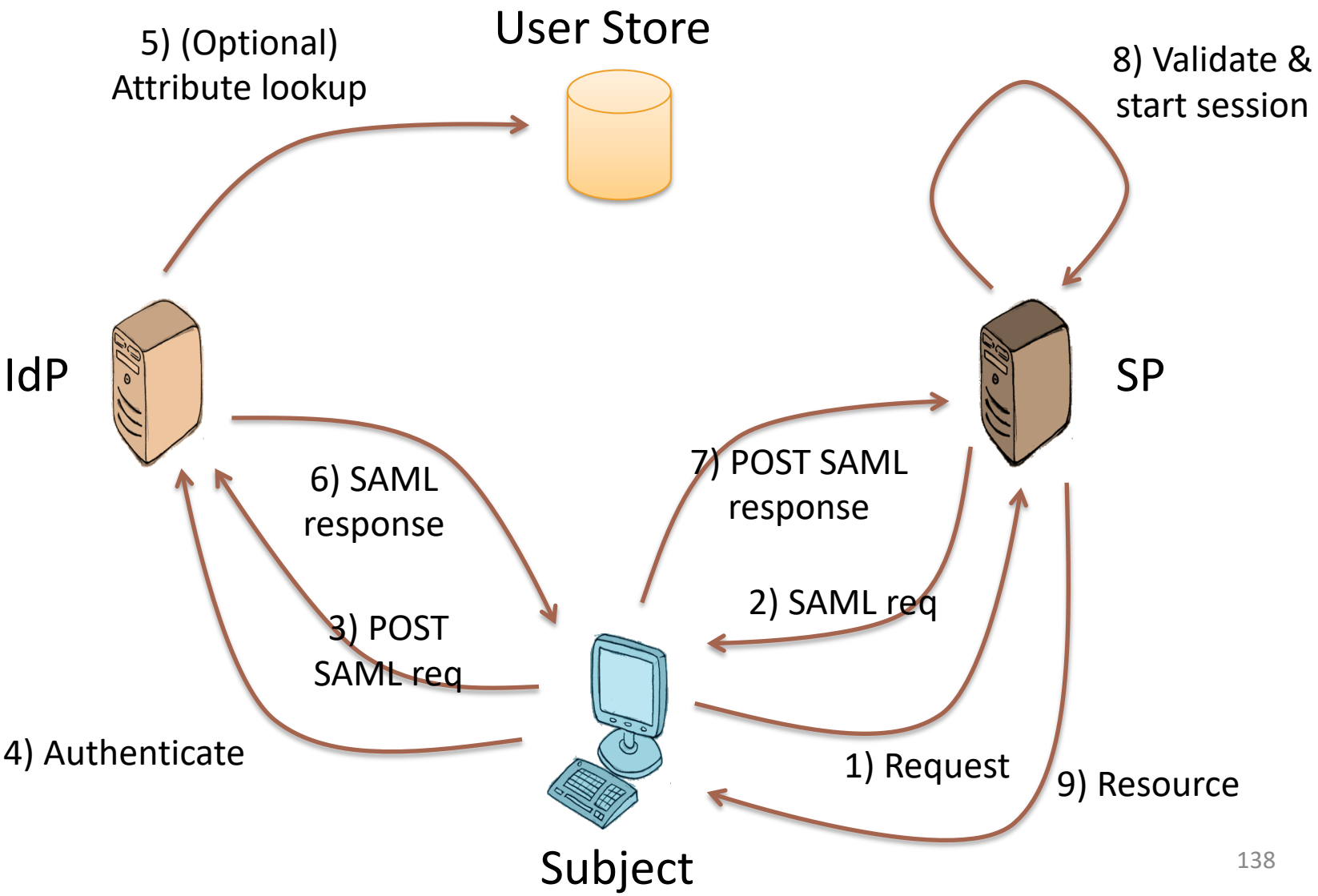
SP-Initiated ("Destination First") SSO



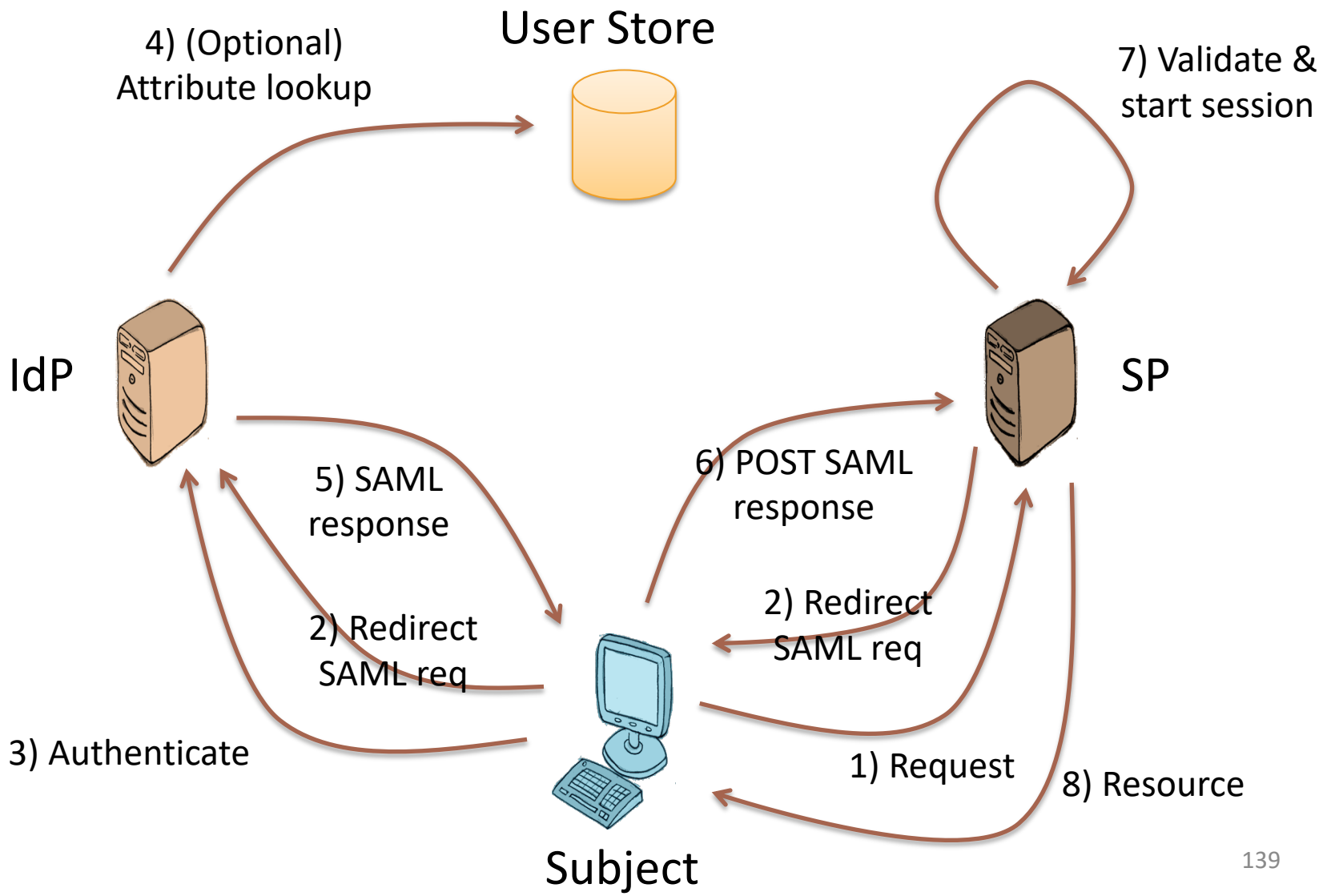
Profile Sources & Delivery Mechanisms (SAML 2.0)



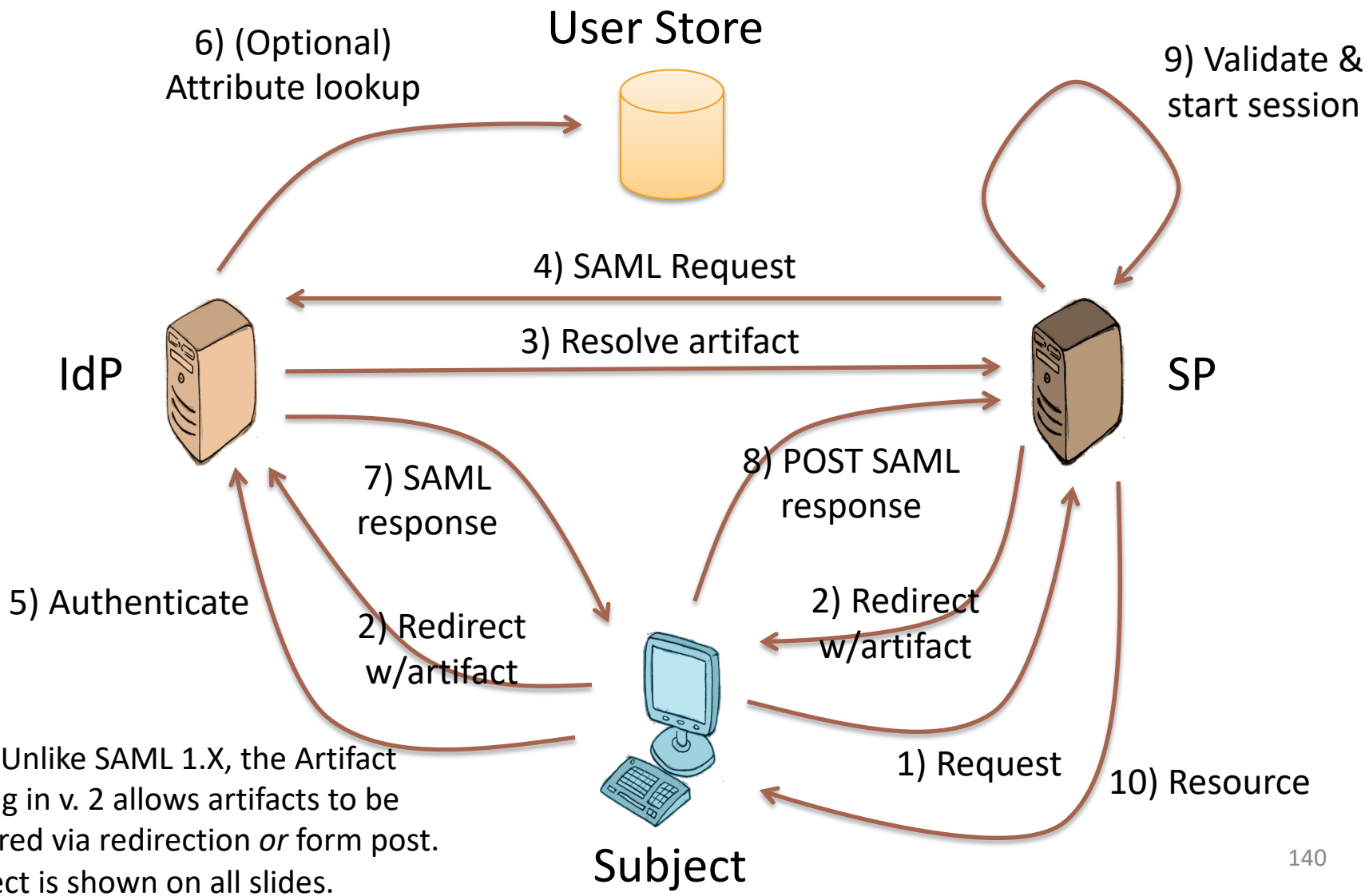
SP-Initiated SSO: POST/POST



SP-Initiated SSO: Redirect/POST

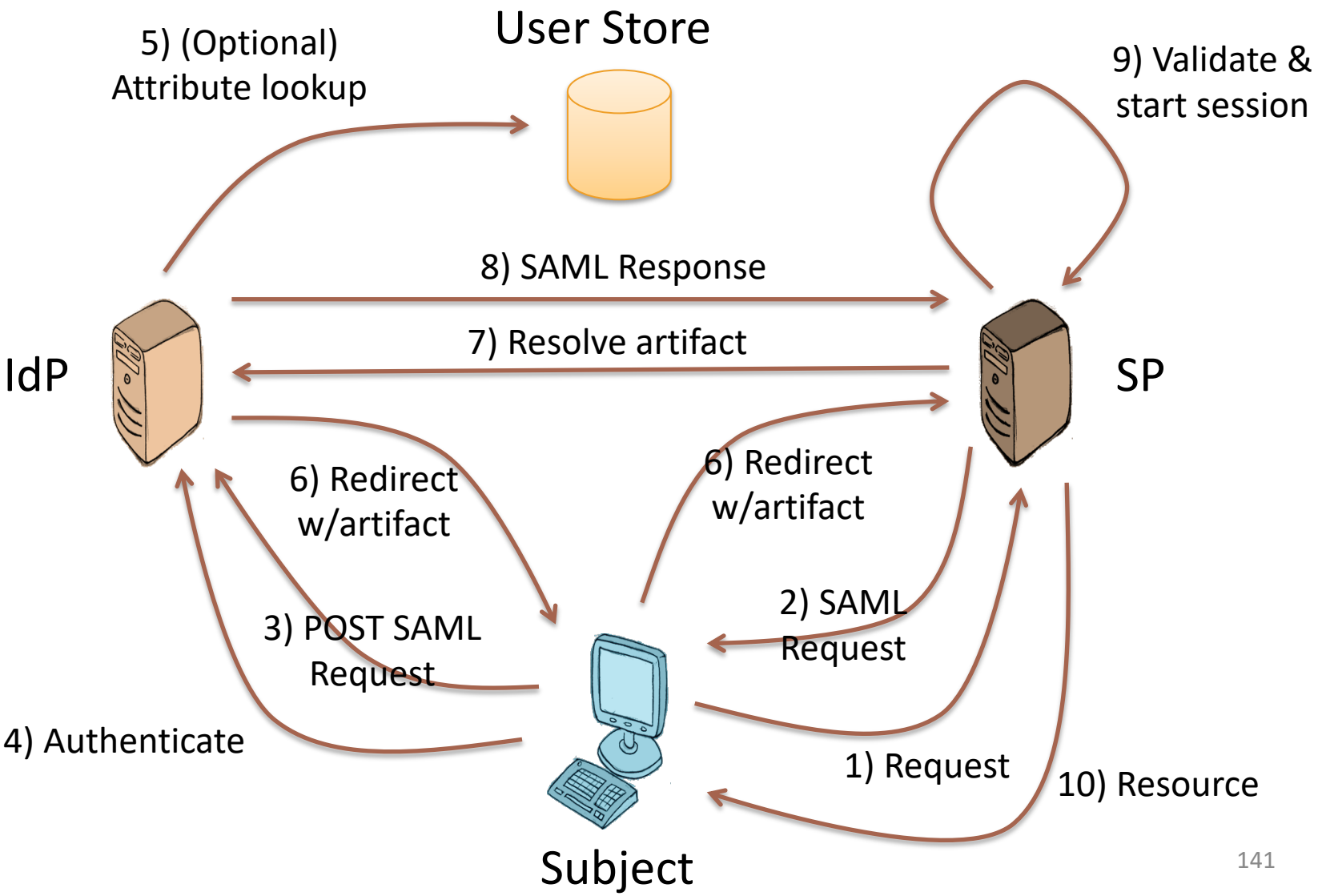


SP-Initiated SSO: Artifact/POST

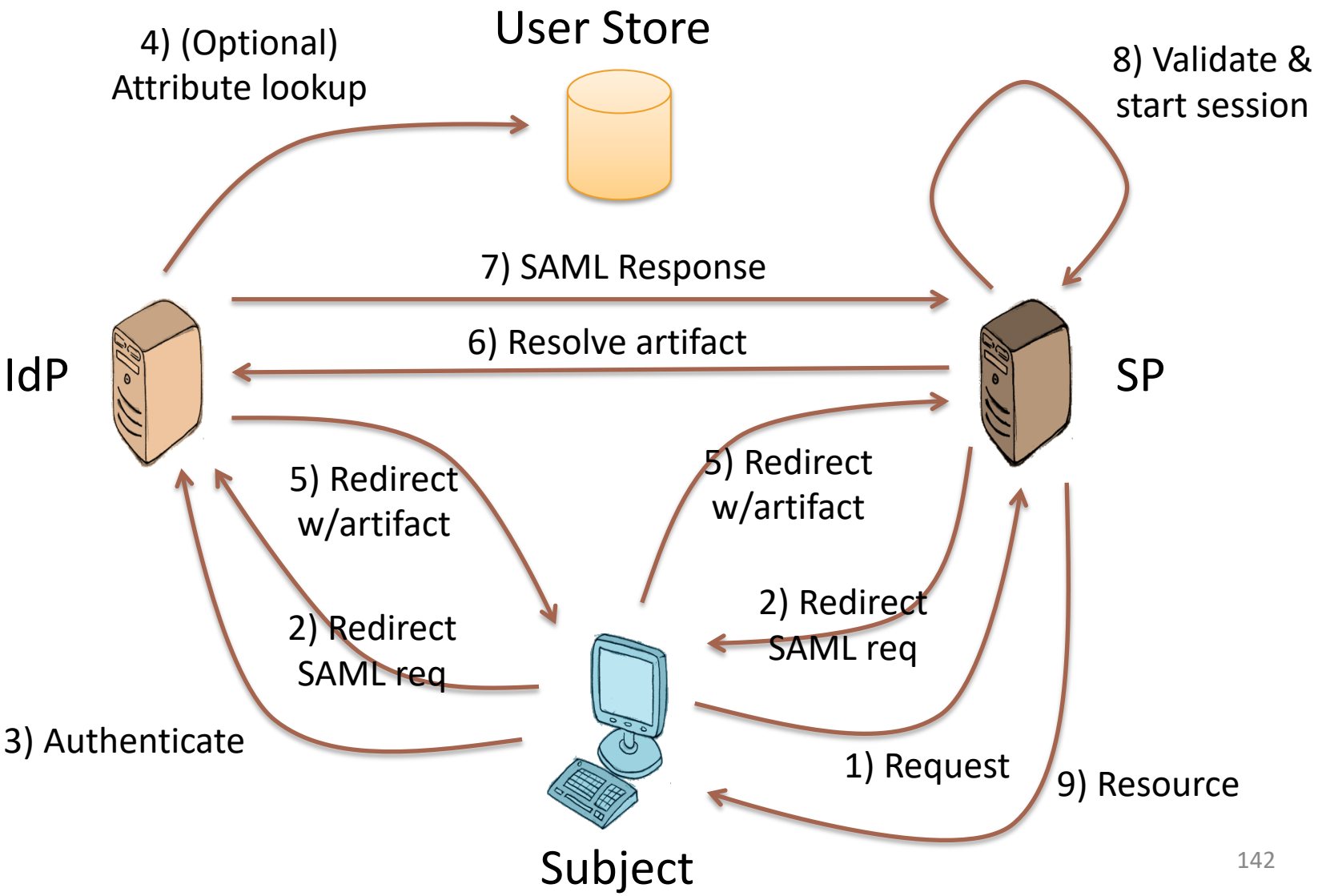


Note: Unlike SAML 1.X, the Artifact binding in v. 2 allows artifacts to be delivered via redirection *or* form post. Redirect is shown on all slides.

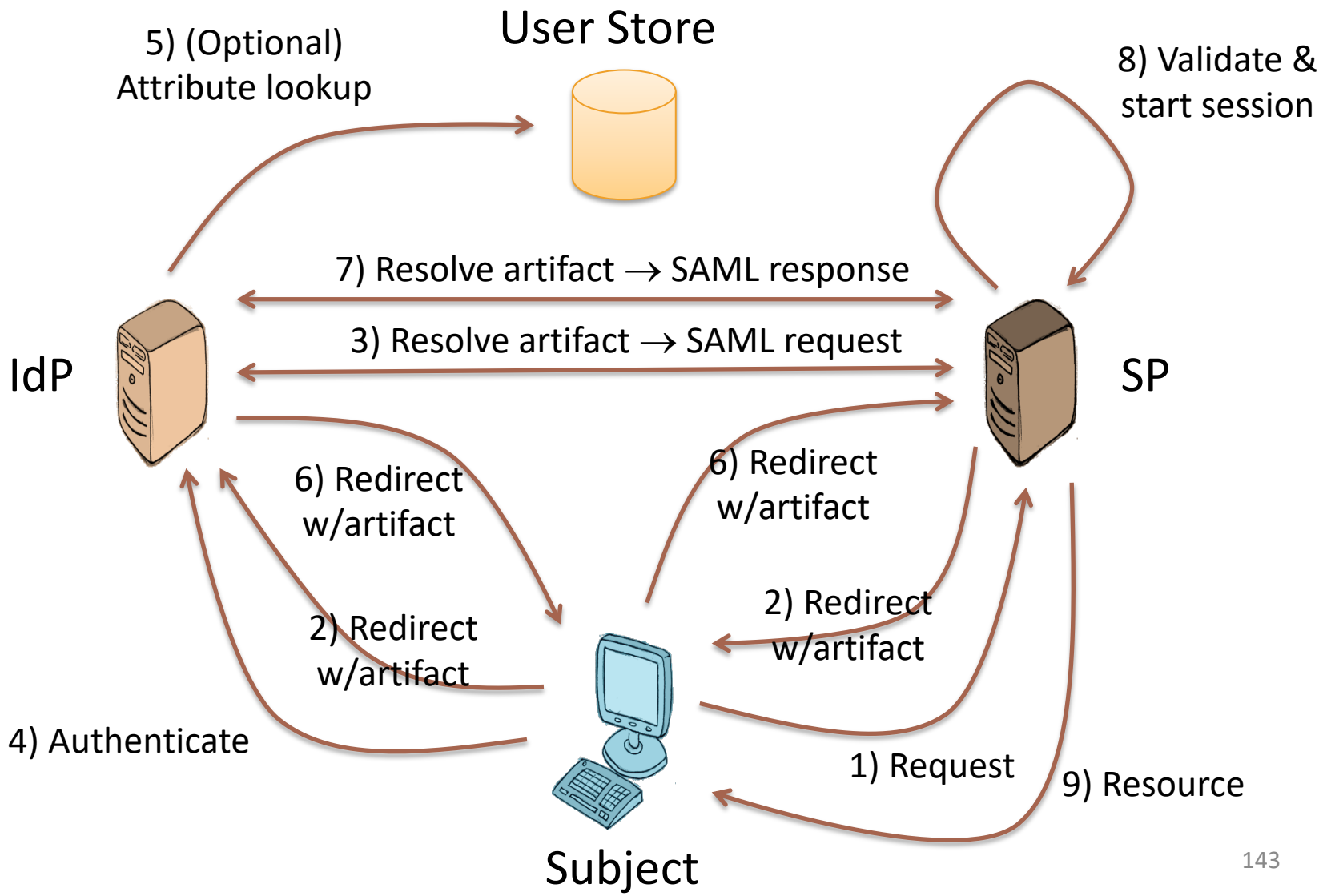
SP-Initiated SSO: POST/Artifact



SP-Initiated SSO: Redirect/Artifact



SP-Initiated SSO: Artifact/Artifact



IdP-Initiated SSO: POST

- Same as SSO: Browser/POST from SAML 1.X

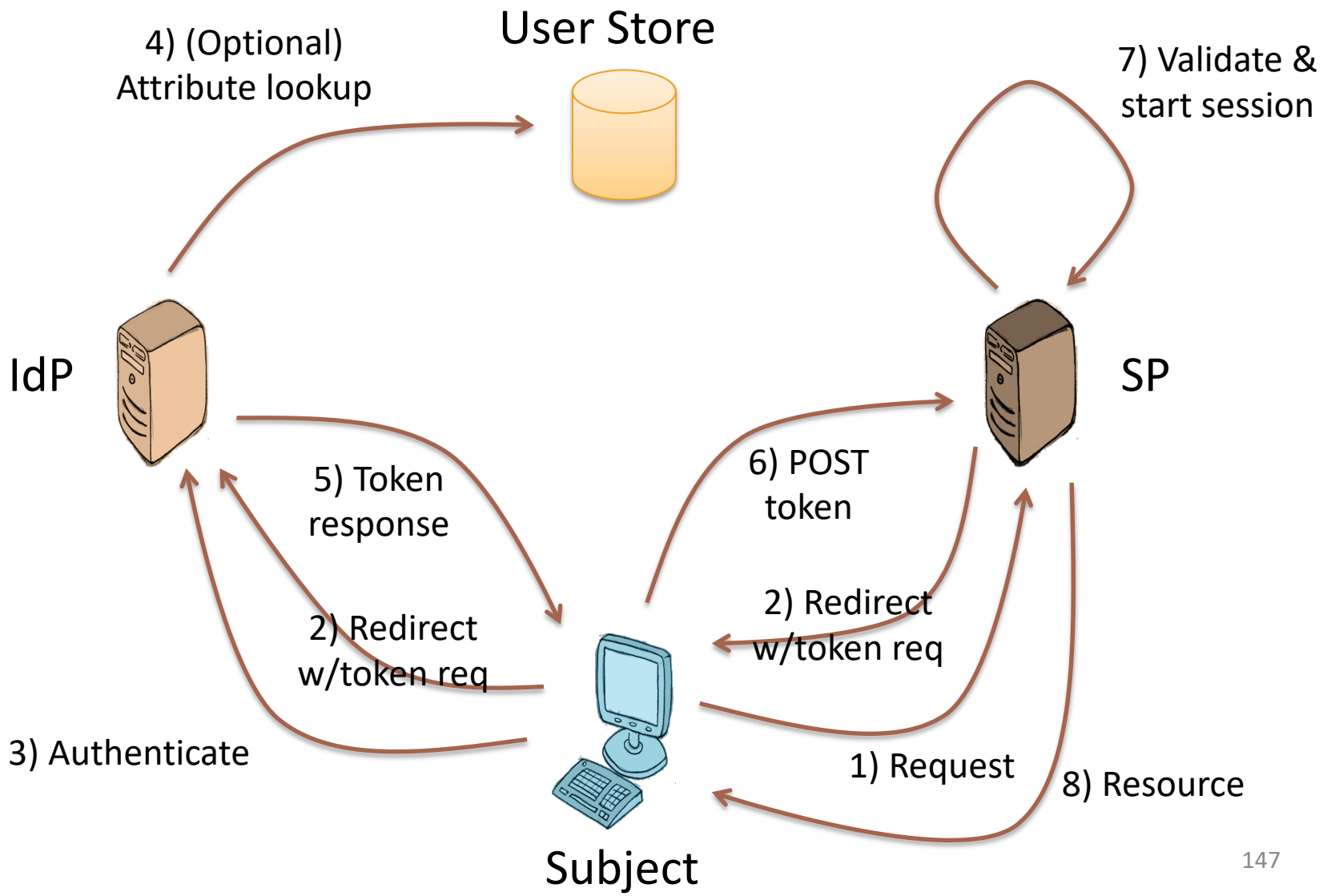
IdP-Initiated SSO: Artifact

- Same as SSO: Browser/Artifact from SAML 1.X

WS-Federation
Passive Requestor Profile

- Similar to SAML 2's SP-Initiated SSO: Redirect/POST profile
- Standard allows for others, but this is the pervasive one

SP-Initiated SSO: Redirect/POST



Choosing a Profile

- Constraints – Connectivity, technology, etc.
- Process of elimination
  1. Standard and version of standard
  2. If SAML chosen as standard
     - Initiator (SP or IdP)
     - Delivery mechanism (push or pull)

When to use IdP-Initiated Profiles

- Web site already maintains identities (effectively making it an IdP)
- Web site will be the jumping off point to 3rd party sites/services on a different domain

Choosing a Binding – Artifact

- Reluctant to expose message to intermediary
- Encryption is not practical
- P2P connection between SP and IdP
- State management req'd – mapping artifacts to messages, artifact usage tracking
- Additional administration – mapping source IDs to endpoints
- Consume/expose service that supports Artifact Resolution protocol over SOAP or other

Choosing a Binding – Redirect

- URL with SAML message < 2,000 characters
- Form submission w/ JavaScript is not viable and manual submission of form does not meet UI requirements

Choosing a Binding – Form Post

- URL with SAML message ≥ 2,000 characters
- Forms can be posted w/ JavaScript or manually

XACML: EXTENSIBLE ACCESS CONTROL MARKUP LANGUAGE

XACML Overview

- Represent access control policies in XML
- 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

General terms

- Resource
  - Data, system component or service
- Subject
  - An actor who makes a request to access certain Resources.
- Action
  - An operation on resource
- Environment
  - The set of attributes that are relevant to an authorization decision and are independent of a particular subject, resource or action
- Attributes
  - Characteristics of a subject, resource, action or environment
- Target
  - Defines conditions that determine whether policy applies to request

XACML Rule Example (Simplified)

```
<Rule RuleId="1" Effect="Permit">
<Description>Allow Joe to send a message</Description>
<Target>
  <Subjects>
    <Subject><SubjectMatch MatchID="string-equal">
      <AttributeValue>Joe</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 Rule Example (Simplified)

```
<Rule RuleId="1" Effect="Permit">
<Description>Allow Joe to send a message</Description>
<Target>
  <Subjects>
    <Subject>
      <SubjectMatch MatchID="string-equal">
        <AttributeValue>Joe</AttributeValue>
        <SubjectAttributeDesignator
          AttributeId="subject-id"/>
      </SubjectMatch>
    </Subject>
  </Subjects>
  <Resources>…</Resources>
  <Actions>…</Actions>
</Target>
</Rule>
```

XACML Rule Example (Simplified)

```
<Rule RuleId="1" Effect="Permit">
<Description>Allow Joe to send a message</Description>
<Target>
  <Subjects>…</Subjects>
  <Resources>
    <Resource>
      <ResourceMatch MatchID="anyURI-equal">
        <AttributeValue>uri:message</AttributeValue>
        <ResourceAttributeDesignator
          AttributeID="resource-id"/>
      </ResourceMatch>
    </Resource>
  </Resources>
  <Actions>…</Actions>
</Target>
</Rule>
```

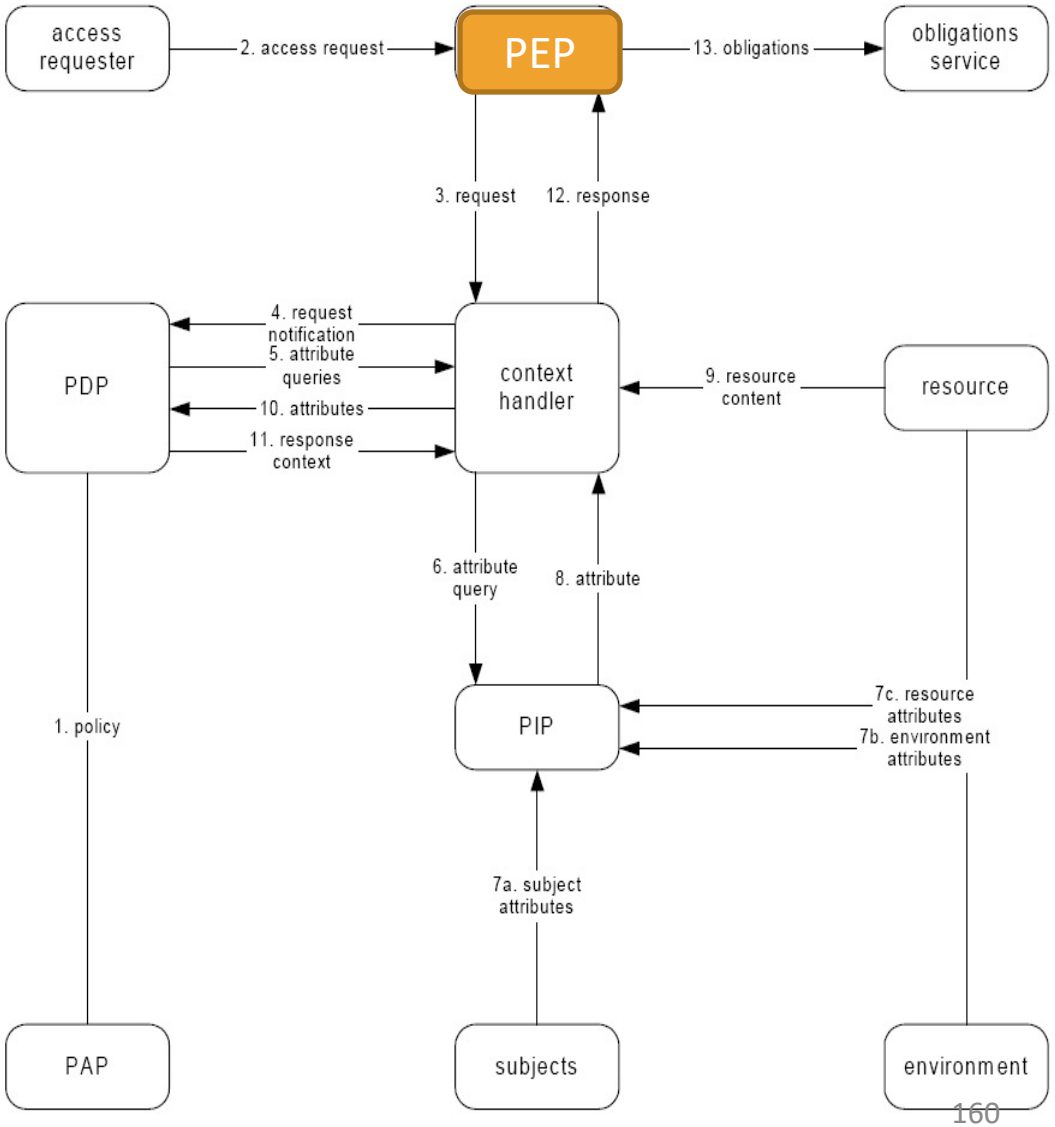
XACML Rule Example (Simplified)

```
<Rule RuleId="1" Effect="Permit">
<Description>Allow Joe to send a message</Description>
<Target>
  <Subjects>…</Subjects>
  <Resources>…</Resources>
  <Actions>
    <Action>
      <ActionMatch MatchID="string-equal">
        <AttributeValue>send</AttributeValue>
        <ActionAttributeDesignator
          AttributeID="action-id"/>
      </ActionMatch>
    </Action>
  </Actions>
</Target>
</Rule>
```

Usage Scenario

Policy Enforcement Point (PEP)

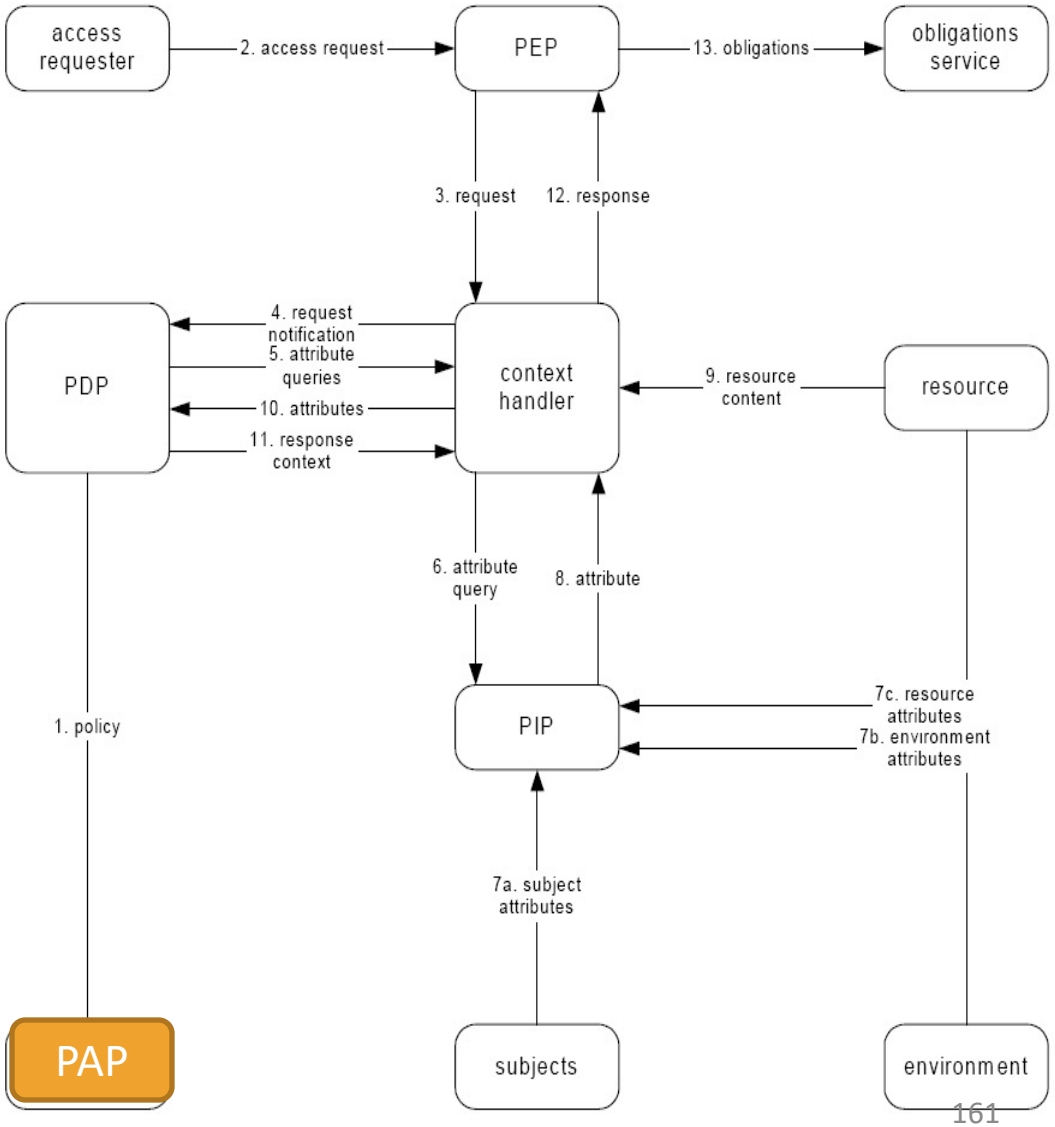
- Entity protecting the resource(e.g. file system)
- Performs access control by making decision requests and enforcing authorization decisions.



Usage Scenario

Policy Administration Point (PAP)

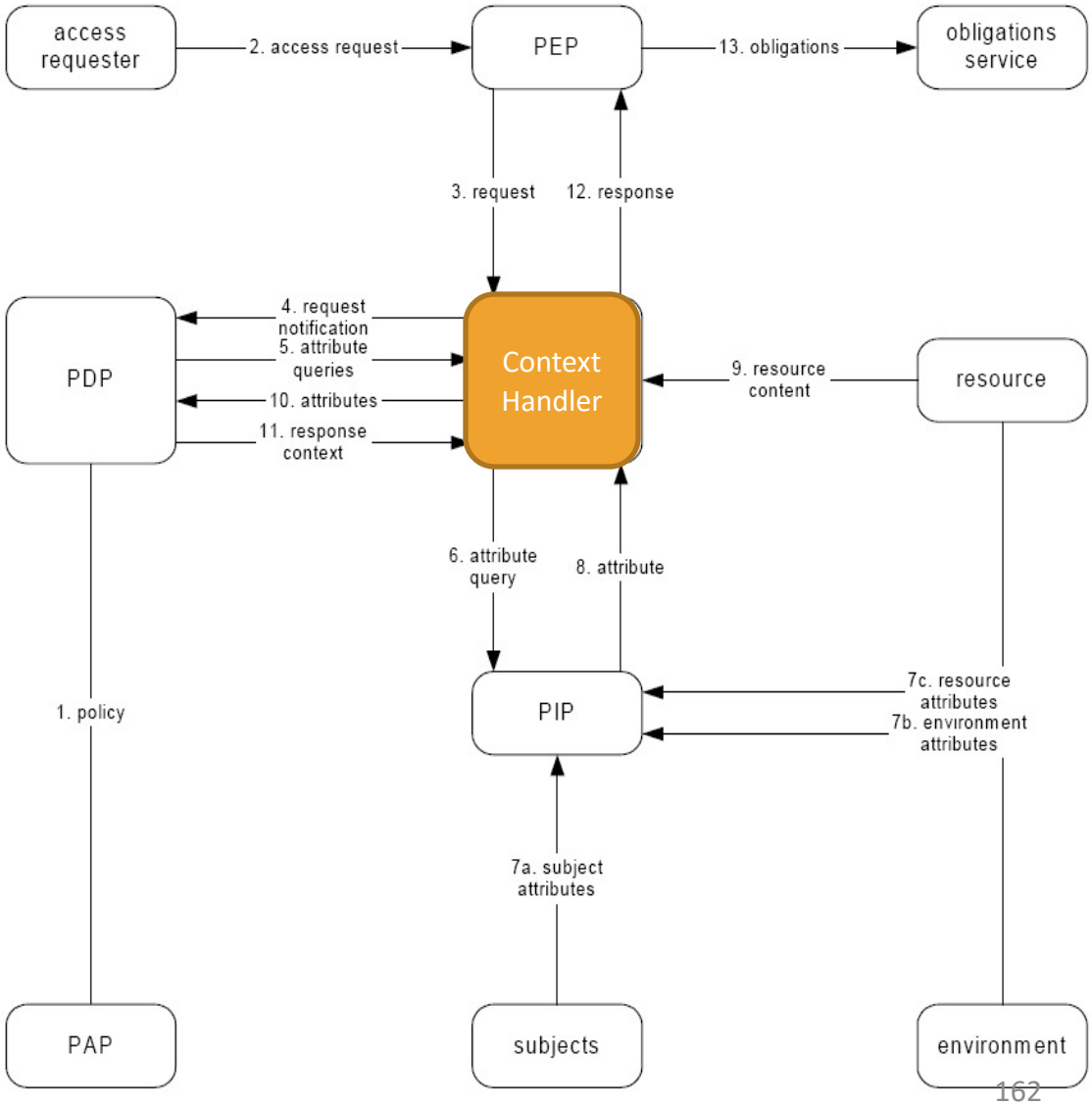
- creates security policies and stores these policies in the repository.



Usage Scenario

Context Handler

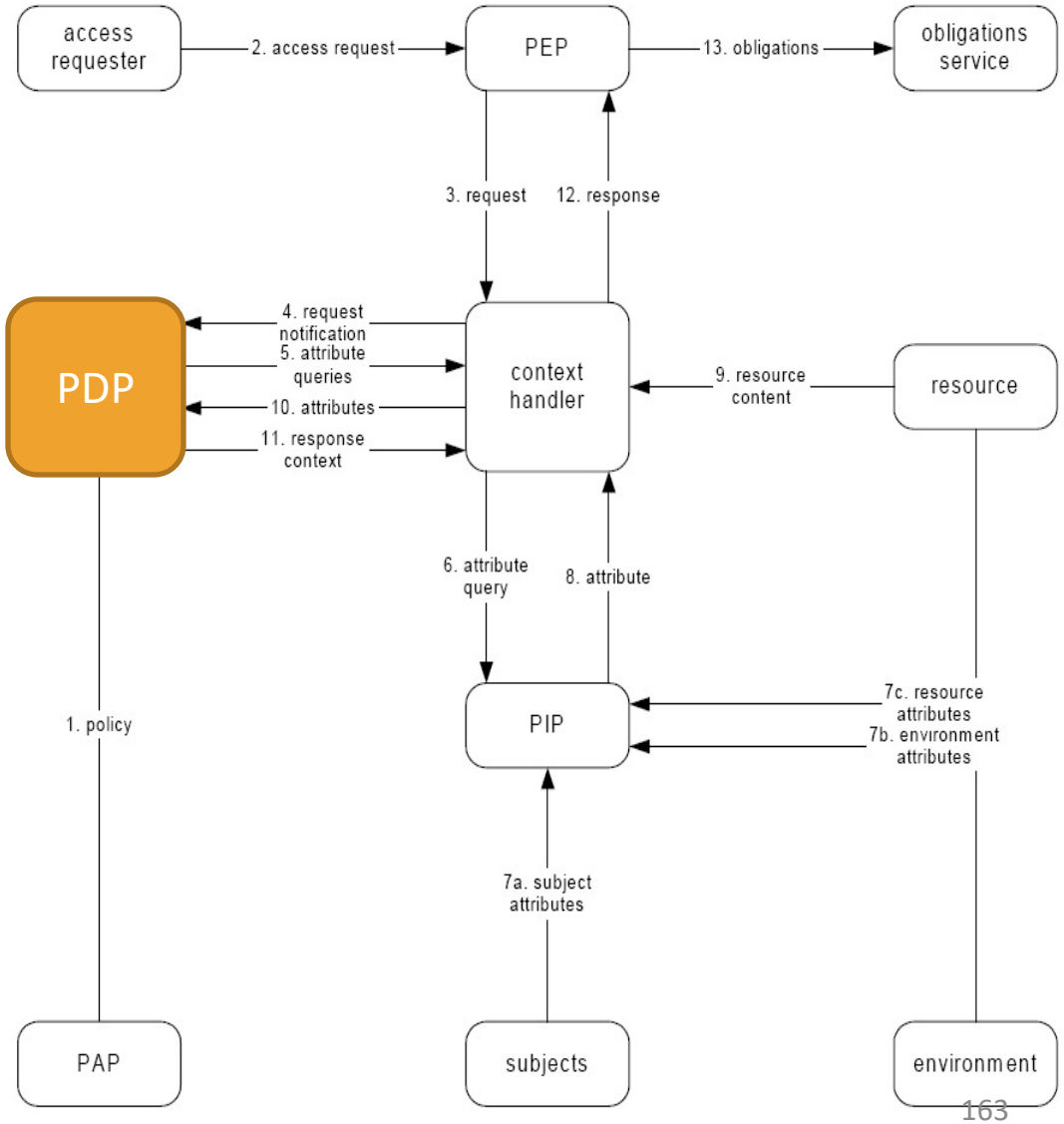
- Canonical representation of a decision request and an authorization decision.
- Can be defined to convert the requests in its native format to the XACML canonical form and to convert the auth decisions in the XACML canonical form to the native format.



Usage Scenario

The Policy Decision Point (PDP)

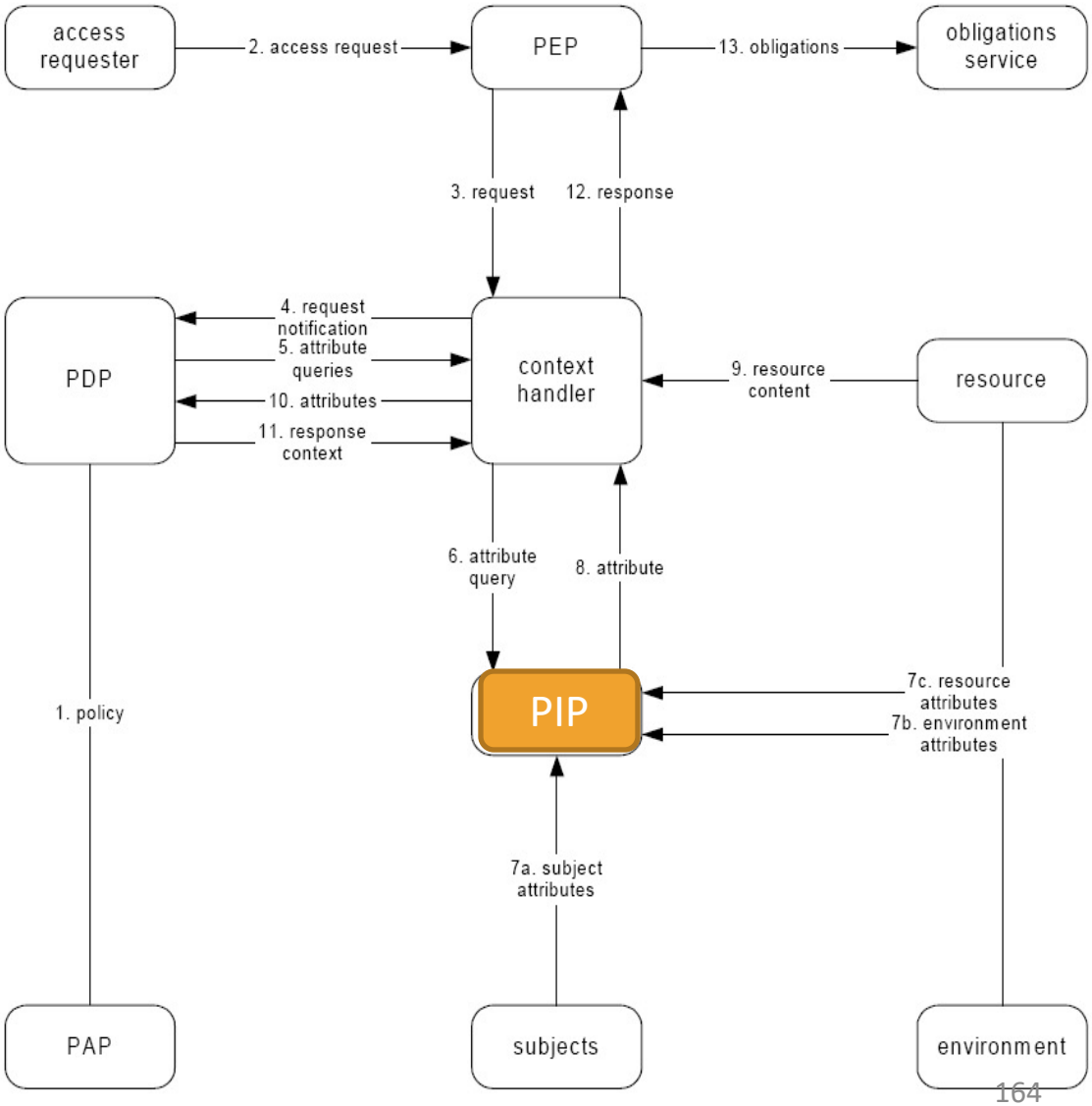
- Receives and examines the request
- Retrieves applicable policies
- evaluates the applicable policy and
- Returns the authorization decision to PEP



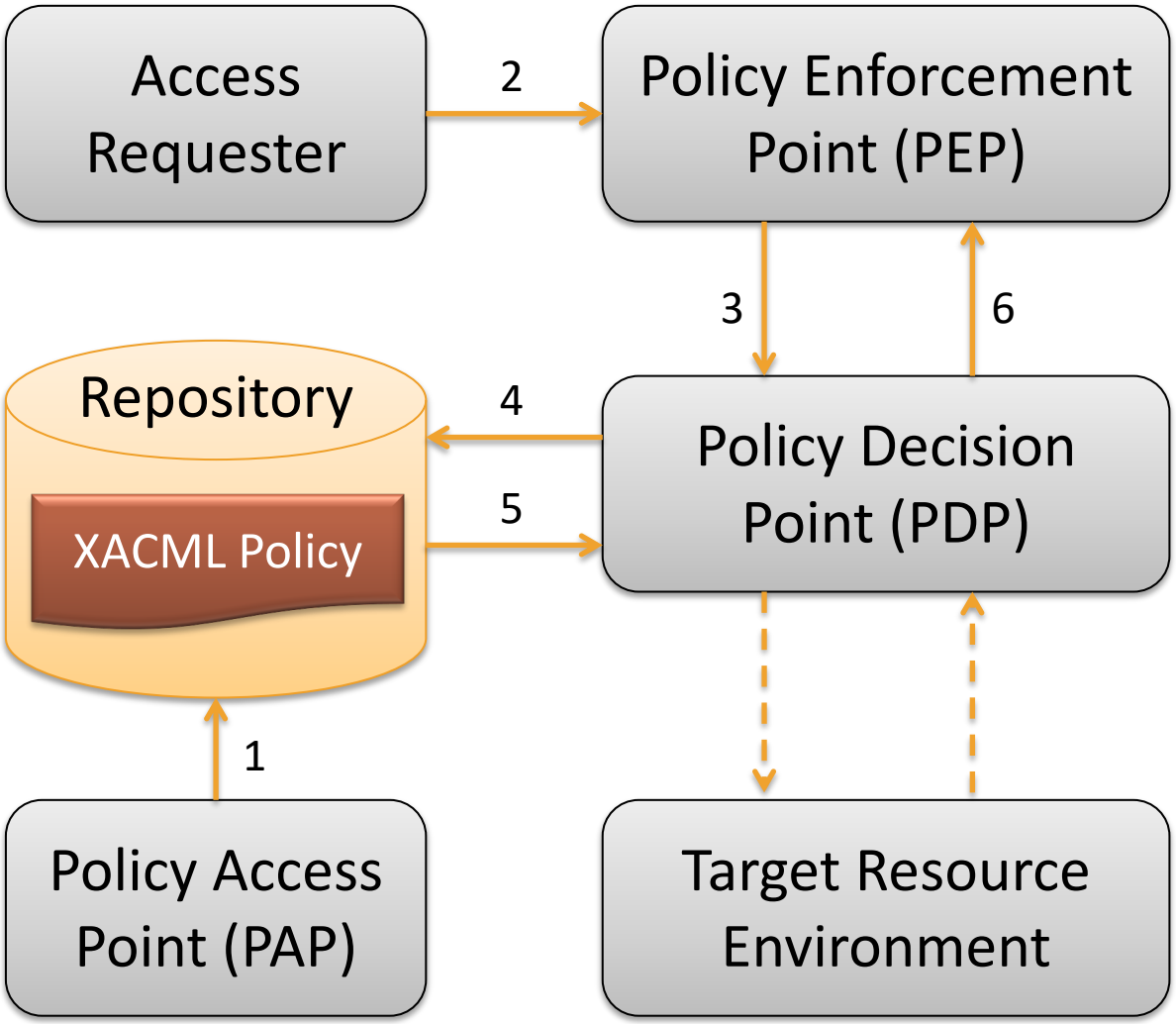
Usage Scenario

Policy Information Point (PIP)

- serves as the source of attribute values, or the data required for policy evaluation.



XACML Architecture



1: Policy Definition

2: Access Request

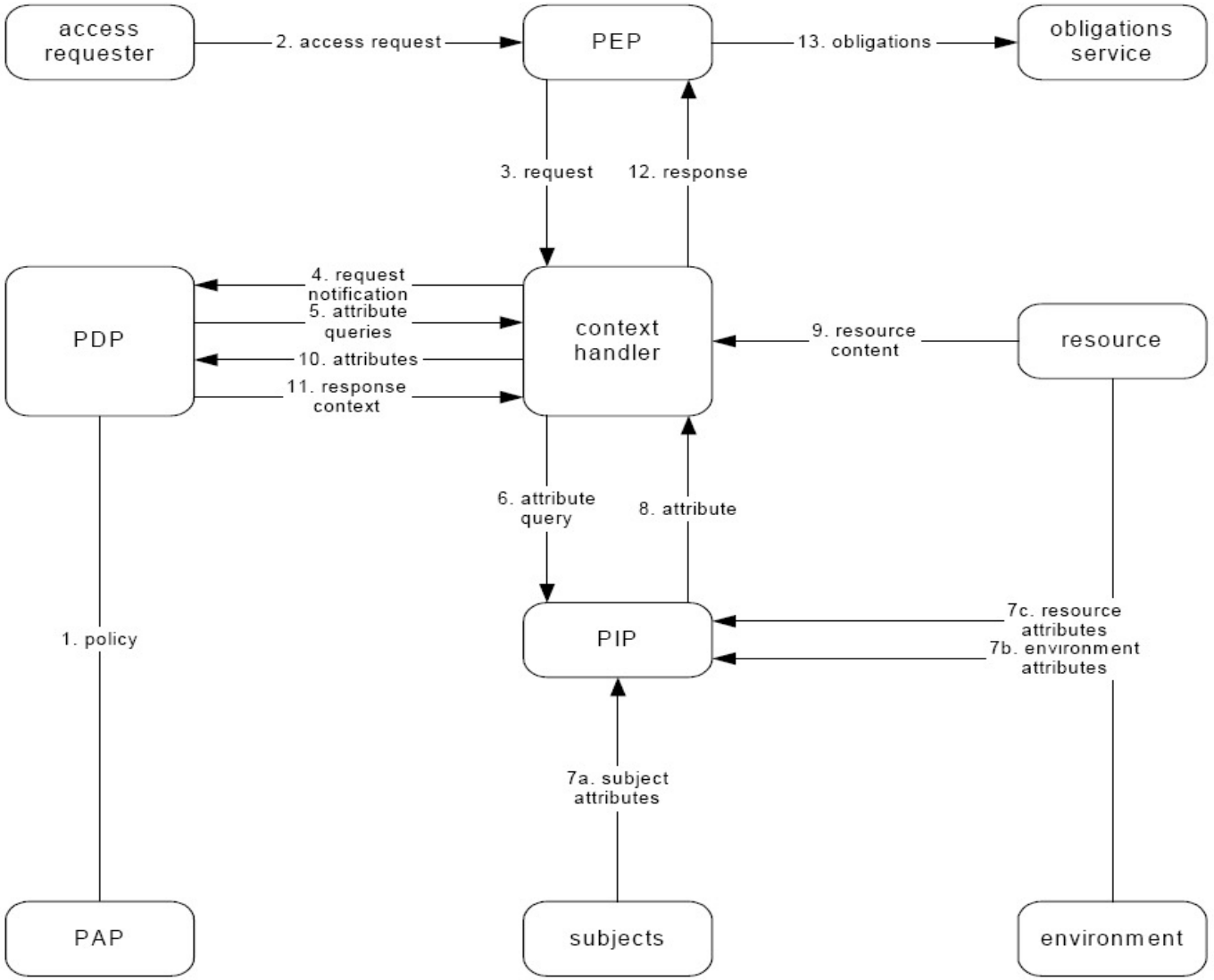
3: SAML Request

4: Policy Lookup

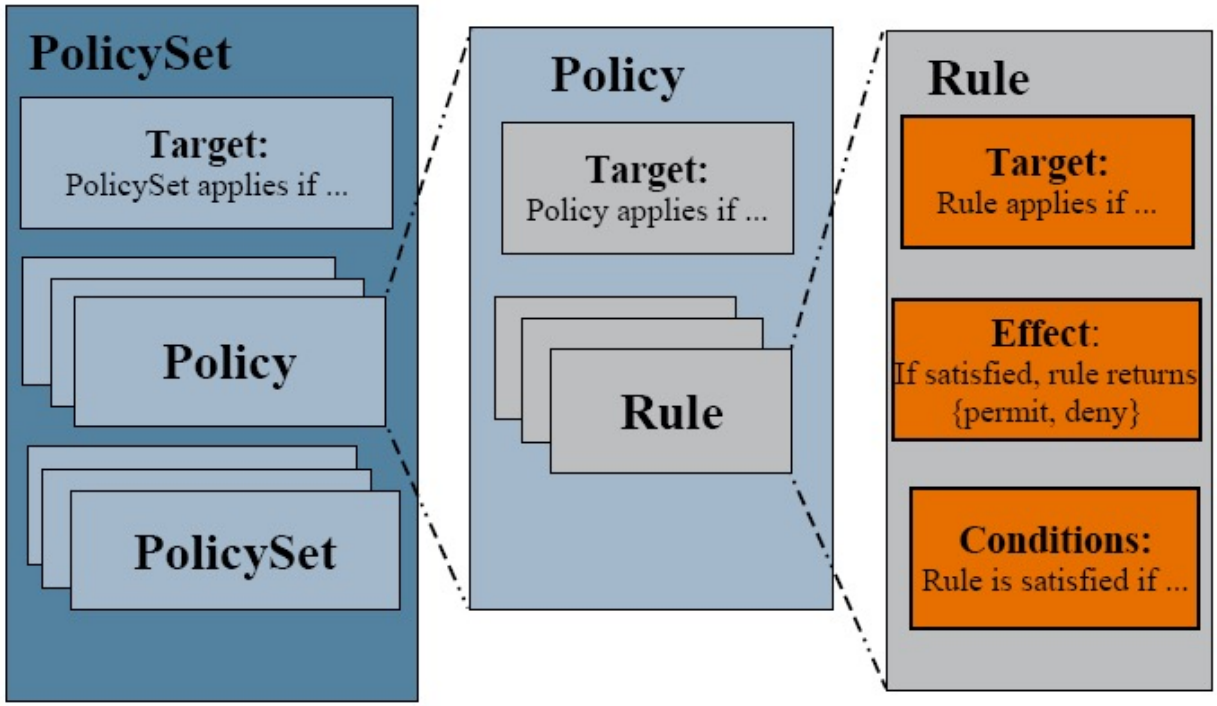
5: Policy

6: SAML Response

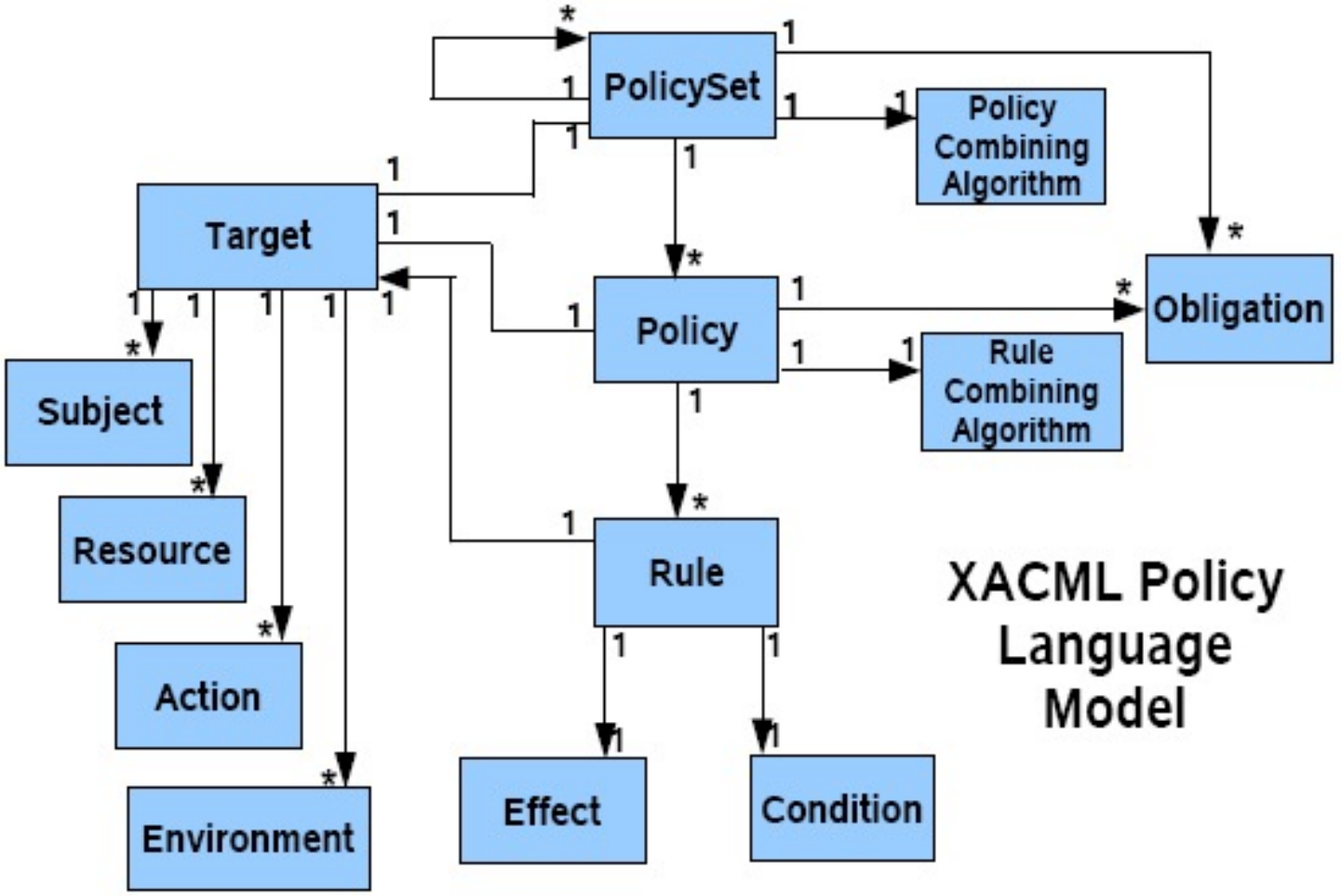
Data Flow


access requester
2. access request
PEP
13. obligations
obligations service
3. request
12. response
PDP
4. request notification
5. attribute queries
10. attributes
11. response context
context handler
9. resource content
resource
6. attribute query
8. attribute
1. policy
PIP
7c. resource attributes
7b. environment attributes
7a. subject attributes
PAP
subjects
environment

XACML Policy Structure



Policy Language model

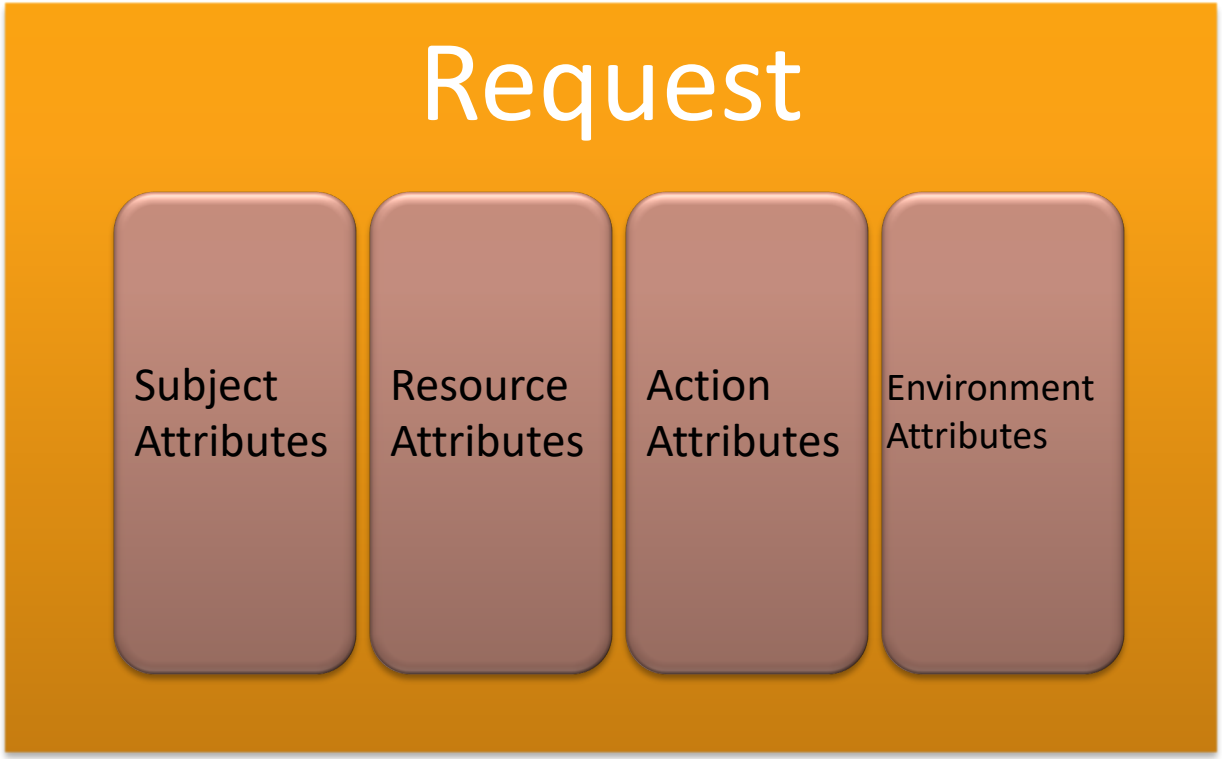


XACML Policy Example

```
<Policy PolicyId="ExamplePolicy"
   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-
   overrides">
  <Target>
    <Subjects><AnySubject/></Subjects>
    <Resources><Resource>
      <ResourceMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
          http://server.example.com/code/docs/developer-guide.html
        </AttributeValue>
        <ResourceAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#anyURI"
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
      </ResourceMatch>
    </Resource></Resources>
    <Actions><AnyAction/></Actions>
  </Target>
  <Rule RuleId="ReadRule" Effect="Permit”> … </Rule>
</Policy>
```

```
<Rule RuleId="ReadRule" Effect="Permit">
  <Target>
    <Subjects><AnySubject/></Subjects>
    <Resources><AnyResource/></Resources>
    <Actions> <Action>
      <ActionMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          read</AttributeValue>
        <ActionAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
      </ActionMatch>
    </Action> </Actions>
  </Target>
  <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
    <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"

      AttributeId="group"/>
    </Apply>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
      developers</AttributeValue>
  </Condition>
```

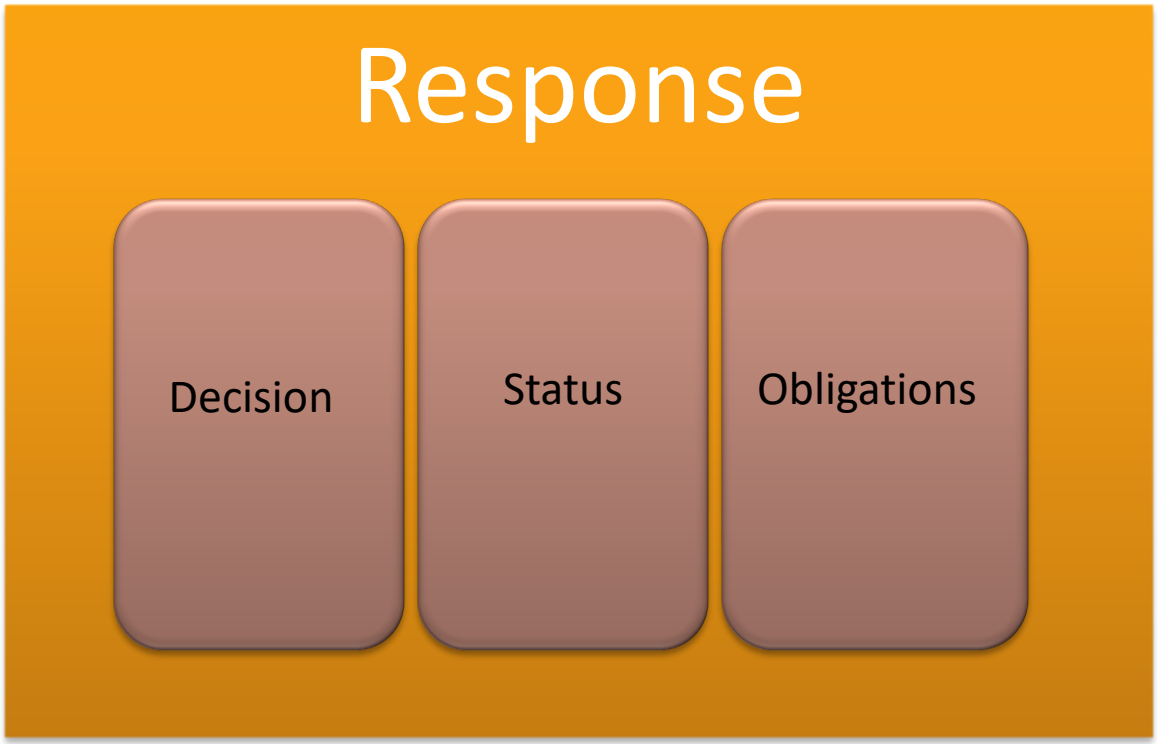
XACML Request Structure



Request Example

```
<Request>
<Subject>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id”
         DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
    <AttributeValue>xyz@users.example.com</AttributeValue>
  </Attribute>
  <Attribute AttributeId="group"  DataType=http://www.w3.org/2001/XMLSchema#string
         Issuer="admin@users.example.com">
    <AttributeValue>developers</AttributeValue>
  </Attribute>
</Subject>
<Resource>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
         DataType="http://www.w3.org/2001/XMLSchema#anyURI">
    <AttributeValue>http://server.example.com/code/docs/developer-guide.html
    </AttributeValue>
  </Attribute>
</Resource>
<Action>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
         DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>read</AttributeValue>
  </Attribute>
</Action>
</Request>
```

XACML Response Structure



XACML Response Example

```
<Response>
  <Result>
    <Decision>Permit</Decision>
    <Status>
      <StatusCode
         Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
```

Effect:

Permit/Deny/Not Applicable/Indeterminate

Combining Algorithms

- Deny-overrides
  - If ***all*** rules evaluate to Permit, then the result is Permit.
  - if any evaluation returns Deny, then the result must be Deny.
- Permit-overrides
  - if ***any*** rule evaluates to Permit, then the result is Permit.
  - If any rule evaluates to Deny and all other rules evaluate to NotApplicable, then the result is Deny.
  - If all rules are found to be NotApplicable, then the result is NotApplicable.

Combining Algorithms

- First applicable – rules evaluated in their listing order
  - For each rule, if the target matches and the condition evaluates to True, then the result of that rule will be the evaluation of the policy (Permit, Deny, or Indeterminate).
  - Otherwise, the algorithm goes to the next rule.
  - If no rule applies, then the result is NotApplicable.
- Only-one-applicable –
  - If no policy applies, then the result is NotApplicable.
  - If more than one policy applies, then the result is Indeterminate.
  - If only one policy applies, then the result is the result of evaluating that policy.

Extensibility

- The following XML attributes may be extended by the creation of new URIs:
  - `AttributeId, DataType, FunctionId, MatchId, ObligationId, PolicyCombiningAlgId, RuleCombiningAlgId, StatusCode, SubjectCategory`.
- XACML users MAY define new attribute identifiers
  - Leaf sub-element of a structured data-type
- XACML users MAY define a new comparison function
  - Compare a value structured data-type against some other value.

Privacy profile

- Attributes:
  - “urn:oasis:names:tc:xacml:2.0:resource:purpose”
    - the purpose for which the data resource was collected
  - “urn:oasis:names:tc:xacml:2.0:action:purpose”
    - the purpose for which access to the data resource is requested
- Matching purpose rule
  - Deny-Overrides
    - Access SHALL be denied unless the purpose for which access is requested matches, by regular-expression match, the purpose for which the data resource was collected.

RBAC profile

- Scope
  - If a subject has roles R1 , R2, ... Rn enabled, can subject X access a given resource using a given action?
  - Is subject X allowed to have role Ri enabled?
  - If a subject has roles R1 , R2, ... Rn enabled, does that mean the subject will have permissions associated with a given role R'? That is, is role R' either equal to or junior to any of roles R1 , R2, …Rn?

RBAC Profile Policies

- Role <PolicySet>,
  - Each Role <PolicySet> references a single corresponding Permission <PolicySet>
- Permission <PolicySet>,
  - actual permissions associated with a given role,
  - references to Permission <PolicySet>s associated with other roles that are junior to the given role
- Role Assignment <Policy> or <PolicySet>
  - which roles can be enabled or assigned to which subjects
- HasPrivilegesOfRole<Policy>
  - a <Policy> in a Permission <PolicySet> that supports requests
  - asking whether a subject has the privileges associated with a given role.

Limitations

- XACML is verbose and complex in some ways.
- Interactions involving PAP, PIP, etc., are not standardized.
- Policy administration, policy versioning, etc., are not standardized.

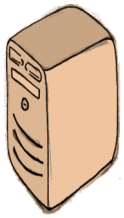
OAUTH

Scenario

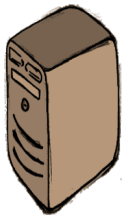
- Suppose I want to use a print service to print photos
  - Print service must access photos in photo service
  - Provide print service with owner credentials?
- OAuth approach: grant *access token* to printer
  - Resource owner (me)
  - Resource client (print service)
  - Resource server (photo service)
  - Authorization server (may be photo service)

Server Roles

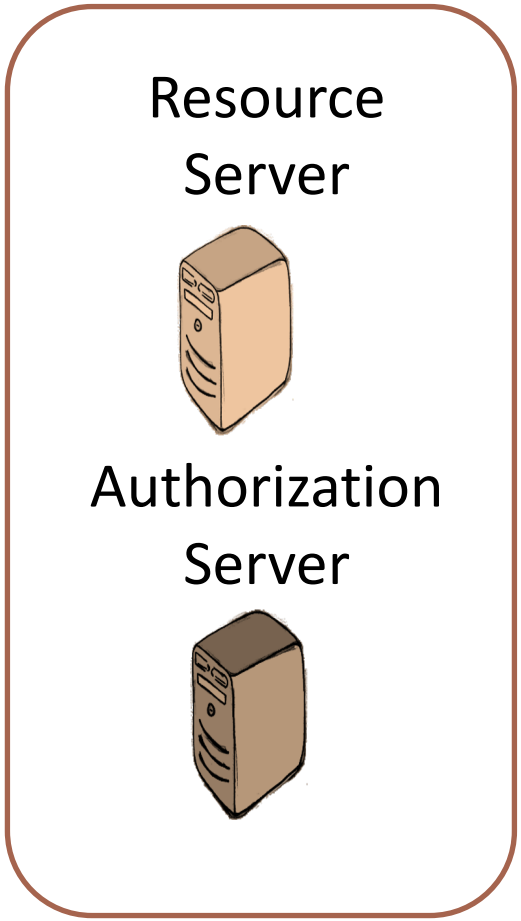
Resource Server



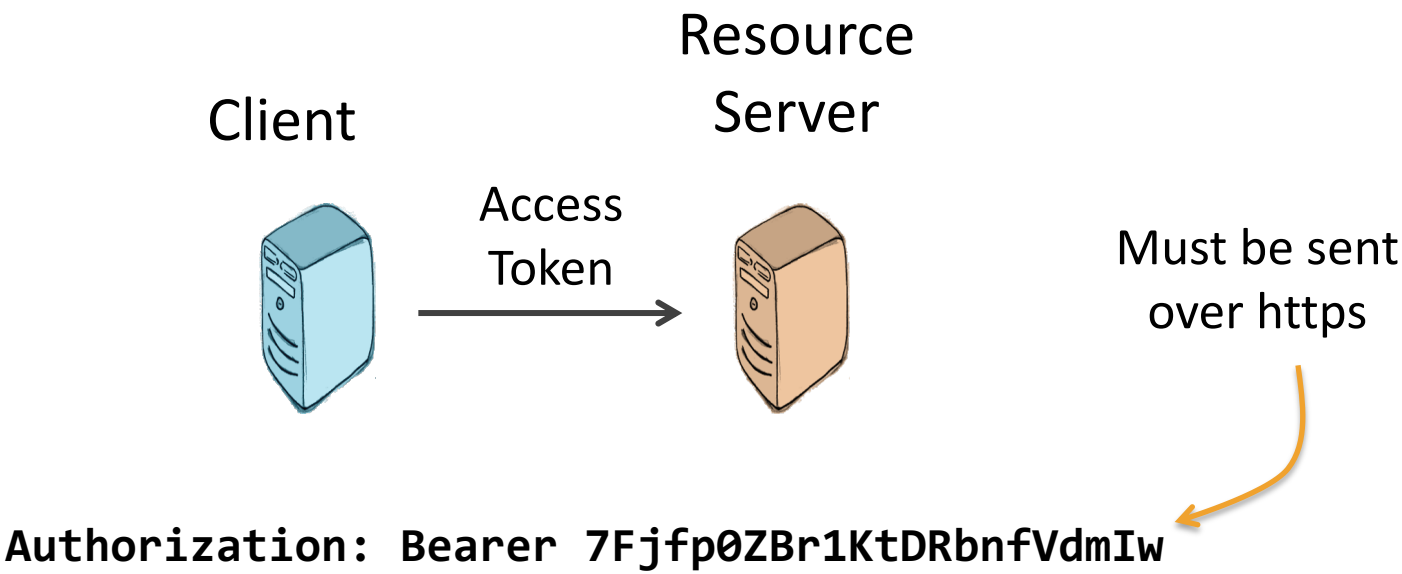
Authorization Server



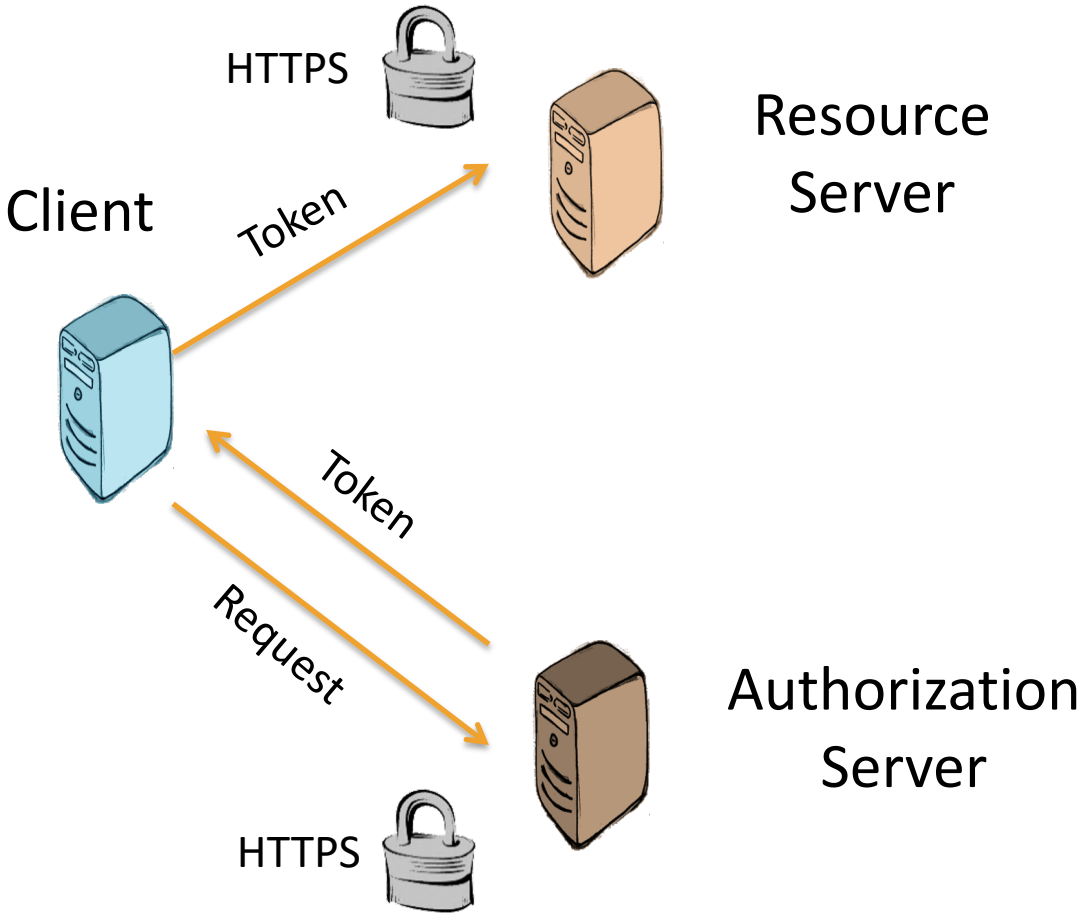
Single Server Implementation



Access Tokens



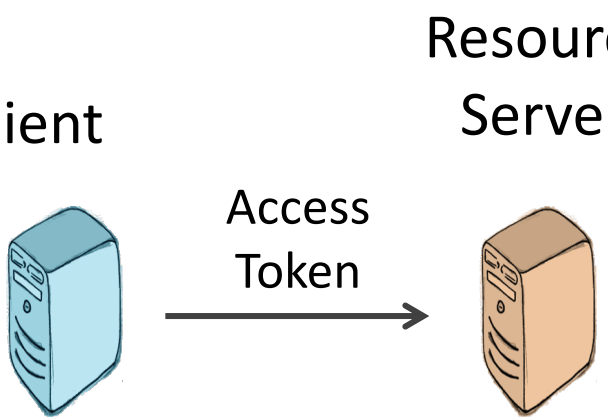
“Bearer Token”



Access Tokens

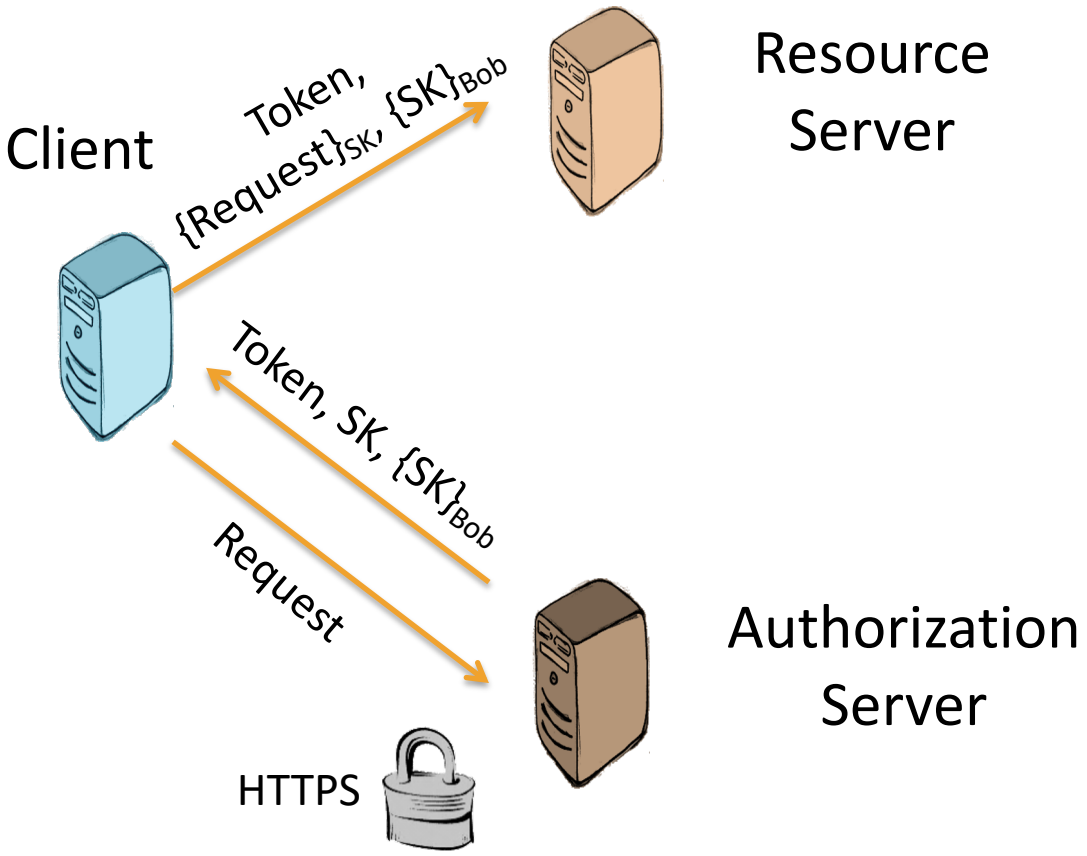
Super-simple cryptography with special support for MAC session cookies

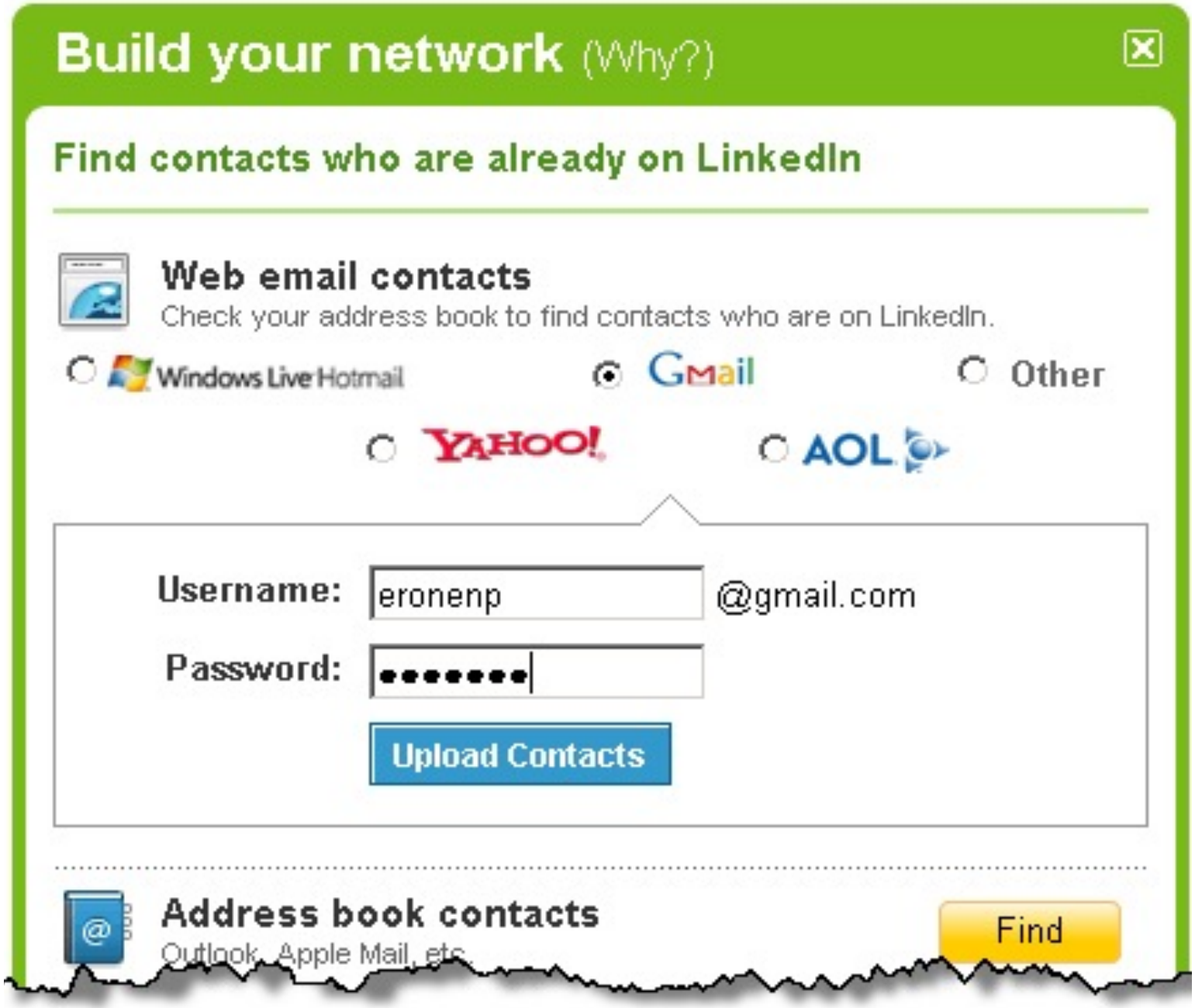
```
var normalized = nonce + '\n' +
      method.toUpperCase() + '\n' +
      httpResource + '\n' +
      host.toLowerCase() + '\n' +
      port + '\n' +
      (bodyhash || '') + '\n' +
      (ext || '') + '\n';
```



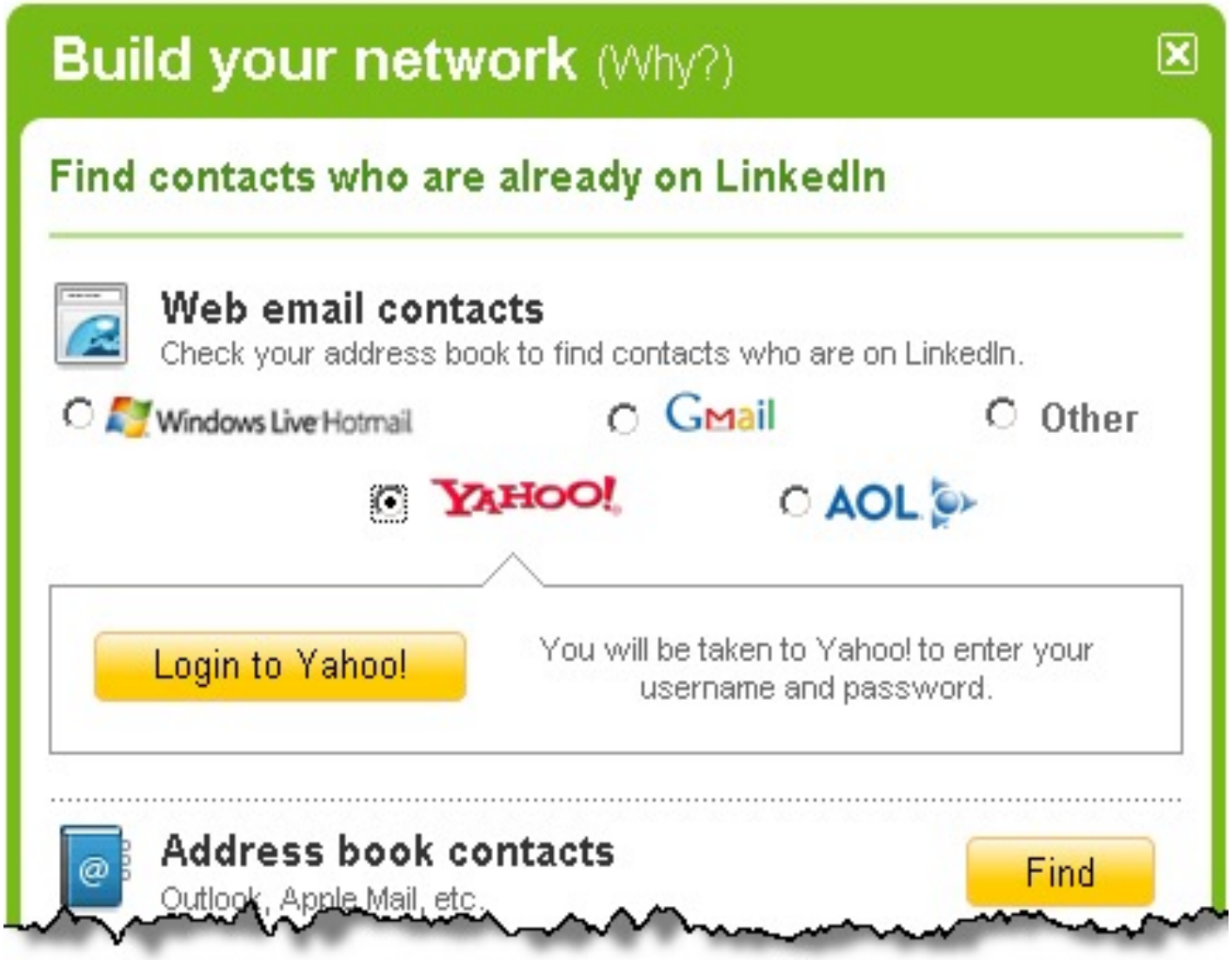

```
Authorization: Bearer 7Fjfp0ZBr1KtDRbnfVdmIw

Authorization: MAC id="h480djs93hd8",
                   nonce="274312:dj83hs9s",
                   mac="kDZvddkndxvhGRXZhvuDjEWhGeE="
```

“Message Signing”

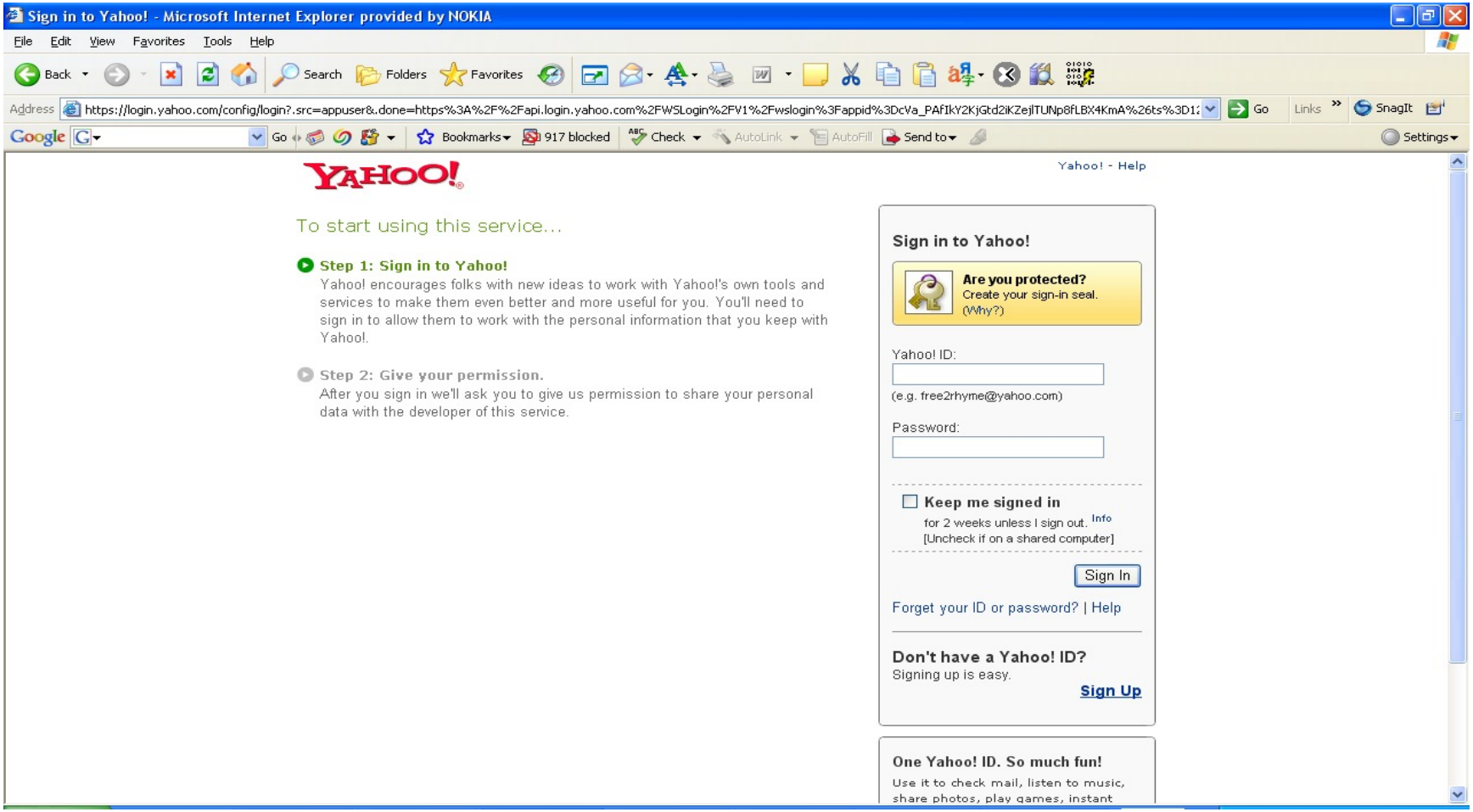



Build your network (Why?)
Find contacts who are already on LinkedIn
Web email contacts
Check your address book to find contacts who are on LinkedIn.
Windows Live Hotmail
Gmail
Other
YAHOO!
AOL
Username:
eronenp
@gmail.com
Password:
Upload Contacts
Address book contacts
Outlook, Apple Mail, etc.
Find

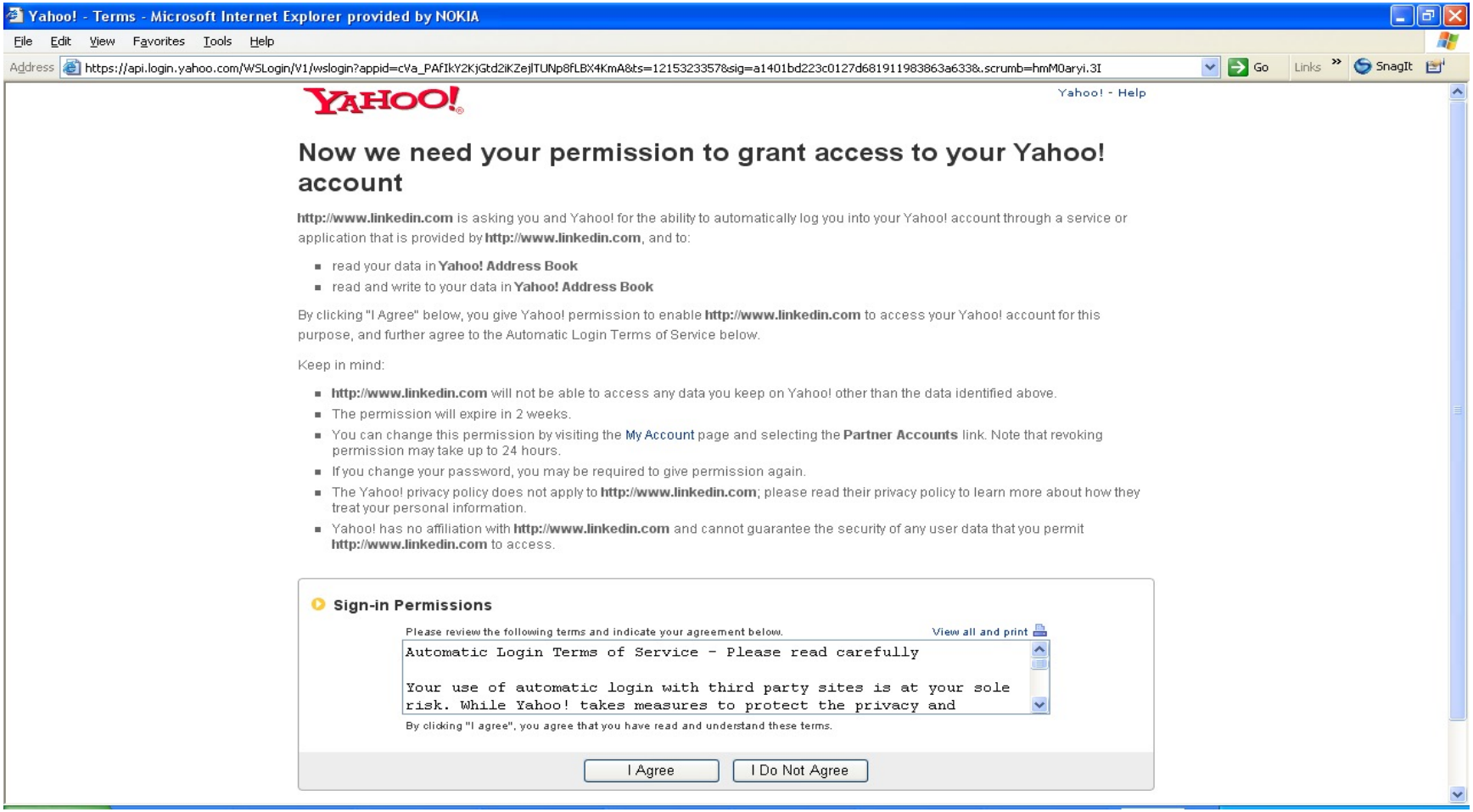
User navigates to Resource Client



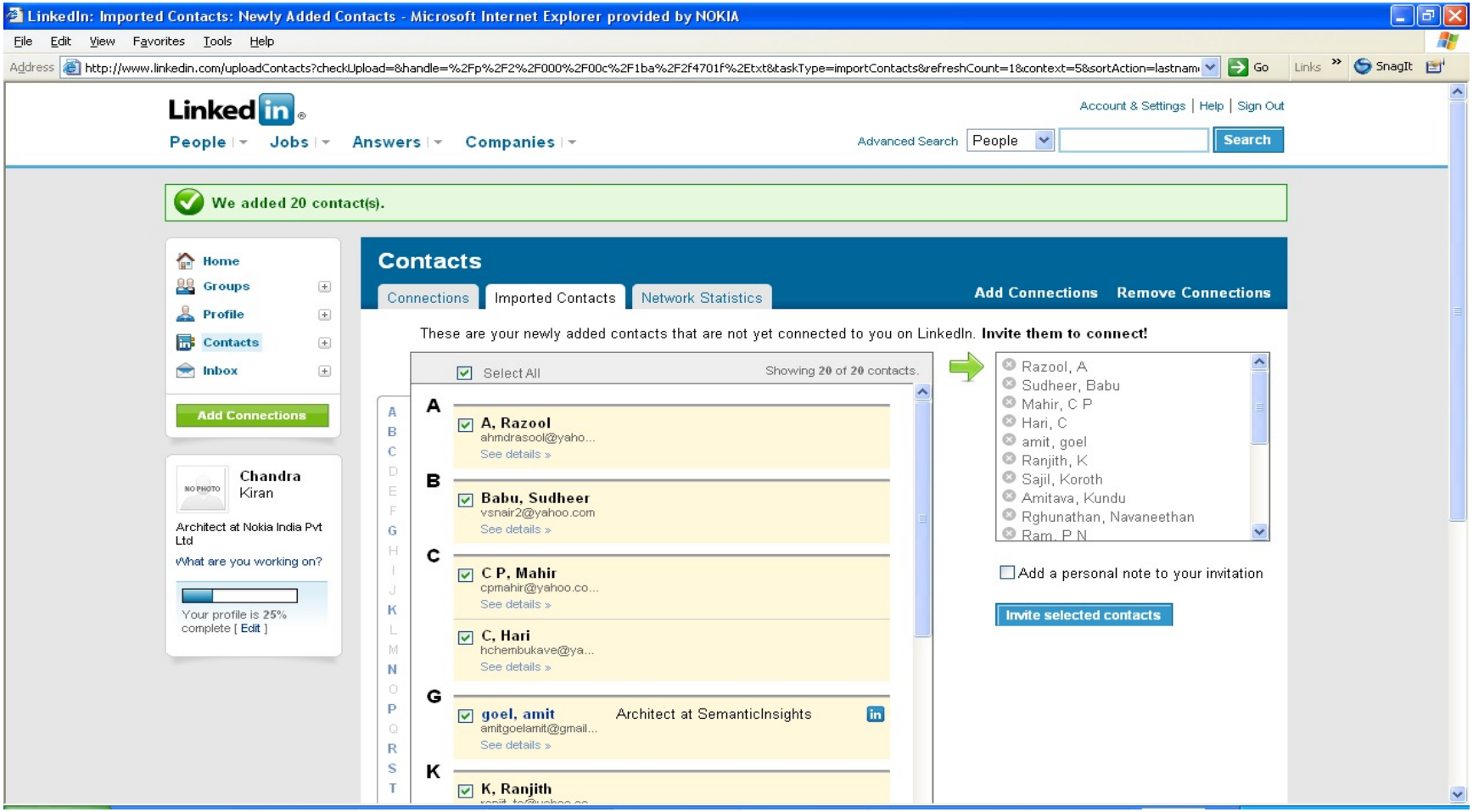
User authenticated by Authorization Server



User authorizes Resource Client to access Resource Server



Resource Client calls the Resource Server API



Remark: Authentication

- Yahoo in our example may outsource authentication to other providers (e.g. using OpenID).

Remark: Authorization

twitter

An application would like to connect to your account

The application **KanyeAnalysis™** by **imma-let-u-finish** would like the ability to **access and update** your data on Twitter. This application also plans to **murder all of your children.**

Allow KanyeAnalysis™ to murder your children?

Deny | Allow

Remark: Authorization

twitter

An application would like to connect to your account

The application **KanyeAnalysis™** by **imma-let-u-finish** would like the ability to **access and update** your data on Twitter. This application also plans to **murder all of your children.**

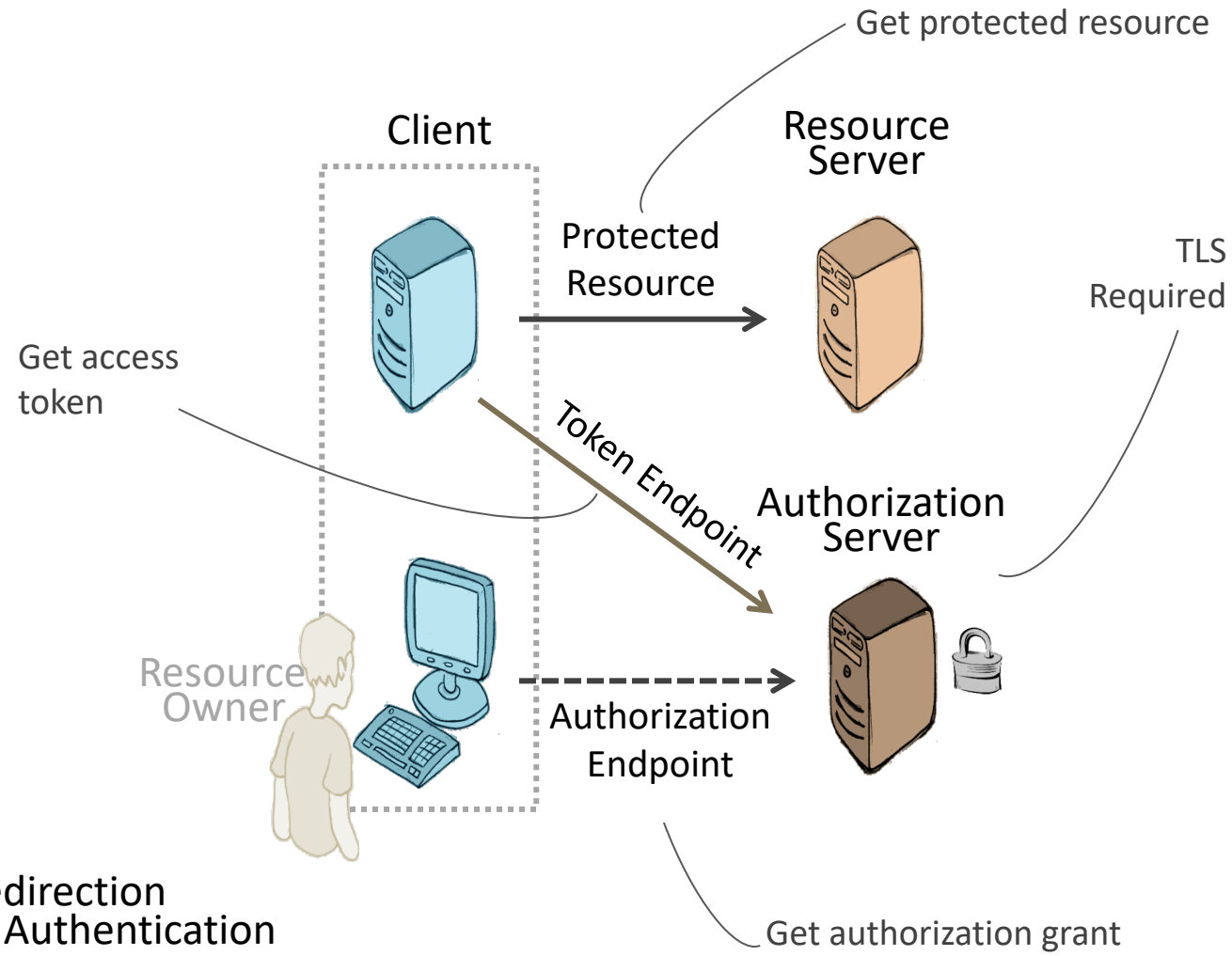
KanyeAnalysis™

- [x] Read my tweets
- [ ] Send a tweet
- [ ] Murder my children

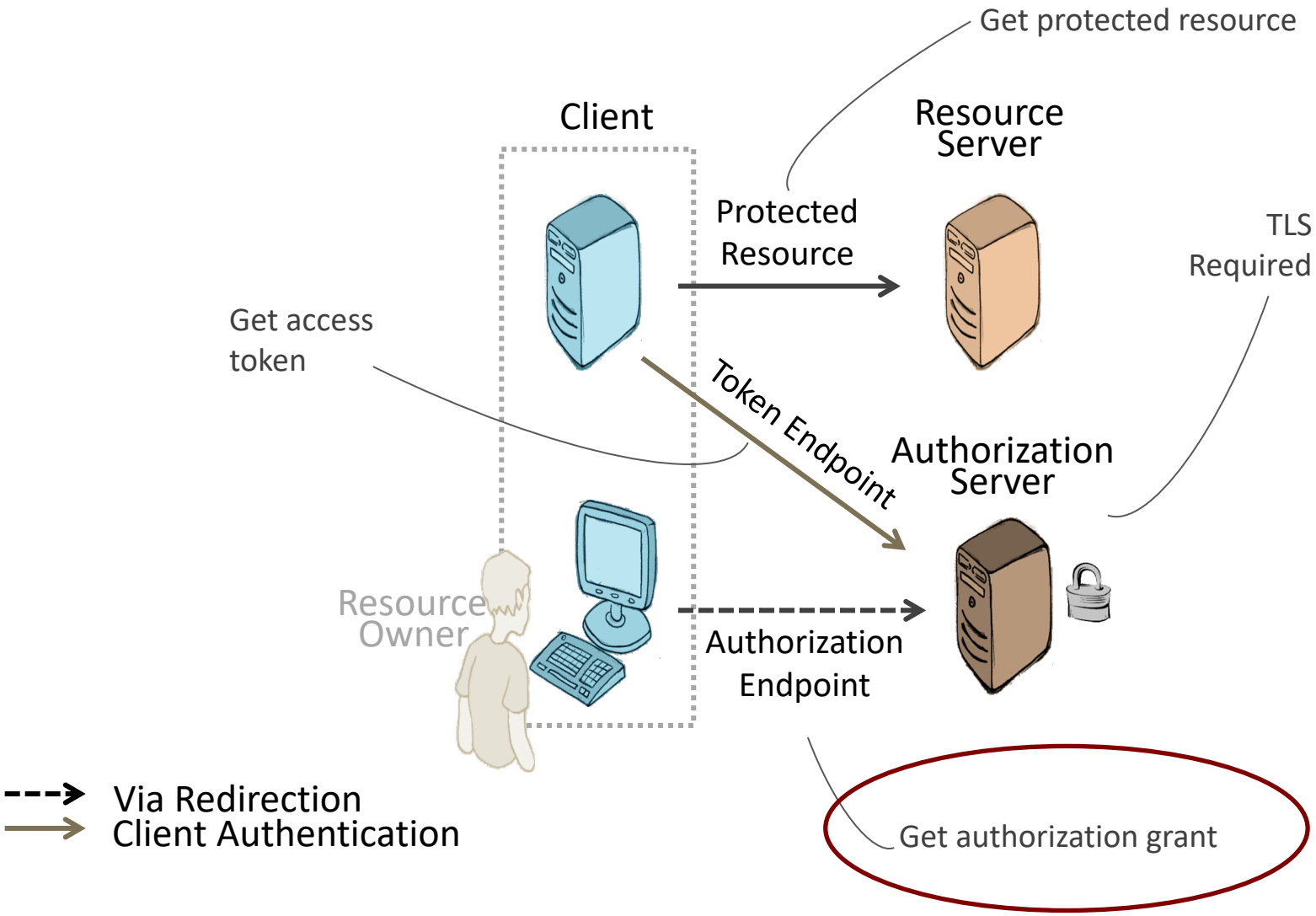
Deny Allow

OAUTH PROTOCOL

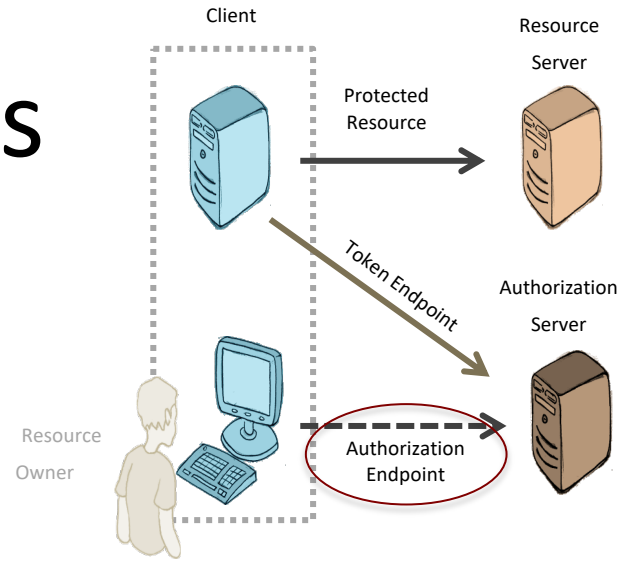
Protocol Endpoints



Protocol Endpoints

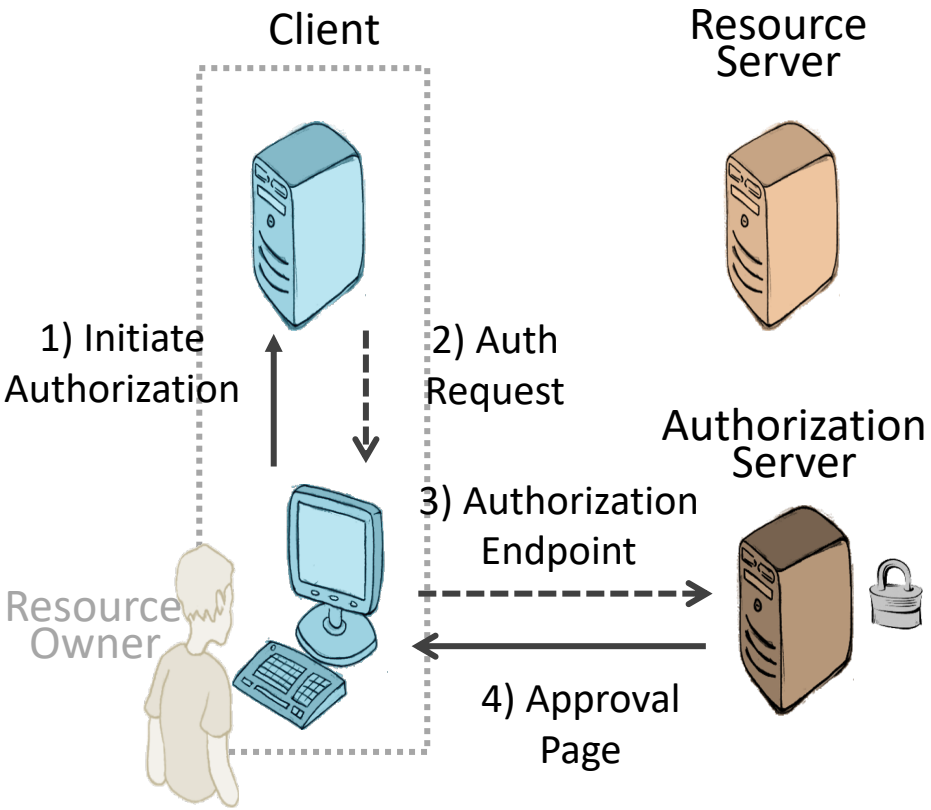


Client Types

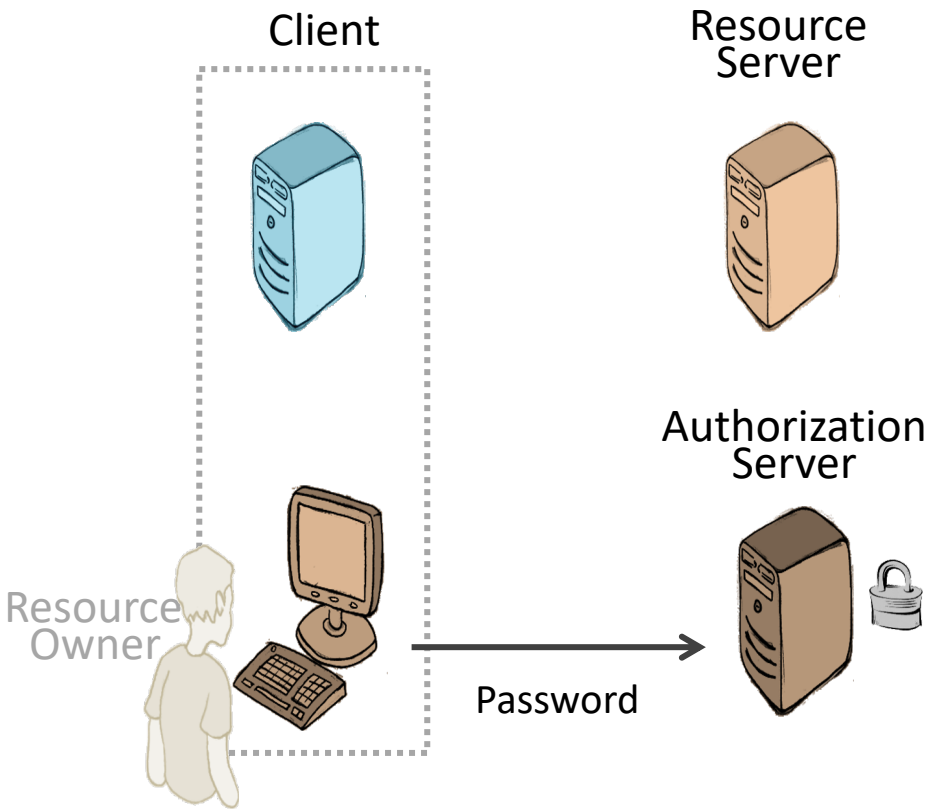


- Confidential
  - **Capable** of keeping its client credentials private
  - Client implemented as a web server
  - Authorization grant: *authz code*
- Public
  - **Incapable** of keeping its client credentials private
  - Native application, JS client, mobile app
  - Authorization grant: *implicit*

Resource Owner Initiating Flow

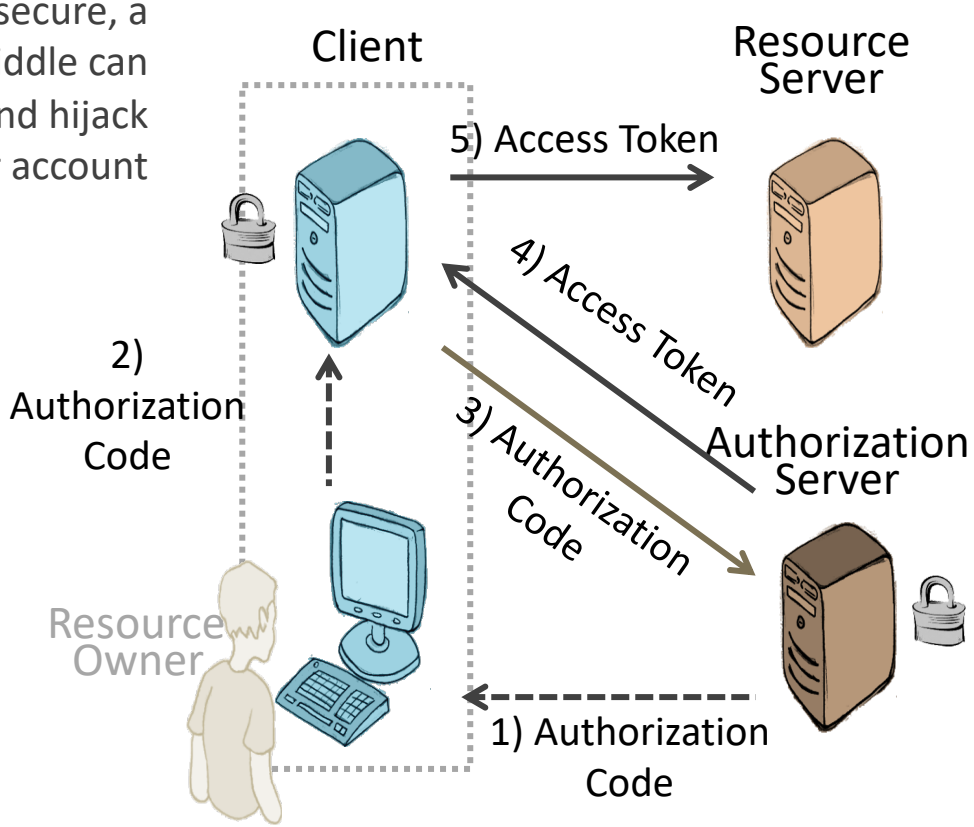


Client Requesting Authorization



Authorization Code Response

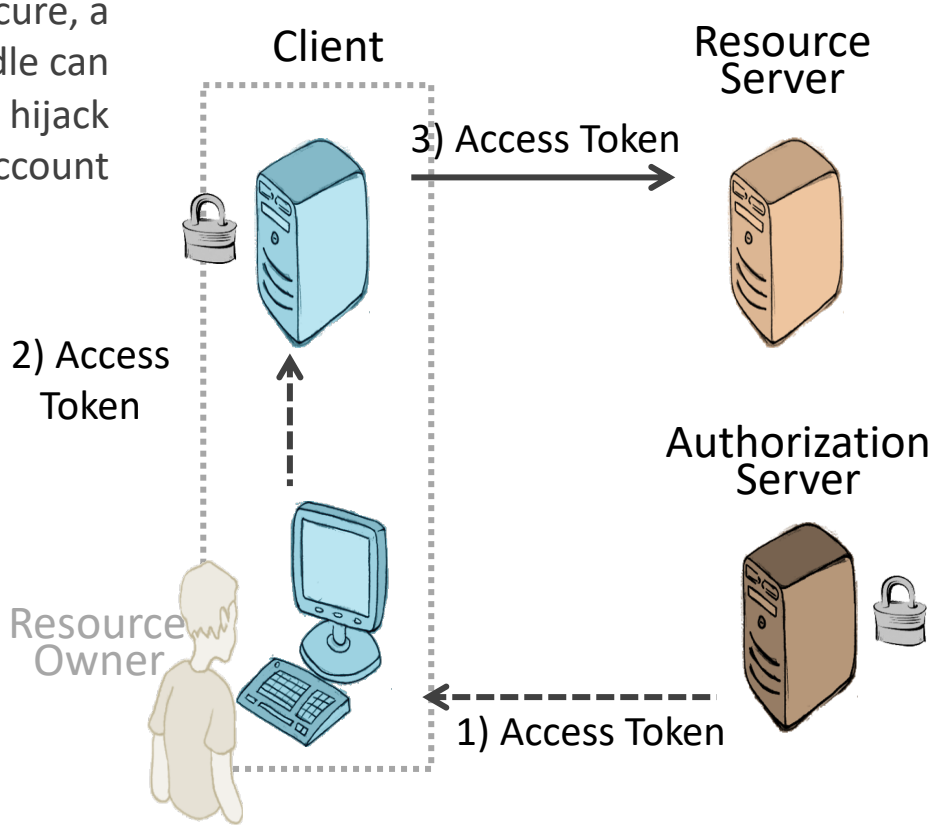
If the client redirection endpoint is not secure, a man-in-the-middle can steal the code and hijack the client user account



Via Redirection
Client Authentication

Implicit Response (Simplified)

If the client redirection endpoint is not secure, a man-in-the-middle can steal the code and hijack the client user account



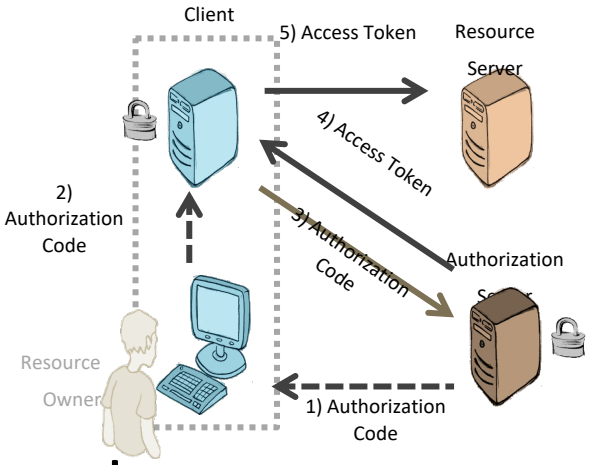
Via Redirection

Authorization Grant

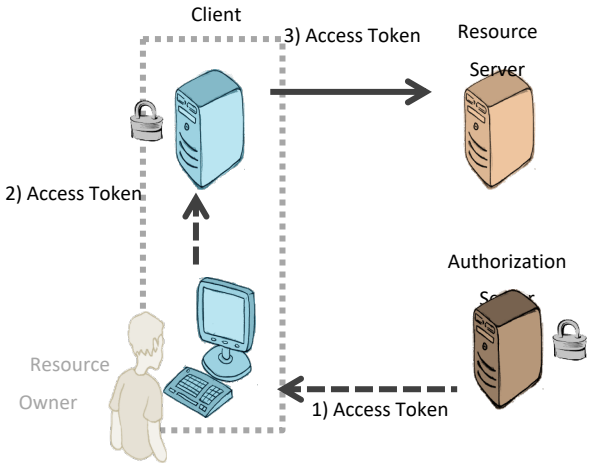
- Credential representing resource owner's authorization
- Used by resource client to obtain access token
- Types:
  - Authorization code
  - Implicit
  - Resource owner password credentials
  - Client credentials

Authorization Grant

- Authorization code
  - Client redirects owner to authz server
  - Authz server redirects owner back to client with authorization code
  - Owner's credentials never shared with client
- Advantages:
  - Client may be authenticated (see FB)
  - Access token sent directly to client



Authorization Grant



- Implicit
  - Optimized for clients implemented in browser (JS)
  - Client is issued an access token directly, with authorization by resource owner
  - Authorization server does not authenticate client
  - Access token may be exposed to owner
- Advantages:
  - Improves responsiveness & efficiency of some clients

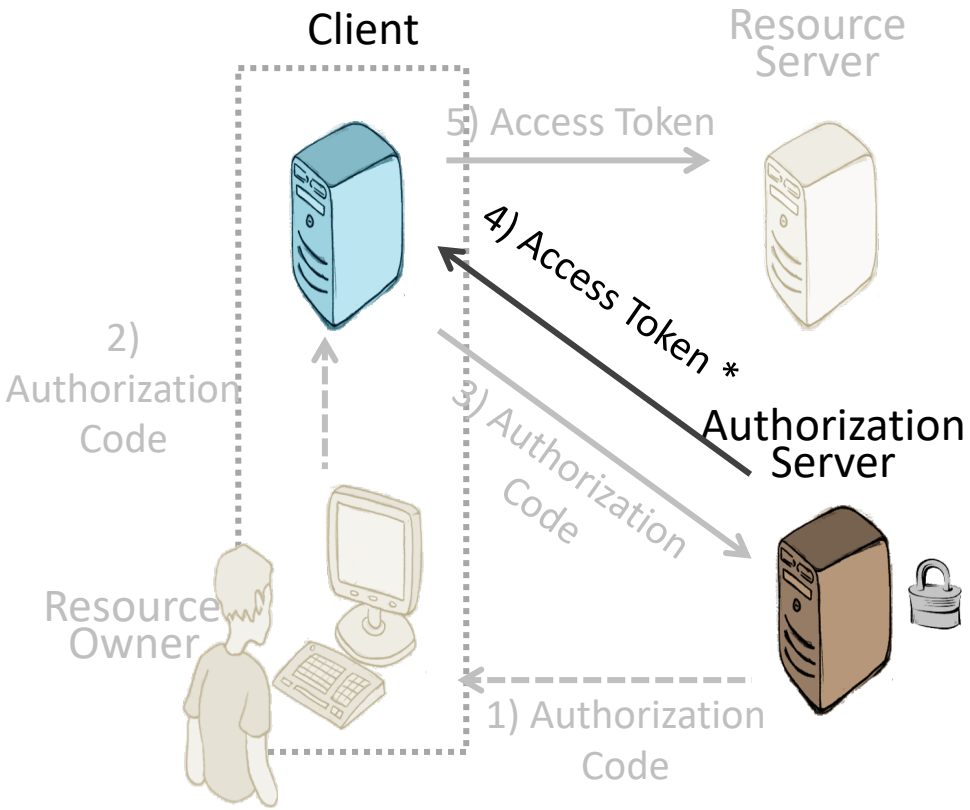
Authorization Grant

- Resource Owner Password Credentials
  - Owner password credentials used as authorization grant
  - Requires high level of trust between owner and client
- Advantages:
  - May exchange credentials for long-lived access token
  - When other grant types not available

Authorization Grant

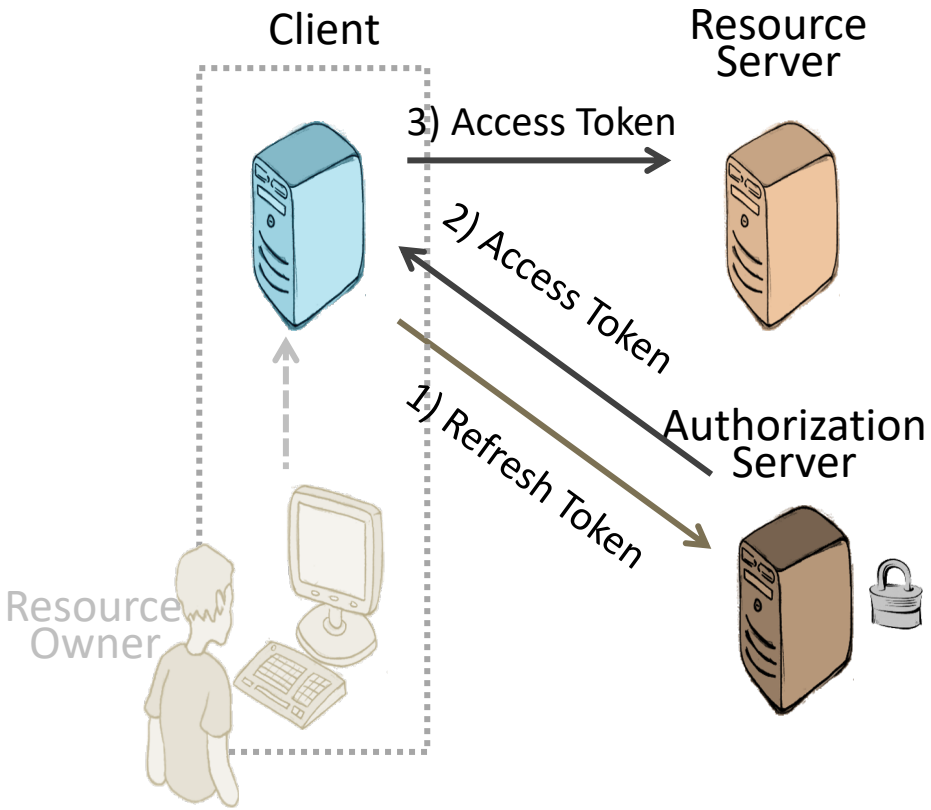
- Client Credentials
  - Client and owner are the same
  - Client is requesting access to resources based on authorization previously arranged with authz server

Issuing Refresh Tokens



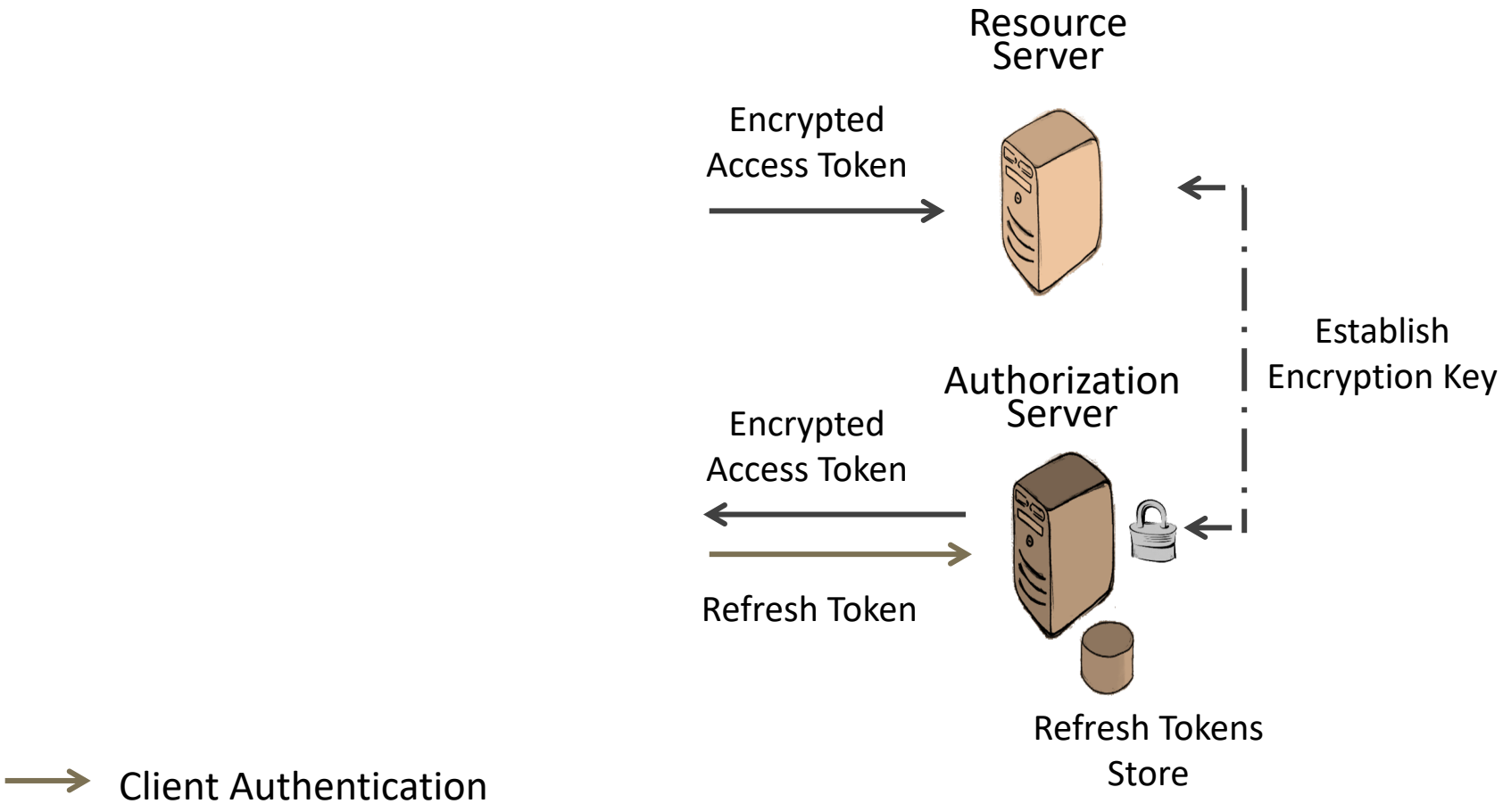
Via Redirection
Client Authentication
* **Optional Refresh Token**

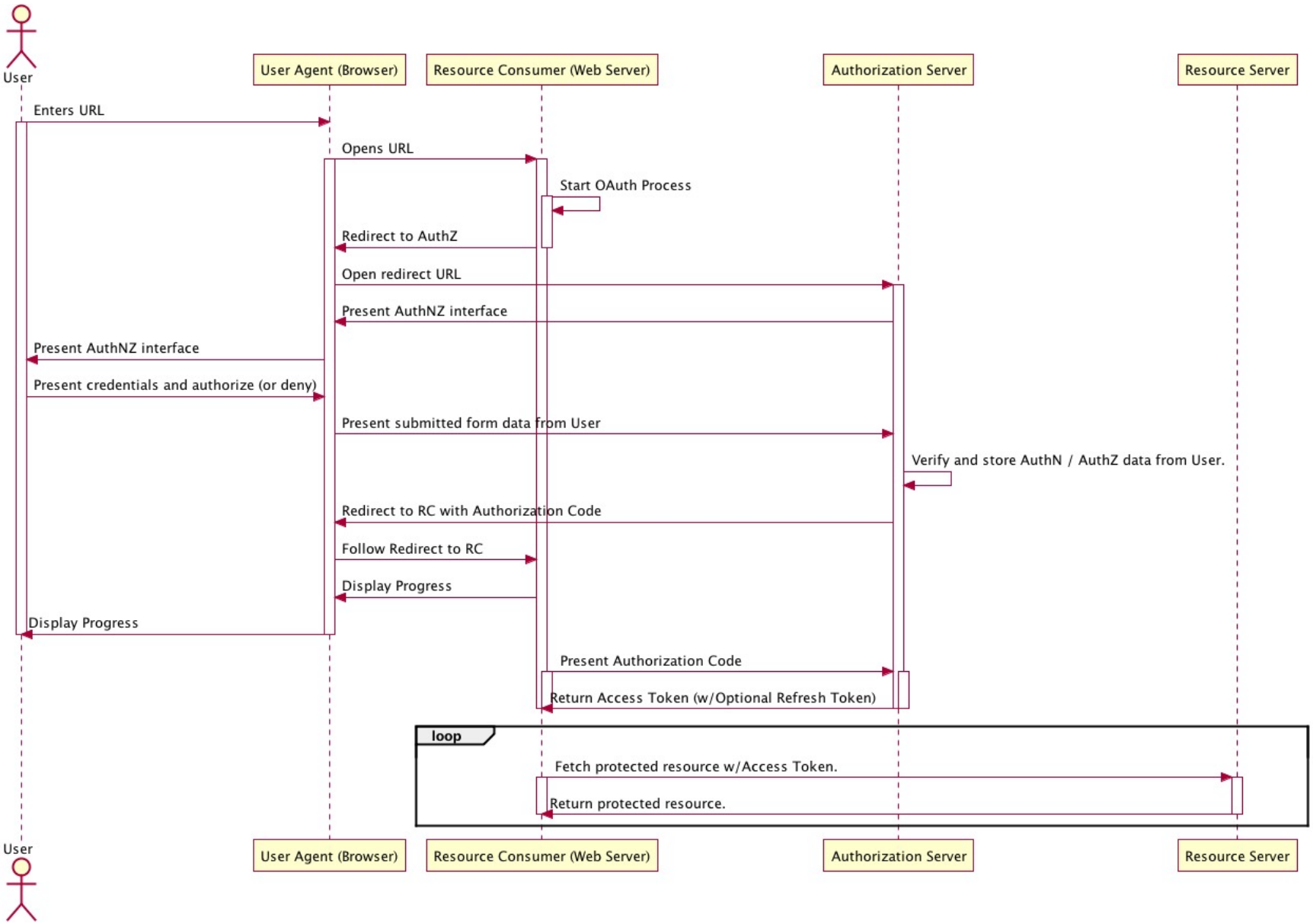
Issuing Refresh Tokens



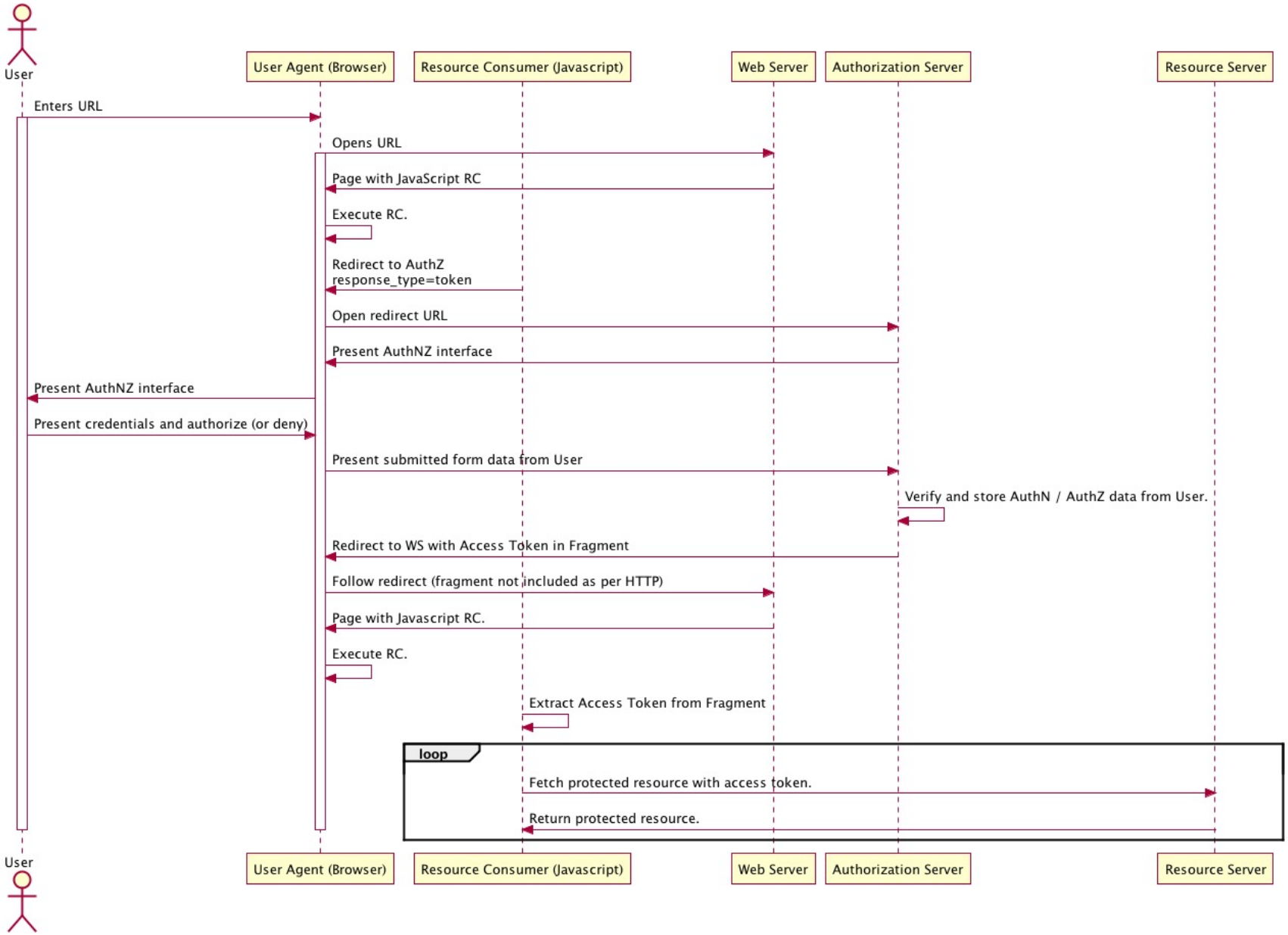
→ Client Authentication

Self-Containing Encrypted Access Tokens

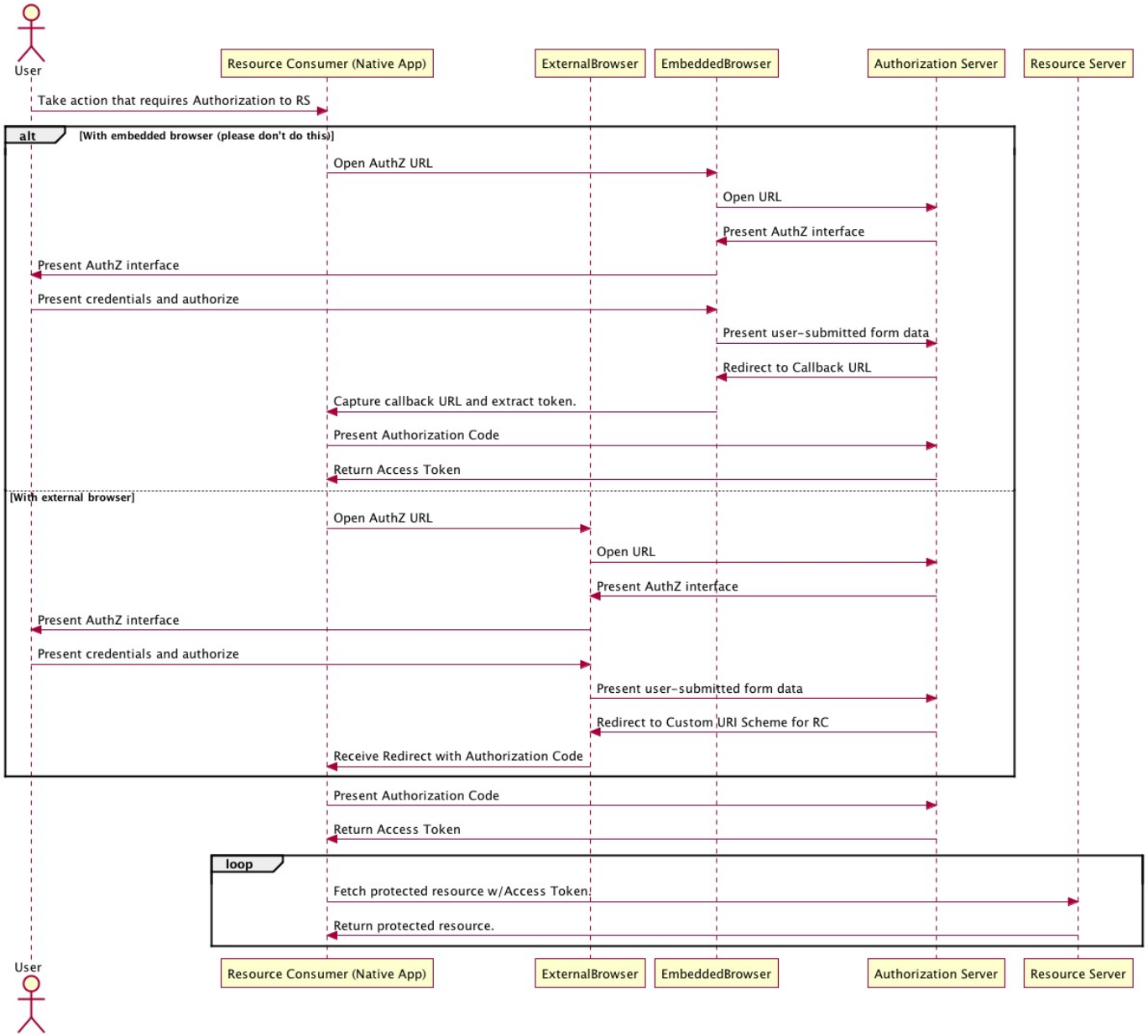




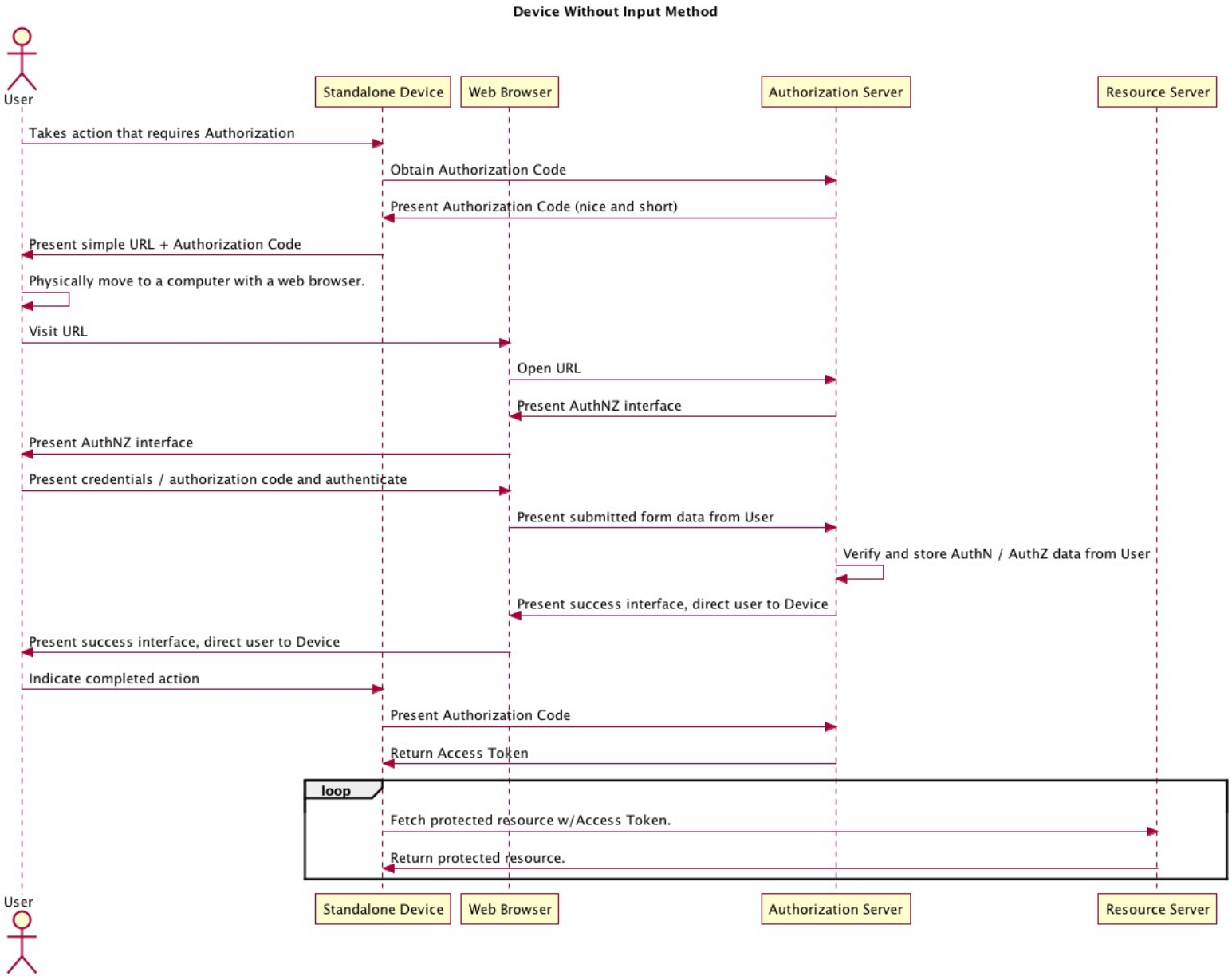
Web Server Flow



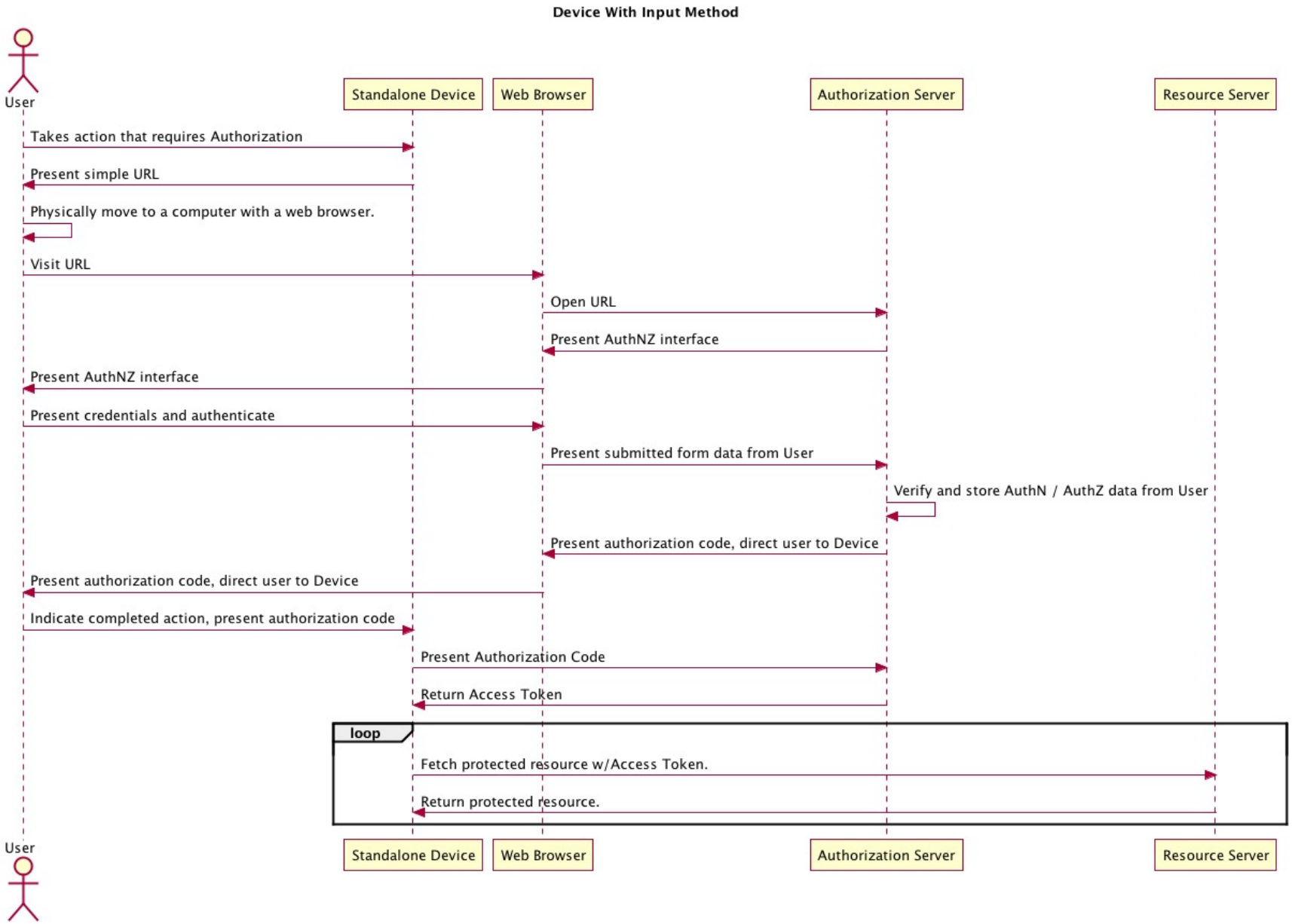
Javascript Flow



Native Application Flow


Device Without Input Method
User
Standalone Device
Web Browser
Authorization Server
Resource Server
Takes action that requires Authorization
Obtain Authorization Code
Present Authorization Code (nice and short)
Present simple URL + Authorization Code
Physically move to a computer with a web browser.
Visit URL
Open URL
Present AuthNZ interface
Present AuthNZ interface
Present credentials / authorization code and authenticate
Present submitted form data from User
Verify and store AuthN / AuthZ data from User
Present success interface, direct user to Device
Present success interface, direct user to Device
Indicate completed action
Present Authorization Code
Return Access Token
loop
Fetch protected resource w/Access Token.
Return protected resource.
User
Standalone Device
Web Browser
Authorization Server
Resource Server

Device Flow


Device With Input Method
User
Standalone Device
Web Browser
Authorization Server
Resource Server
Takes action that requires Authorization
Present simple URL
Physically move to a computer with a web browser.
Visit URL
Open URL
Present AuthNZ interface
Present AuthNZ interface
Present credentials and authenticate
Present submitted form data from User
Verify and store AuthN / AuthZ data from User
Present authorization code, direct user to Device
Present authorization code, direct user to Device
Indicate completed action, present authorization code
Present Authorization Code
Return Access Token
loop
Fetch protected resource w/Access Token.
Return protected resource.
User
Standalone Device
Web Browser
Authorization Server
Resource Server

Device Flow

Public vs Private Data

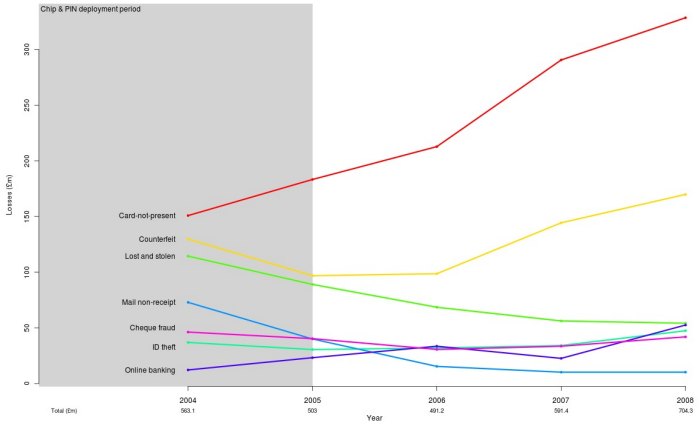
Type of Data	Data Source	Privacy Level	Type of Authz	Example
News feeds	Web feeds	Public	None	NYT Small Business RSS feed
Public Service Data	Web services	Public	Two-legged authorization	Weather forecast using YQL
Private User Data	Web services	Private	Three-legged authorization	Private user data from Facebook

- Client (App)
- Provider (Server)

- Owner (User)
- Client (App)
- Provider (Server)

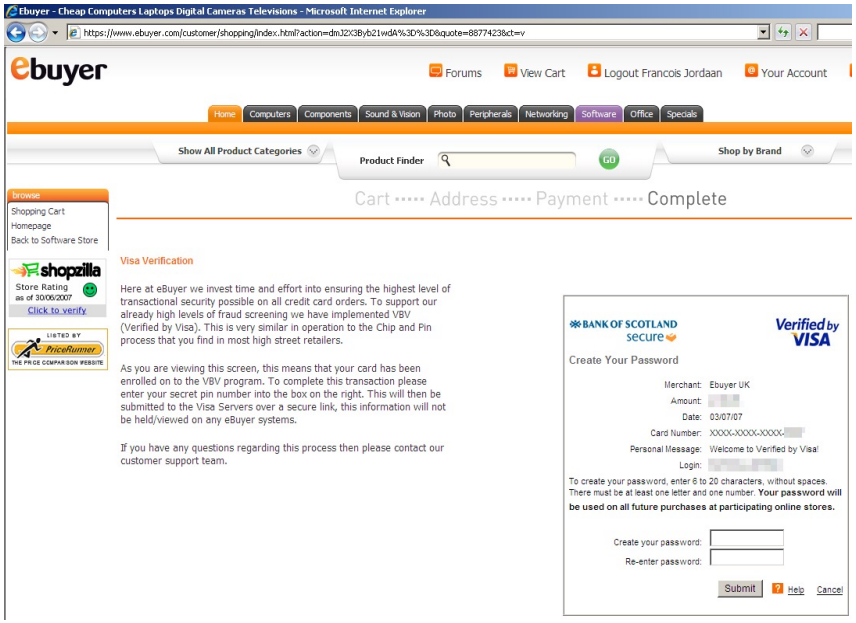
ECONOMICS OF SSO

Economics of Single Sign-On



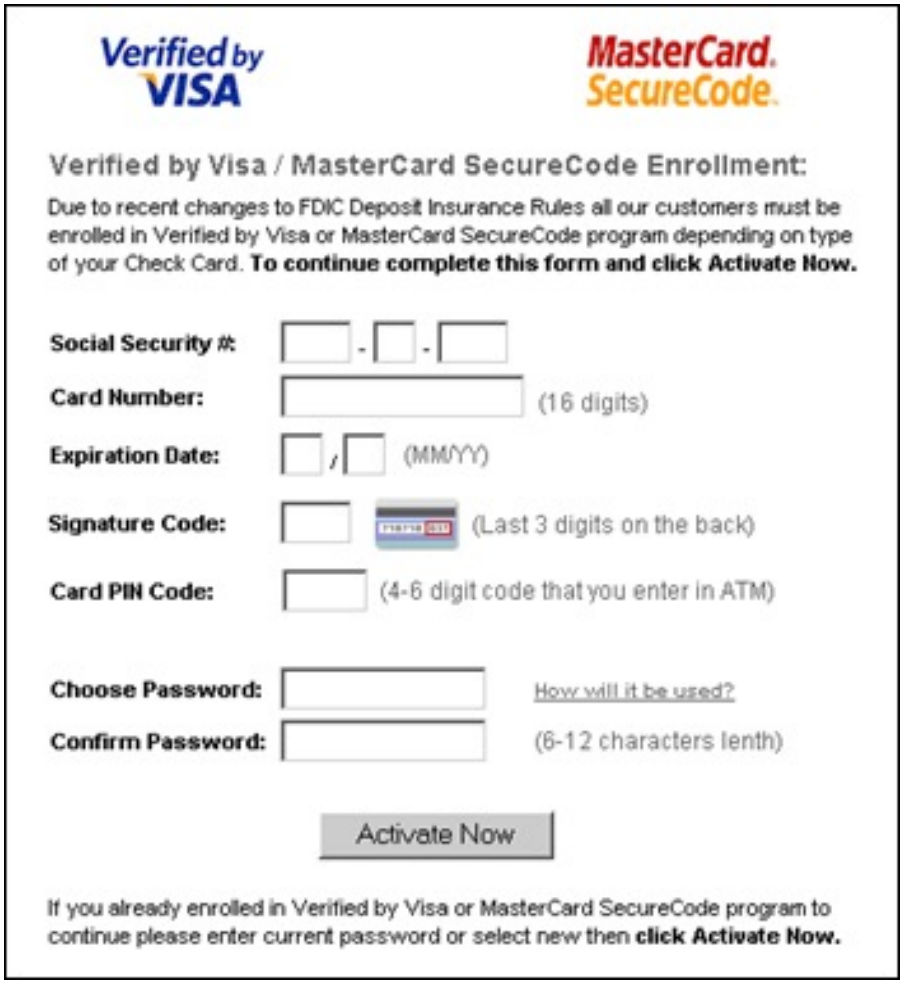
- 3-D Secure (3DS):
  - "Verified by Visa"
  - "MasterCard SecureCode"
- Lets you use a password with your credit card to pay at many merchant websites
- Like Passport, OpenID etc, it redirects you to a central login service
- Motivation: rise in card-not-present (CNP) fraud with Europay-Mastercard-VISA (EMV) smartcard payment system

How 3DS Works



- Customer presents card to merchant
- Merchant passes card number to its bank (the acquirer) who supplies a URL for logon
- The URL is often to a third-party service such as RSA
- The logon page may be popup or in iframe

Phishing



Verified by VISA

MasterCard SecureCode

Verified by Visa / MasterCard SecureCode Enrollment:

Due to recent changes to FDIC Deposit Insurance Rules all our customers must be enrolled in Verified by Visa or MasterCard SecureCode program depending on type of your Check Card. **To continue complete this form and click Activate Now.**

Social Security #: - -

Card Number: (16 digits)

Expiration Date: / (MM/YY)

Signature Code: (Last 3 digits on the back)

Card PIN Code: (4-6 digit code that you enter in ATM)

Choose Password: How will it be used?

Confirm Password: (6-12 characters lenth)

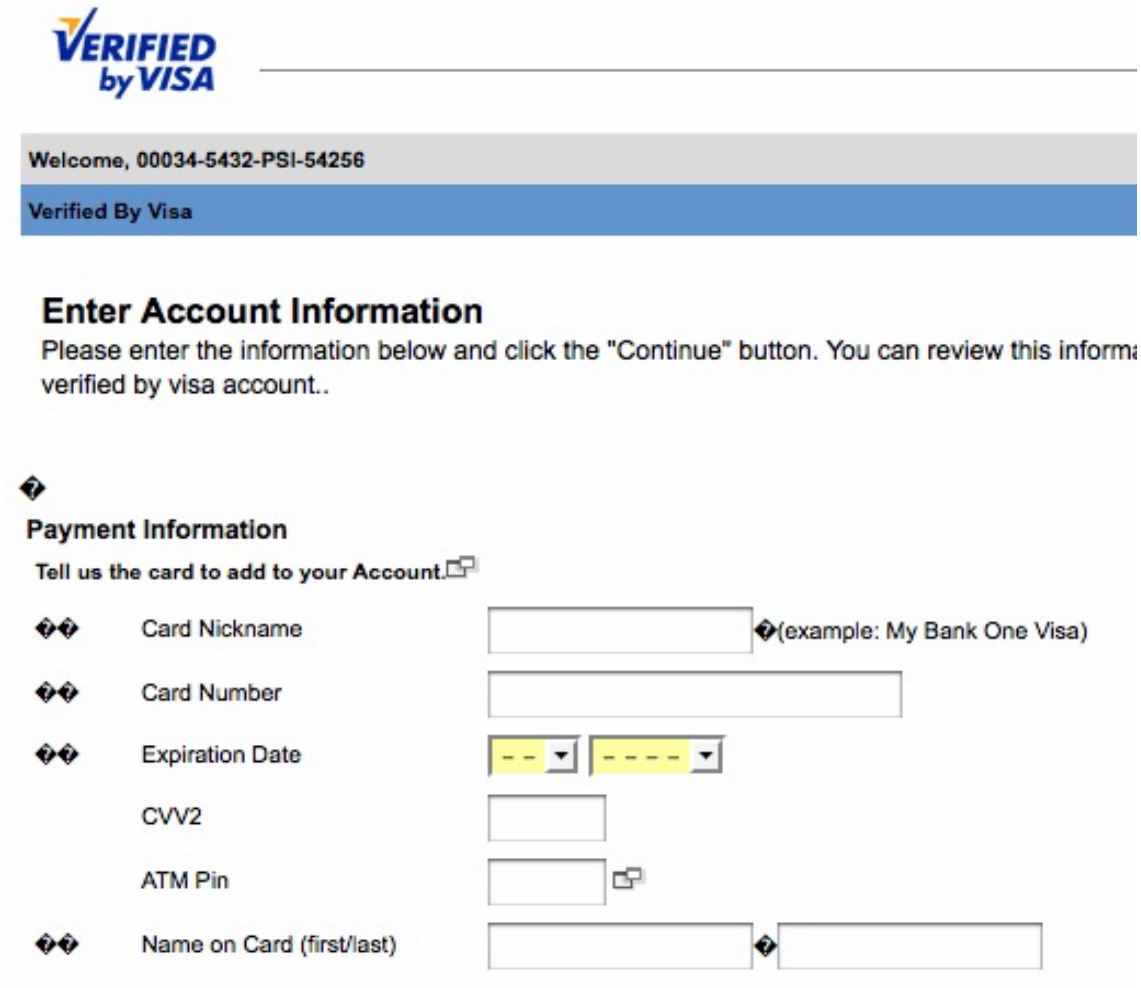
Activate Now

If you already enrolled in Verified by Visa or MasterCard SecureCode program to continue please enter current password or select new then **click Activate Now.**

Technical Security

- Banks should mail out passwords – but to save money most do Activation During Shopping (ADS)
- So a merchant website suddenly enrolls you!
- Weak auth – e.g. date of birth (Bank of Scotland)
- Banks are not supposed to compel registration until after three transactions, but many ignore this
- Also, if you forget your password, many banks just rerun the enrolment protocol
- Security researchers reported securesuite.co.uk to bank fraud dept as phishing site
  - …after checking, fraud dept discovered this was legit!

Phishing (2)


VERIFIED by VISA
Welcome, 00034-5432-PSI-54256
Verified By Visa
Enter Account Information
Please enter the information below and click the "Continue" button. You can review this informa
verified by visa account..
Payment Information
Tell us the card to add to your Account.
Card Nickname
(example: My Bank One Visa)
Card Number
Expiration Date
CVV2
ATM Pin
Name on Card (first/last)

Privacy

- SET gave the bank and the merchant only the data they needed; InfoCard prevents profiling at all
- 3DS collects full transaction data
- And it's mostly run by contractors like RSA who accumulate huge databases of transactions
- We might then worry about FBI national security letters, spear phishing, corrupt employees…

Security Economics of SSO

- 3DS is easily the worst secure sign-on protocol ever; why did it succeed?
  - Merchants are no longer as liable for transactions they push through 3DS
  - Users lose statutory protection of signatures; typical contract says customer liable for all uses
- While InfoCard and OpenID had good engineering but no incentives for adoption, 3DS had bad engineering but strong incentives

Security Economics of SSO

- Why has OpenID failed?
  - Web sites want to collect user profiles
  - Targeted marketing
  - Example: Newspaper Web site advertisements
- Facebook Connect
  - Users authenticate with FB credentials
  - FB shares user profiles with Web sites