Secure Services with Apache CXF

Andrei Shakirin, Talend

ashakirin@talend.com
ashakirin.blogspot.com/

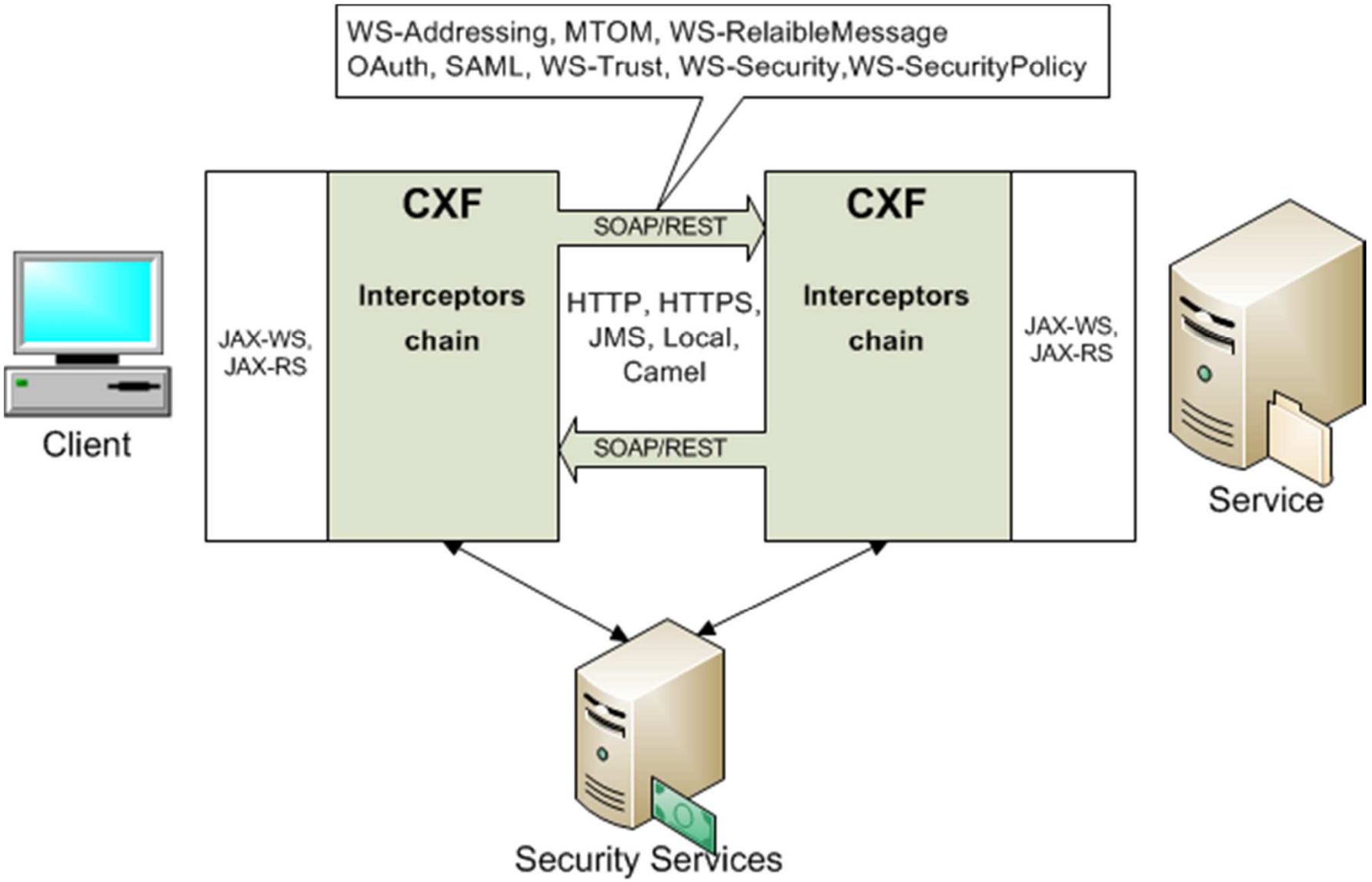
Agenda

- Introduction in Apache CXF
- Security Requirements
- Apply security features to CXF Services (JAX-RS)

About Me

- Software architect in Talend Team
- PMC and committer in Apache CXF and commiter in Apache Syncope projects
- Speaker for Apache and Java conferences

Apache CXF


WS-Addressing, MTOM, WS-RelaibleMessage
OAuth, SAML, WS-Trust, WS-Security,WS-SecurityPolicy
CXF
Interceptors chain
SOAP/REST
HTTP, HTTPS, JMS, Local, Camel
SOAP/REST
CXF
Interceptors chain
JAX-WS, JAX-RS
JAX-WS, JAX-RS
Client
Service
Security Services

Why CXF?

Alternatives:

- **Jersey**
- **RestEasy**
- **Axis 2**
- **Metro**
- **…**

CXF Benefits:

- **Strong standards support**
- **SOAP & Rest services**
- **Comprehensive Security**
- **Streaming and performance**
- **Flexibility**
- **Large and active community**

Who uses CXF?

- Apache: Camel, ServiceMix, Syncope
- JBoss JAX-WS stack
- TomEE JAX-WS and JAX-RS stacks
- Talend, Fusesource, MuleSoft, WSO2
- Google Adwords, TomTom, …

Security Requirements

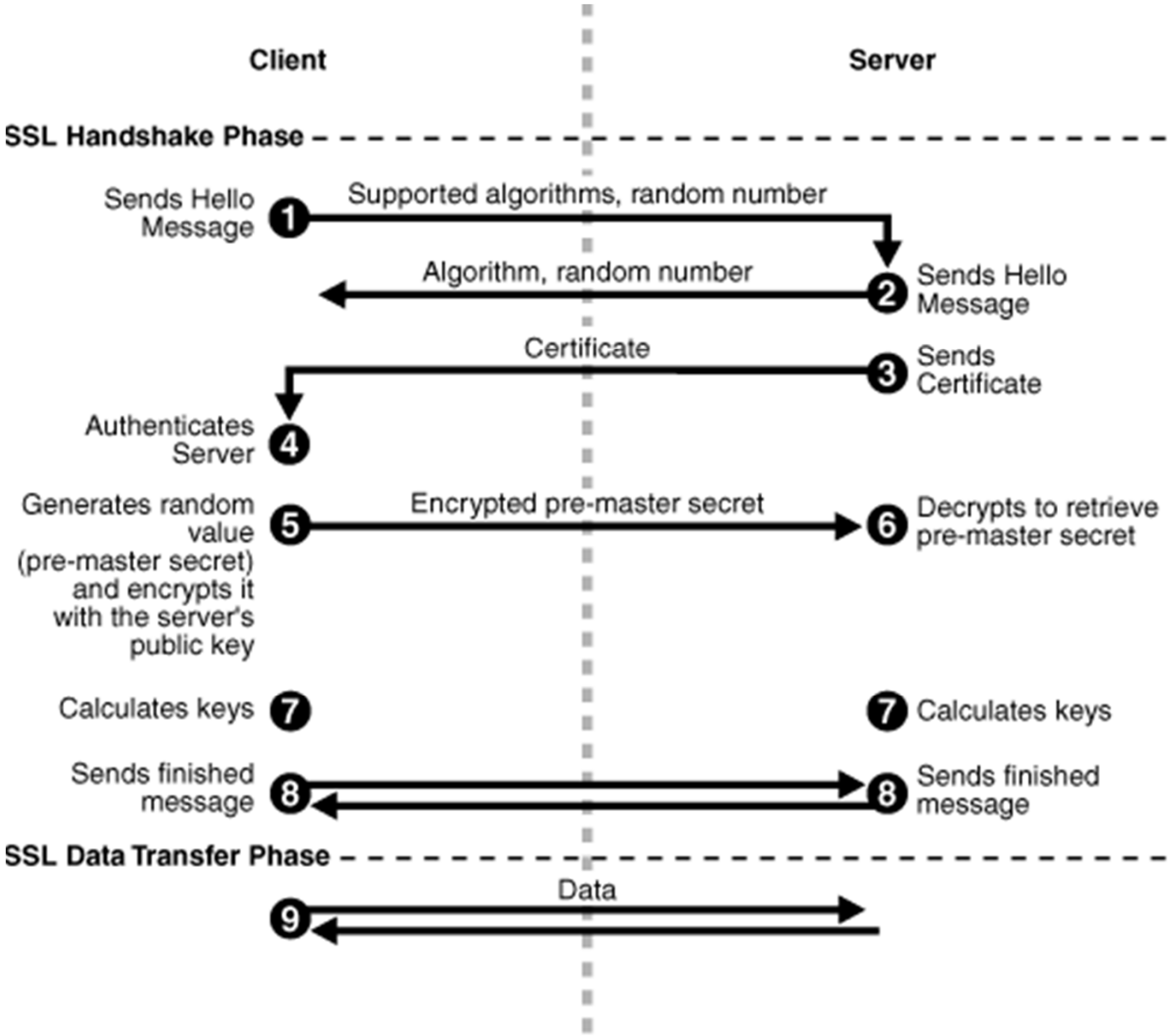
- **Authentication** (HTTP basic, digest, UsernameToken, X500, Kerberos, SAML)
- **Authorization** (method/resource based, XACML)
- **Confidentiality** (SSL, message encryption)
- **Integrity** (SSL, message signature)
- **Non-repudiation** (message signature)

Transport Layer Security

- **Authentication**
- **Confidentiality**
- **Integrity**

Is SSL Safe?

SSL Handshake



Server Certificate Validation

1. Check certificate validity period
2. Check is Certificate Authority (CA) a trusted CA
3. Check the issuer's digital signature in whole chain
4. Check if domain name in the server's certificate match the domain name of the server itself
5. Check CA revocation list

Recommendations

- Apply negative security tests using abnormal SSL certificates (self-signed, issues for another host, etc)
- Don‘t modify application code and disable certificate validation even for testing
- Verify libraries default set up, use explicit options if necessary

HTTPS: JAX-RS 2.0 Client


```
public Response startClientJAXRS() throws NoSuchAlgorithmException {

    ClientBuilder builder = ClientBuilder.newBuilder();

    KeyStore trustStore = loadStore("truststore.jks", "secret");
    builder.trustStore(trustStore);

    KeyStore keyStore = loadStore("keystore.jks", "secret");
    builder.keyStore(keyStore, "password");

    builder.hostnameVerifier(CertificateHostnameVerifier.STRICT);

    Response response = builder.build()
                        .target("https://api.syncope.org:8443/customers/ca1b2c3")
                        .request()
                        .accept(MediaType.APPLICATION_XML_TYPE)
                        .get();

    return response;
}
```

HTTPS: CXF Client


```
public Customer startCXFClientJAXRS() throws NoSuchAlgorithmException,
        KeyStoreException, UnrecoverableKeyException {

    CustomerService proxy = JAXRSClientFactory.create("https://api.syncope.org:8443",
            CustomerService.class);

    HTTPConduit conduit = WebClient.getConfig(proxy).getHttpConduit();

    TLSClientParameters tlsParams = new TLSClientParameters();
    conduit.setTlsClientParameters(tlsParams);

    KeyStore trustStore = loadStore("truststore.jks", "secret");

    TrustManagerFactory trustFactory = TrustManagerFactory
            .getInstance(TrustManagerFactory.getDefaultAlgorithm());
    trustFactory.init(trustStore);
    TrustManager[] tm = trustFactory.getTrustManagers();
    tlsParams.setTrustManagers(tm);

    KeyStore keyStore = loadStore("keystore.jks", "secret");
    KeyManagerFactory keyFactory = KeyManagerFactory
            .getInstance(KeyManagerFactory.getDefaultAlgorithm());
    keyFactory.init(keyStore, "secret".toCharArray());
    KeyManager[] km = keyFactory.getKeyManagers();
    tlsParams.setKeyManagers(km);

    return proxy.getCustomer("ca1b2c3");
}
```

HTTPS Server: Jetty


```
<httpj:engine-factory bus="cxf">
    <httpj:engine port="9000">
        <httpj:tlsServerParameters>
            <sec:keyManagers keyPassword="skpass">
                <sec:keyStore file="src/main/config/serviceKeystore.jks" password="sspass" type="JKS" />
            </sec:keyManagers>
            <sec:trustManagers>
                <sec:keyStore file="src/main/config/trustStore.jks" password="sspass" type="JKS" />
            </sec:trustManagers>
            <sec:cipherSuitesFilter>
                <!-- these filters ensure that a ciphersuite with export-suitable
                    or null encryption is used, but exclude anonymous Diffie-Hellman key change
                    as this is vulnerable to man-in-the-middle attacks -->
                <sec:include>.*_EXPORT_.*</sec:include>
                <sec:include>.*_EXPORT1024_.*</sec:include>
                <sec:include>.*_WITH_DES_.*</sec:include>
                <sec:include>.*_WITH_AES_.*</sec:include>
                <sec:include>.*_WITH_NULL_.*</sec:include>
                <sec:exclude>.*_DH_anon_.*</sec:exclude>
            </sec:cipherSuitesFilter>
            <sec:clientAuthentication want="true"
                required="true" />
        </httpj:tlsServerParameters>
    </httpj:engine>
</httpj:engine-factory>
```

Authentication

- HTTP basic, digest
- Kerberos
- X509
- SAML
- JWT

HTTP Basic

1. Client:
GET /users/ua1b2c3 HTTP/1.0

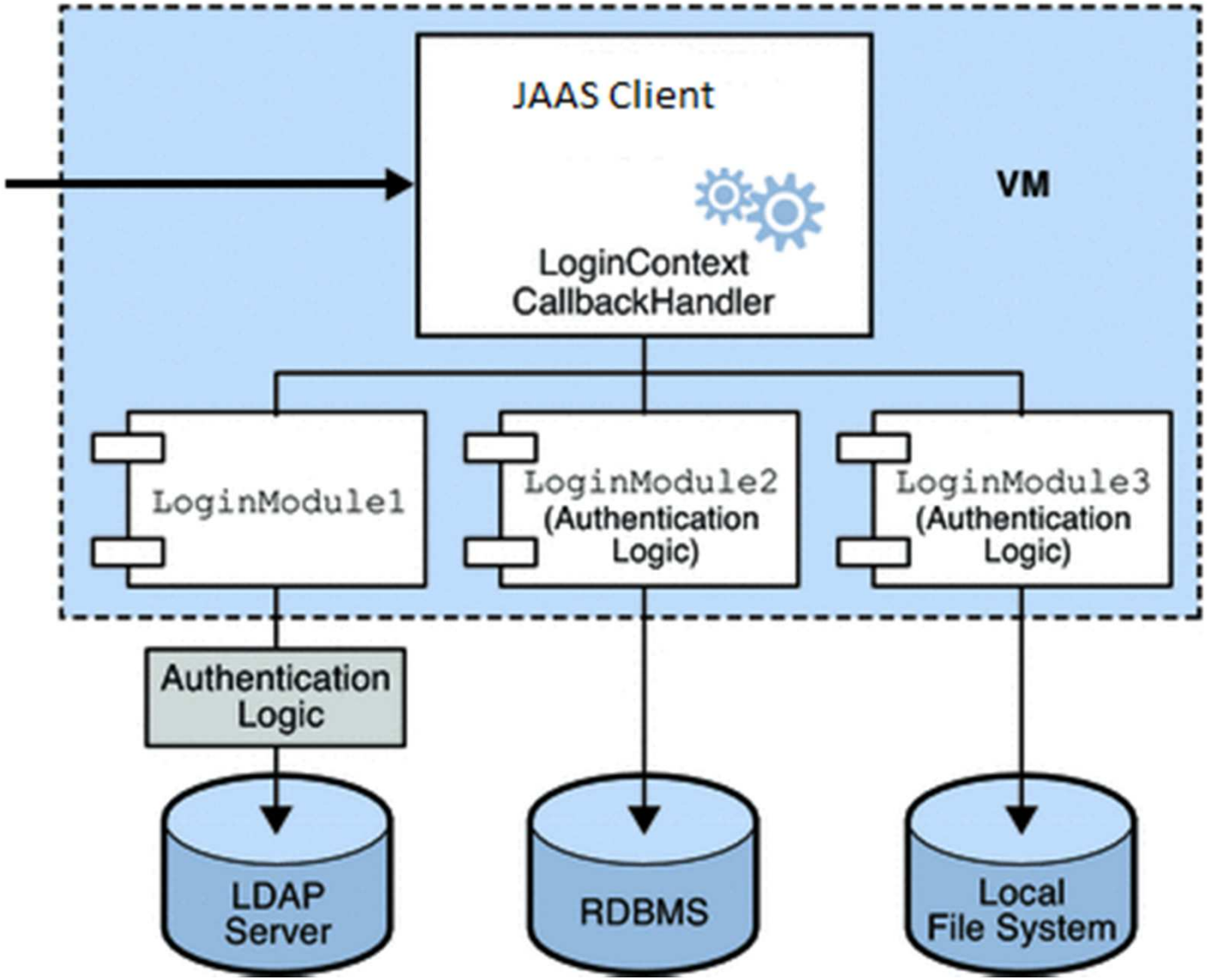
2. Server:
HTTP/1.0 401 Unauthorized
WWW-Authenticate: Basic realm="*nmrs_m7VKmomQ2YM3:*"

3. Client:
GET /users/ua1b2c3 HTTP/1.0
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

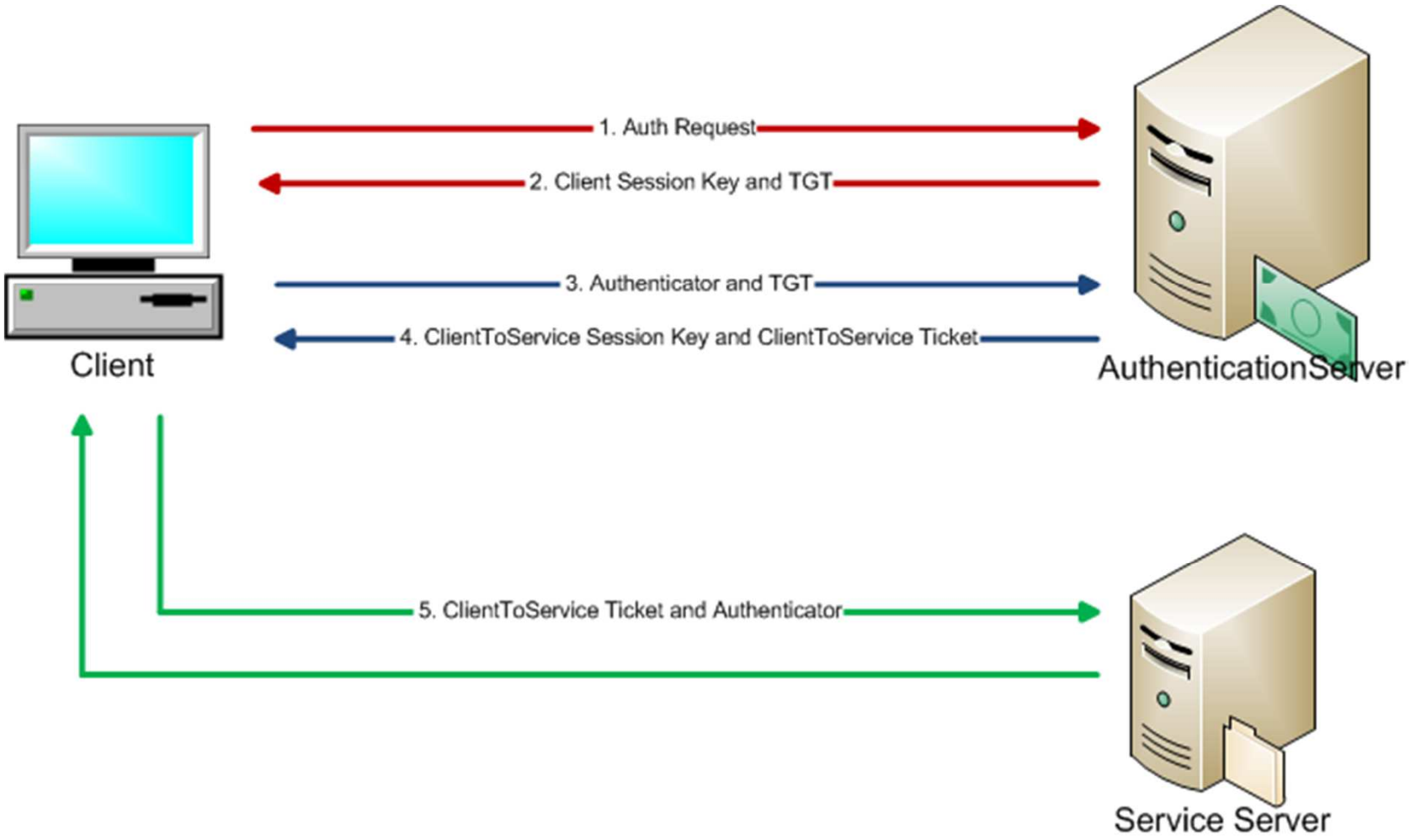
HTTP Digest


```
1. Client:
GET /users/ua1b2c3 HTTP/1.0

2. Server:
HTTP/1.0 401 Unauthorized
WWW-Authenticate: Digest realm="testrealm@host.com",
                         qop="auth,auth-int",
                         nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                         opaque="5ccc069c403ebaf9f0171e9517f40e41"

3. Client:
GET / /users/ua1b2c3 HTTP/1.0
Authorization: Digest username=“ashakirin",
                      realm="testrealm@host.com",
                      nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                      uri="/dir/index.html",
                      qop=auth,
                      nc=00000001,
                      cnonce="0a4f113b",
                      response="6629fae49393a05397450978507c4ef1",
                      opaque="5ccc069c403ebaf9f0171e9517f40e41"


 HA1=MD5(username:realm:password)
 HA2=MD5(method:digestURI)
 response=MD5(HA1:nonce:HA2)
```

JAAS


JAAS Client
LoginContext
CallbackHandler
VM
LoginModule1
LoginModule2
(Authentication
Logic)
LoginModule3
(Authentication
Logic)
Authentication
Logic
LDAP
Server
RDBMS
Local
File System

Kerberos


1. Auth Request
2. Client Session Key and TGT
3. Authenticator and TGT
4. ClientToService Session Key and ClientToService Ticket
Client
AuthenticationServer
5. ClientToService Ticket and Authenticator
Service Server

SAML


```
<saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    ID="_EAF1D65DB246FCB19013668749383952" IssueInstant="2013-04-25T07:28:58.395Z"
    Version="2.0" xsi:type="saml2:AssertionType">
    <saml2:Issuer>issuer]</saml2:Issuer>
    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
            <ds:Reference URI="#_EAF1D65DB246FCB19013668749383952">
                <ds:DigestMethod
                    Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                <ds:DigestValue>BeqHM3hDi2jkd1i/9VgL52HpqtE=</ds:DigestValue>
            </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>d4rbn3fo1t1349pri</ds:SignatureValue>
        <ds:KeyInfo>
            <ds:X509Data>
                <ds:X509Certificate>MIIC</ds:X509Certificate>
            </ds:X509Data>
        </ds:KeyInfo>
    </ds:Signature>
    <saml2:Subject>
        <saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
            NameQualifier="http://cxf.apache.org/sts">alice@EXAMPLE.COM</saml2:NameID>
        <saml2:SubjectConfirmation
            Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
            <saml2:SubjectConfirmationData
                xsi:type="saml2:KeyInfoConfirmationDataType">
                <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                    <ds:X509Data>
                        <ds:X509Certificate>MIIEFjCCA3+</ds:X509Certificate>
                    </ds:X509Data>
                </ds:KeyInfo>
            </saml2:SubjectConfirmationData>
        </saml2:SubjectConfirmation>
    </saml2:Subject>
    <saml2:Conditions NotBefore="2013-04-25T07:28:58.397Z"
        NotOnOrAfter="2013-04-25T07:58:58.397Z">
        <saml2:AudienceRestriction>
            <saml2:Audience>https://localhost:9001/services/ReservationServiceProvider
            </saml2:Audience>
        </saml2:AudienceRestriction>
    </saml2:Conditions>
    <saml2:AttributeStatement>
        <saml2:Attribute
            Name="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"
            NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
            <saml2:AttributeValue xsi:type="xs:string">manager</saml2:AttributeValue>
        </saml2:Attribute>
    </saml2:AttributeStatement>
</saml2:Assertion>
```

SAML in Rest Services

1. Enveloped

```
<env:Envelope xmlns:env="http://org.apache.cxf/rs/env">
   <Book ID="67ca6441-0c4e-4430-af0e-9463ce9226aa">
      <id>125</id>
      <name>CXF</name>
    </Book>
    <!-- SAML assertion with an enveloped signature -->
    <saml2:Assertion>
    ...
    </saml2:Assertion>
</env:Envelope>
```

2. Authorization header

Address: https://localhost:9000/samlheader/bookstore/books/123

Http-Method: GET

Headers: {Accept=[application/xml], Authorization=[SAML eJydV1mTokgQf …]}

3. Form values

Encoding: ISO-8859-1

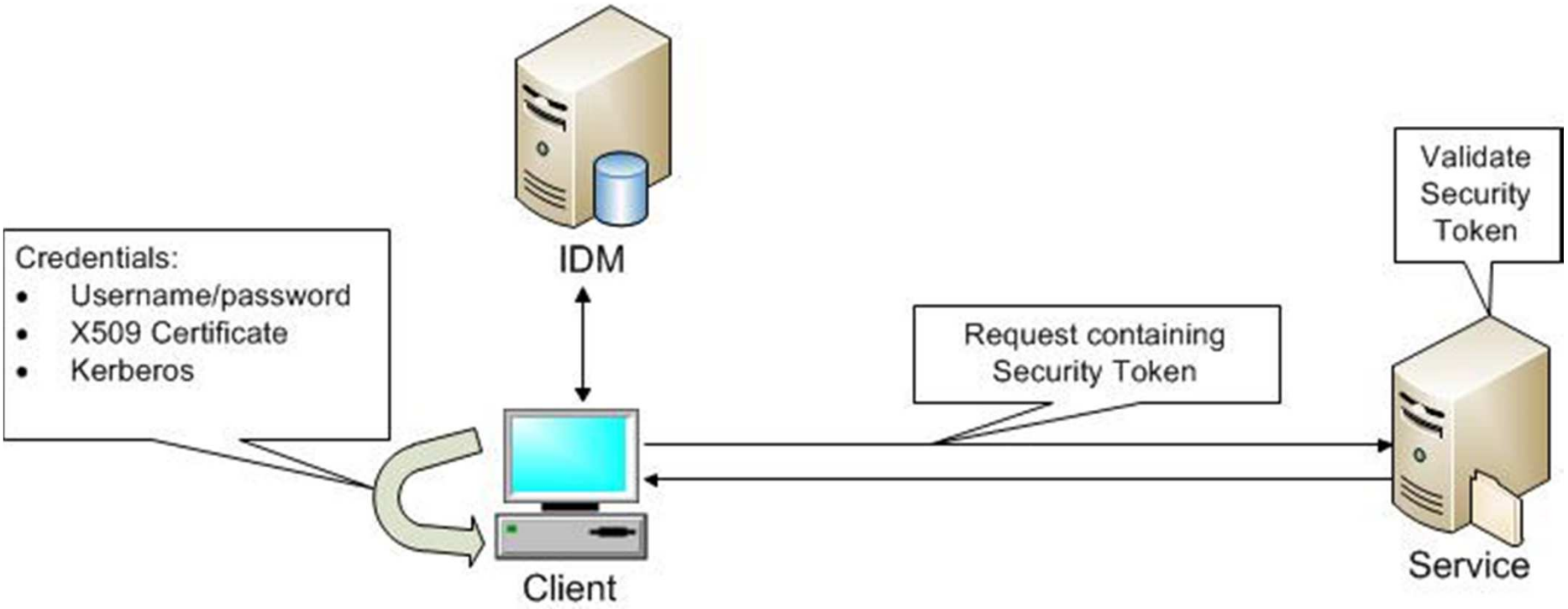
Http-Method: POST

Content-Type: application/x-www-form-urlencoded

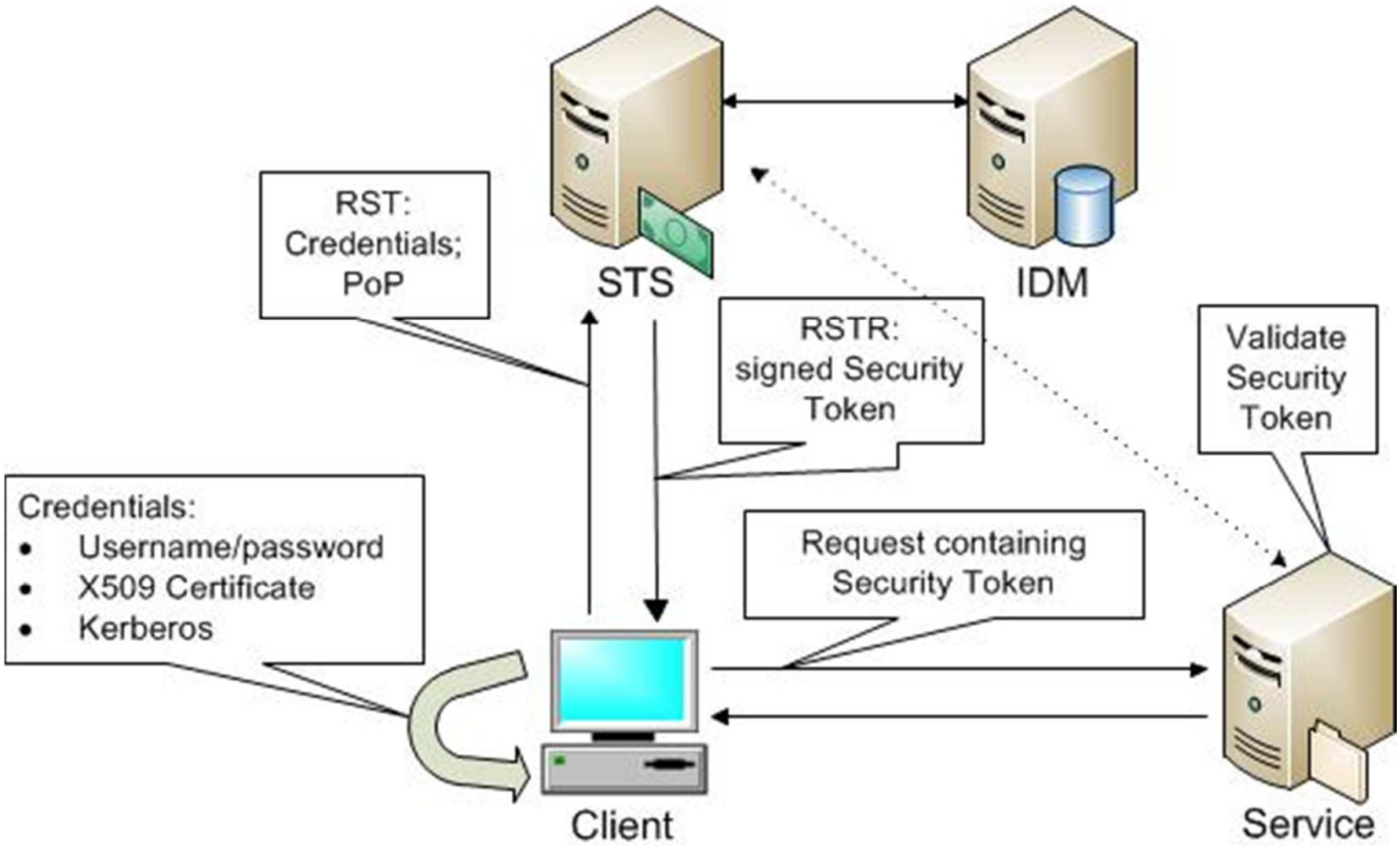
Headers: {Accept=[application/xml], Cache-Control=[no-cache], connection=[keep-alive]

Payload: name=CXF&id=125&SAMLToken=eJydV1tzqkgQfs+vsDiPWcNFjWIdUzUIGqJ

Use Case for WS-Trust



WS-Trust: Security Token Service



JSON Web Token

Header	{"typ":"JWT", "alg":"HS256"}
Claims	{"iss":"cxf:oauth", "sub":"userA" "exp":1300819380, "iat":1300819370, "nbf":1300800000, "aud":"api.mydomain.org" "mycustomclaim":"roleA"}

Header	eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9 .
Claims	eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt cGxlLmNvbS9pc19yb290Ijp0cnVlfQ .
Signature	dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk

Signature = HMACSHA256(BASE64URL(UTF8(JWT Header)) + ’.’ + BASE64URL(JWT Claims), key)

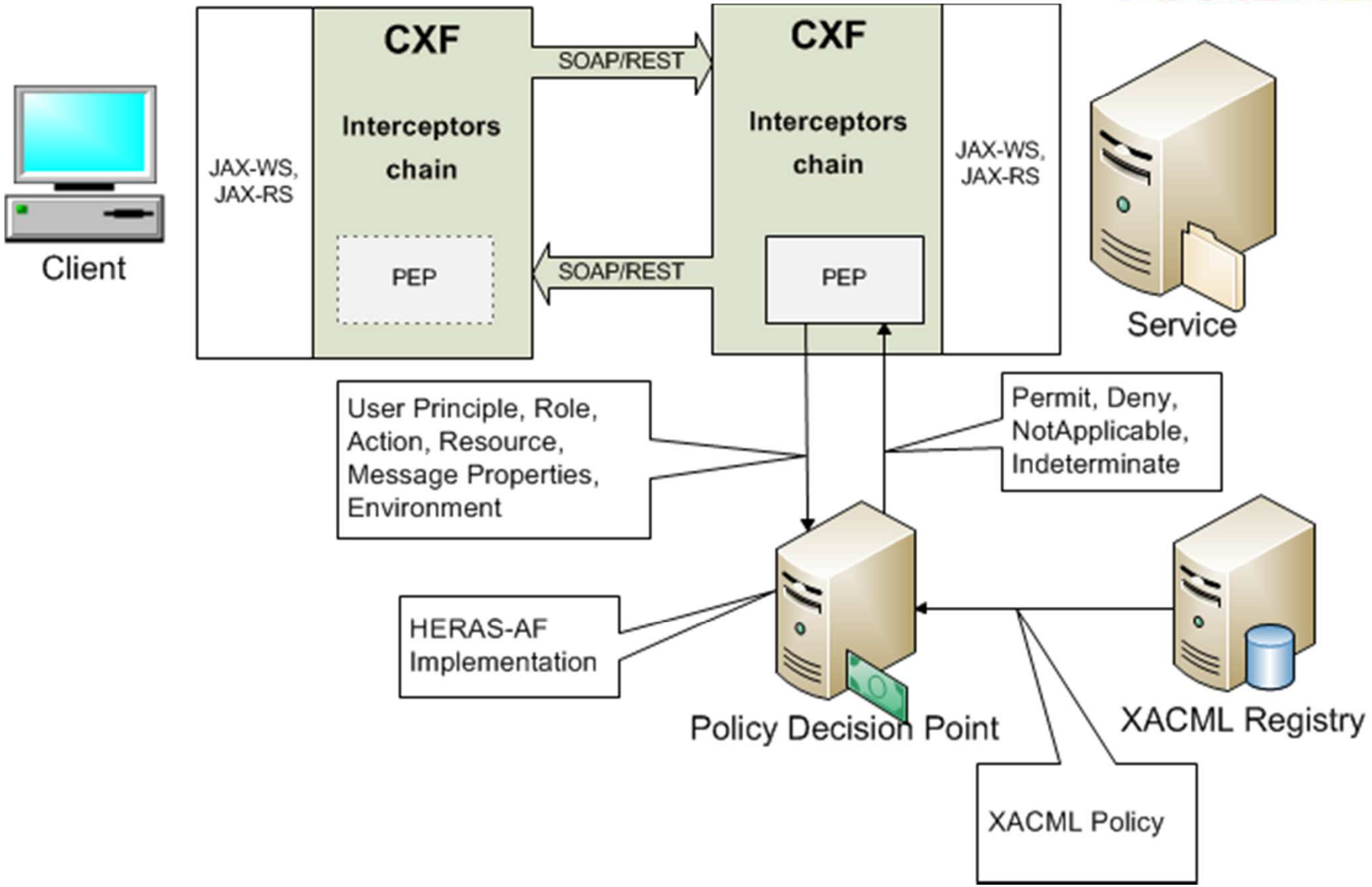
Choose Authentication Method

- What kind of credentials users will provide for the authentication (passwords, smart cards, public/private keys)?
- Which authentication methods are supported by existing infrastructure?
- Should you support Single sign-On?
- Is it necessary to associate additional data with user principals?
- Do you need to communicate with external services?

Authorization

- Container based
- Simple (mapping user -> role, method -> role)
- Annotation based
- XACML
- OAuth

XACML


CXF
Interceptors chain
JAX-WS, JAX-RS
PEP
SOAP/REST
SOAP/REST
CXF
Interceptors chain
JAX-WS, JAX-RS
PEP
Client
Service
User Principle, Role, Action, Resource, Message Properties, Environment
Permit, Deny, NotApplicable, Indeterminate
HERAS-AF Implementation
Policy Decision Point
XACML Registry
XACML Policy

OAuth 2.0

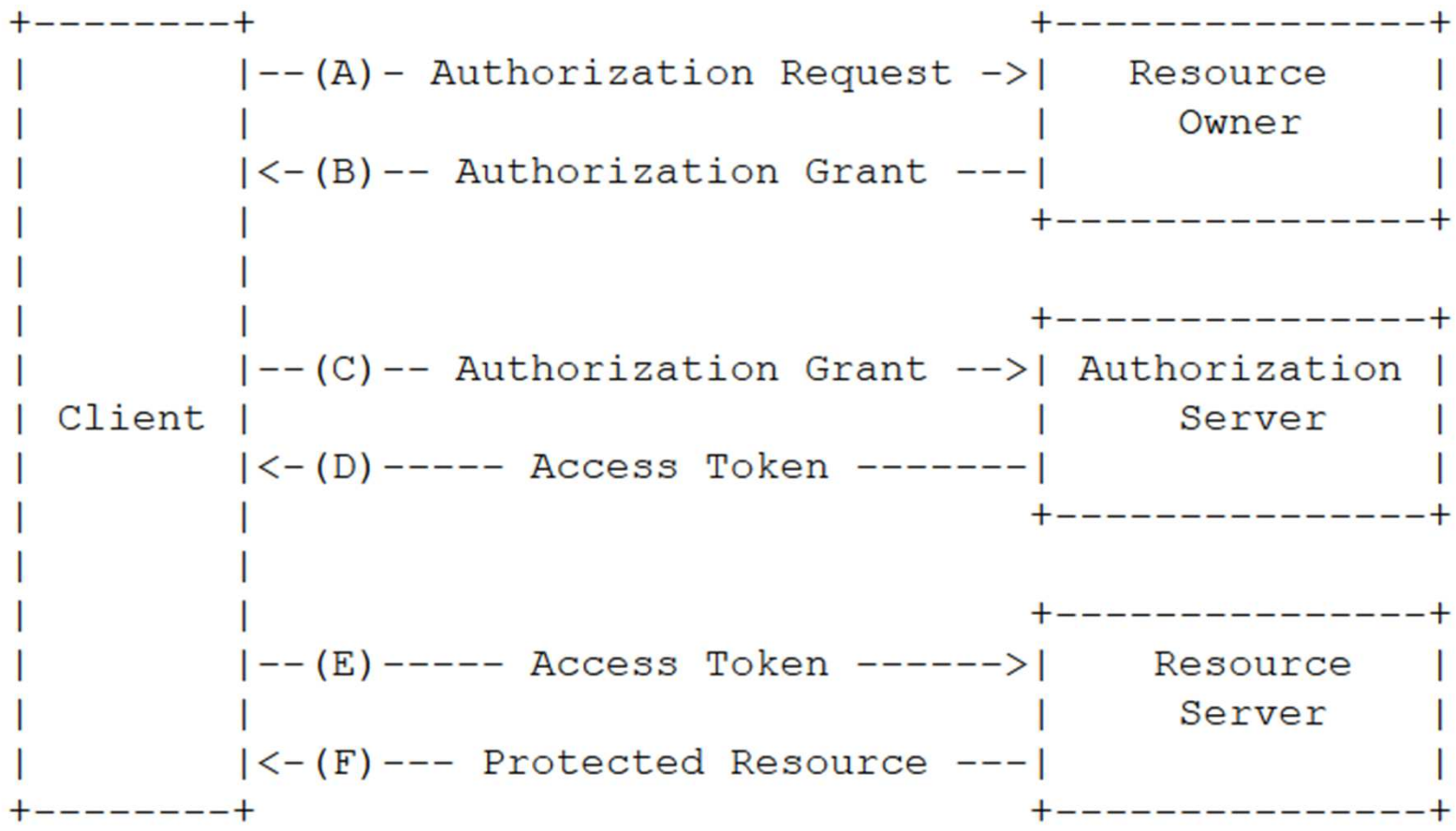
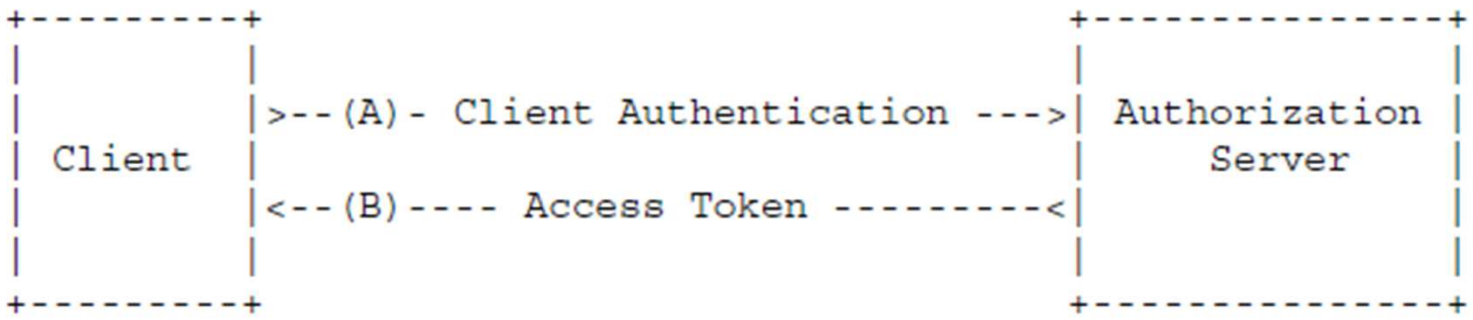


Diagram from OAuth 2.0 spec

OAuth 2.0


Client
(A) Client Authentication
Authorization Server
(B) Access Token

Choose Authorization Method

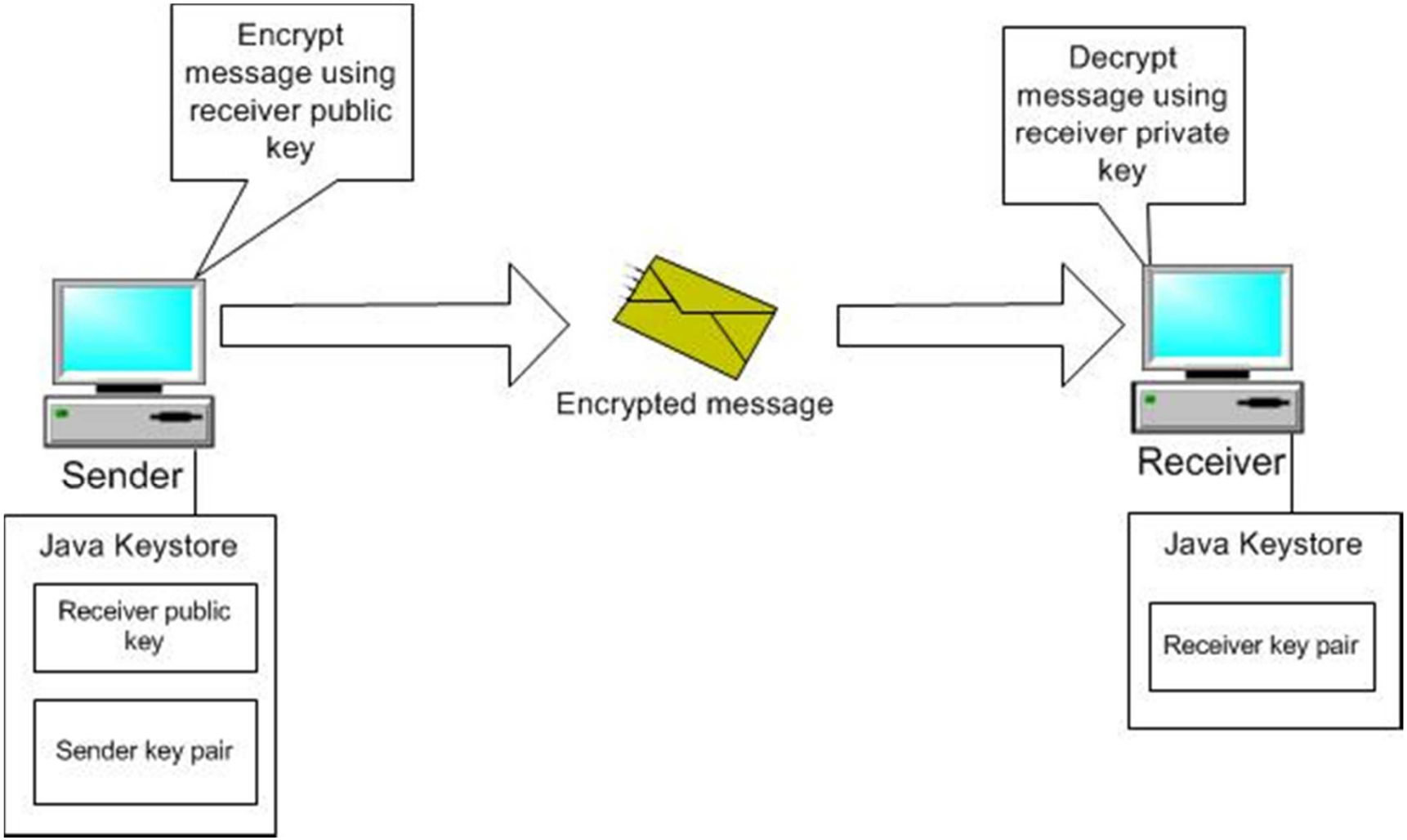
- What kind of resources should be protected?
- Does your Web / Application Server provide authorization?
- What information is required to make authorization decision?
- Do you need to provide limited access to resources for the third parties?

Message Level Security: XML

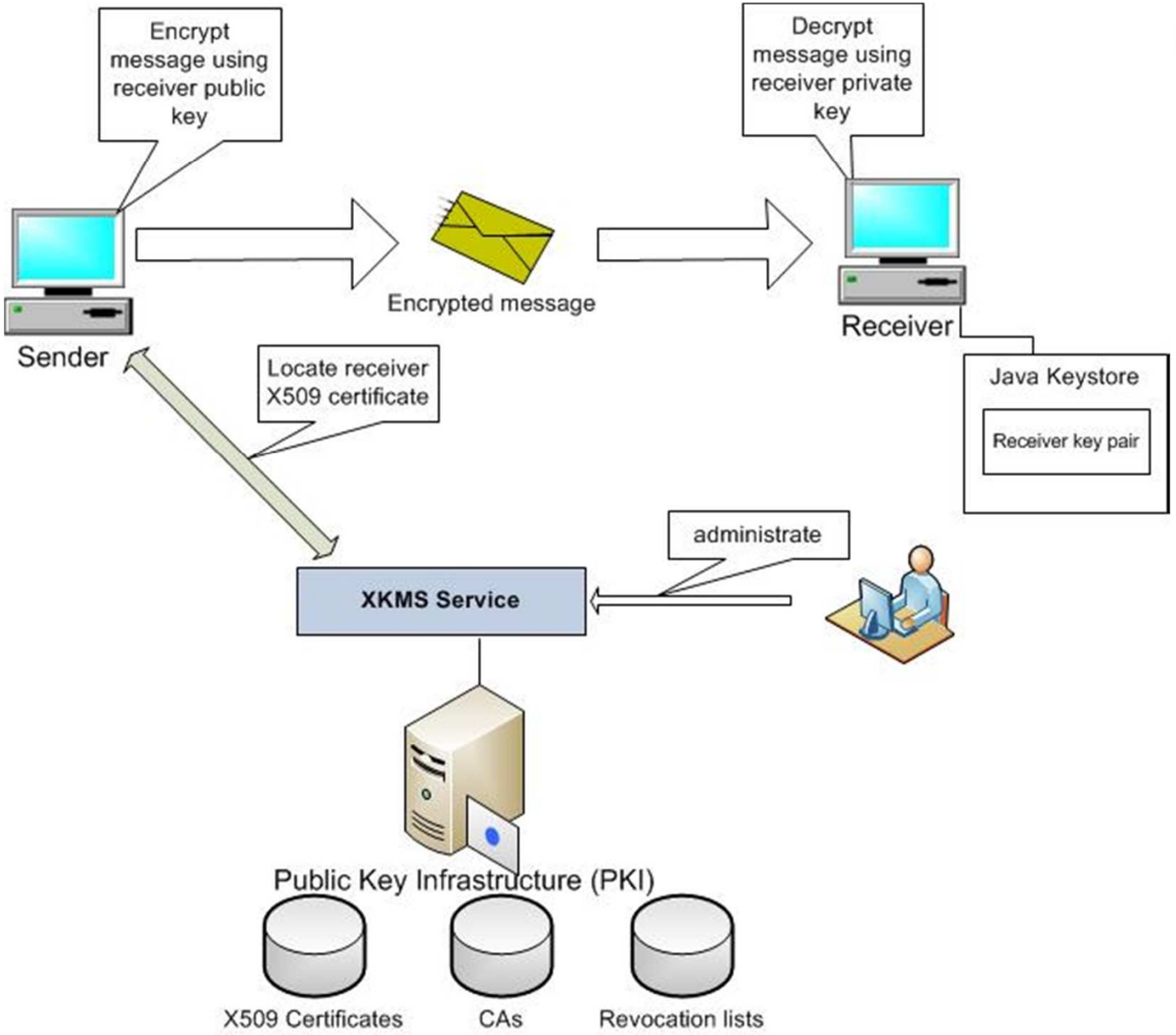
XML Signature and Encryption

- JAX-WS: WS-Policy, WSS4J
- JAX-RS: Enveloped, Enveloping and Detached Signatures, Encryption

XKMS Use Case



XML Key Management Service



JSON Web Signature

- JwsJsonWriterInterceptor;
- JwsJsonClientResponseFilter and JwsJsonContainerRequestFilter

```
Header:
{"alg":"HS256","cty":"json"}

Payload:
{"Book":{"id":123,"name":"book"}}
```

```
JWS JSON serialization:
 {"payload":
        "eyJpc3MiOiJqb2UiLA0KICJleHAiLy9leGF
         tcGxlLmNvbS9pc19yb290Ijp0cnVlfQ",
       "signatures":[
        {"protected":"eyJhbGciOiJSUzI1NiJ9",
         "header":
          {"kid":"2010-12-29"},
         "signature":
          "cC4hiUPoj9EetdgQGe77Rw ..."},
        {"protected":"eyJhbGciOiJFUzI1NiJ9",
         "header":
          {"kid":"e9bc097a-ce51-4036-9562-d2ade882db0d"},
         "signature":
          "DtEhU3ljbEg8L38VWAfUAqO-Kg6NU1Q ..."}]
}
```

```
Compact JWS:
 eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
 .
 eyJpc3MiOiJqb2UiLA0KICJleHAmh0dHA6Ly9leGFt
 cGxlLmNvbS9pc19yb290Ijp0cnVlfQ
 .
 dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

JSON Web Encryption

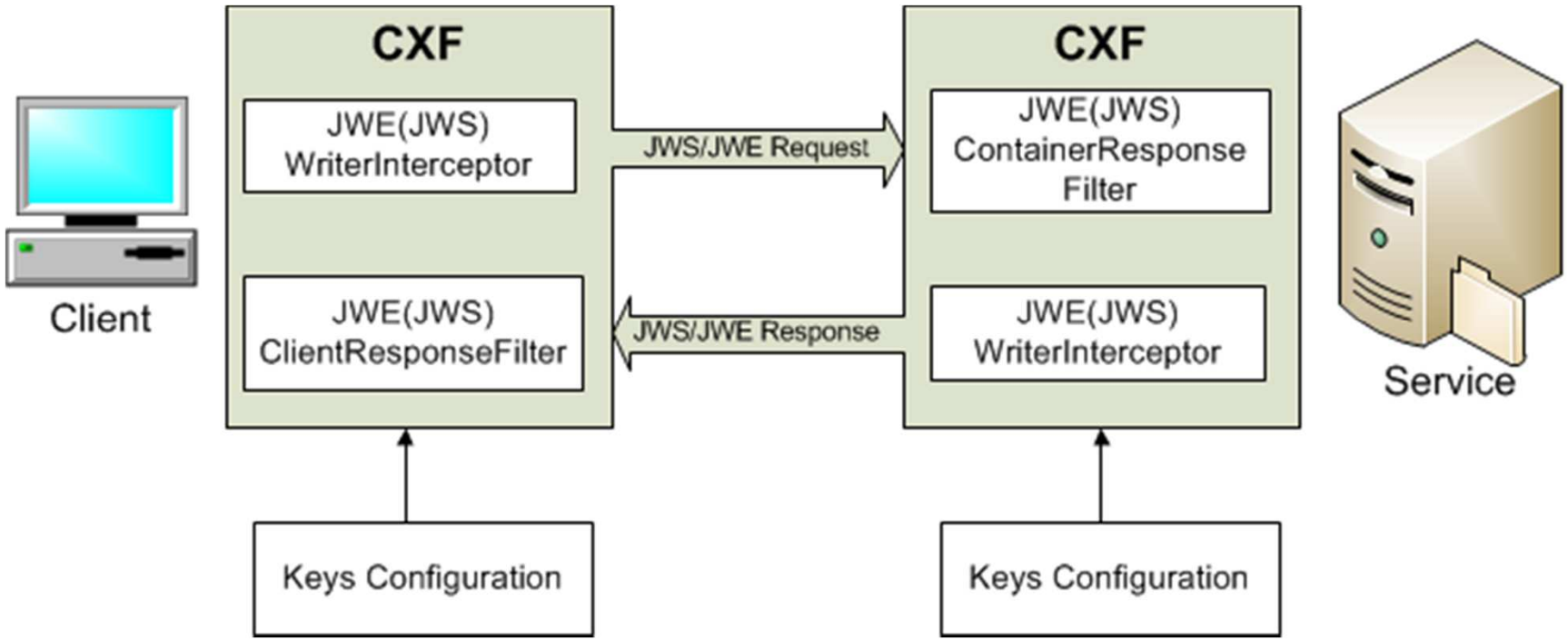
- JweWriterInterceptor;
- JweClientResponseFilter and JweContainerRequestFilter

```
Header:
{"alg":"RSA-OAEP",
 "enc":"A256GCM"}
```

```
Encrypted structure:
BASE64URL(UTF8(JWE Protected Header)) + '.'
+ BASE64URL(JWE Encrypted Key) + '.'
+ BASE64URL(JWEInitialization Vector) + '.'
+ BASE64URL(JWE Ciphertext) + '.'
+ BASE64URL(JWE Authentication Tag)
```

```
eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00ifQ.
OKOawDo13gRp2ojaHV7LFpZcgV7T6DVZKTyKOMTYUmKoTCVJRgckCL9kiMT03JGe
ipsEdY3mx_etLbbWSrFr05kLzcSr4qKAq7YN7e9jwQRb23nfa6c9d-StnImGyFDb
Sv04uVuxIp5Zms1gNxKKK2Da14B8S4rzVRltdYwam_lDp5XnZAYpQdb76FdIKLaV
mqgfwX7XWRxv2322i-vDxRfqNzo_tETKzpVLzfiwQyeyPGLBIO56YJ7eObdv0je8
1860ppamavo35UgoRdbYaBcoh9QcfylQr66oc6vFWXRcZ_ZT2LawVCWTIy3brGPi
6UklfCpIMfIjf7iGdXKHzg.
48V1_ALb6US04U3b.
5eym8TW_c8SuK0ltJ3rpYIzOeDQz7TALvtu6UG9oMo4vpzs9tX_EFShS8iB7j6ji
SdiwkIr3ajwQzaBtQD_A.
XFBoMYUZodetZdvTiFvSkQ
```

JWS/JWE in CXF



Conclusion

1. CXF provides a wide range of security solutions: from very simple to really complicated
2. Choice of the security features for your services is based on use case, requirements and existing infrastructre
3. Follow standards and prefer established solutions to secure your services
4. Check security issues for using frameworks
5. Apply negative security tests as part of your system or integration tests

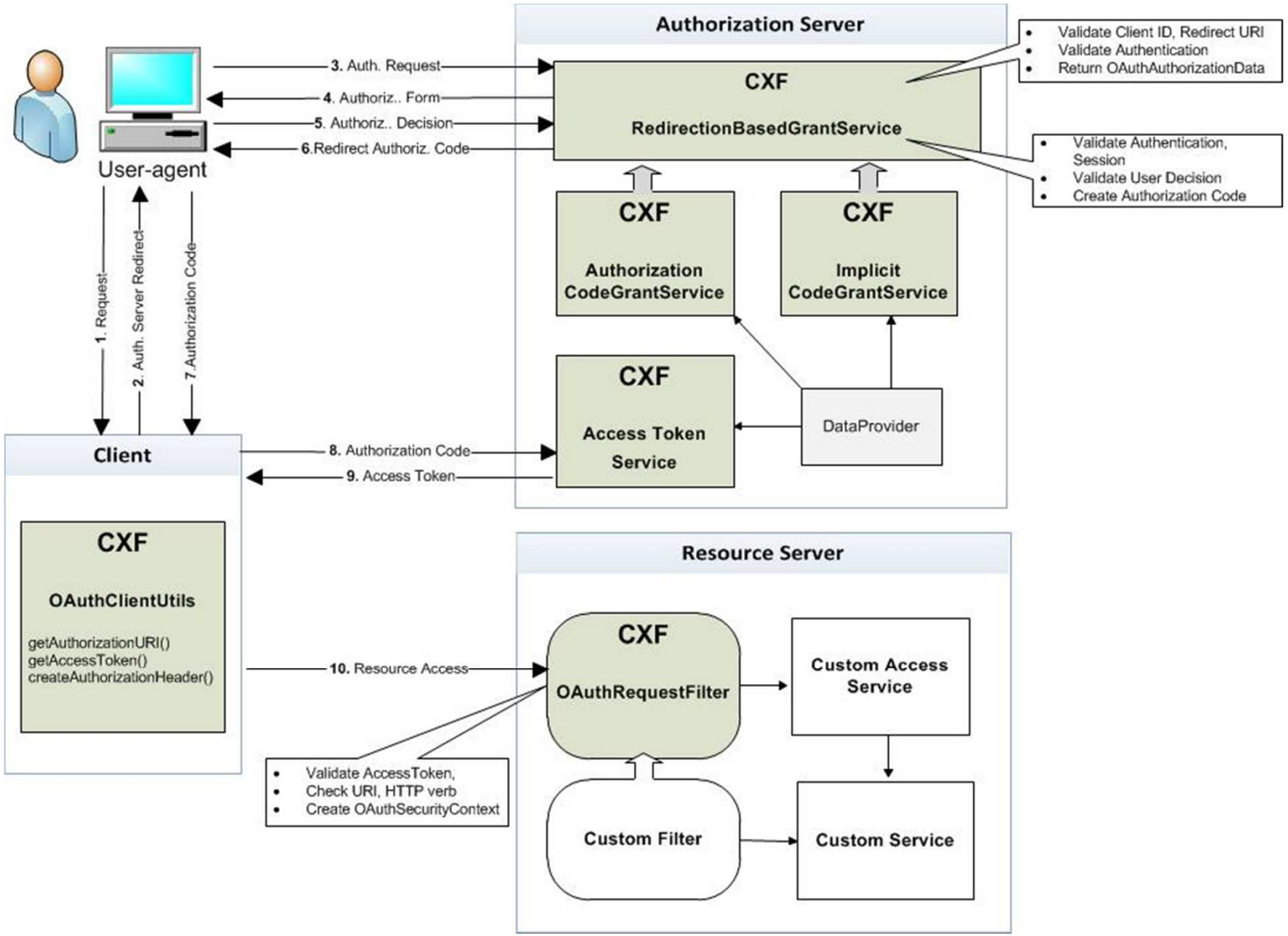
Links

- CXF Rest Services security:
  http://cxf.apache.org/docs/secure-jax-rs-services.html

- CXF Soap Services security:
  http://cxf.apache.org/docs/ws-security.html
  http://cxf.apache.org/docs/ws-trust.html
  http://cxf.apache.org/docs/ws-securitypolicy.html

- Blogs:
  http://ashakirin-cxf-security.blogspot.de/
  http://coheigea.blogspot.de/
  http://sberyozkin.blogspot.com

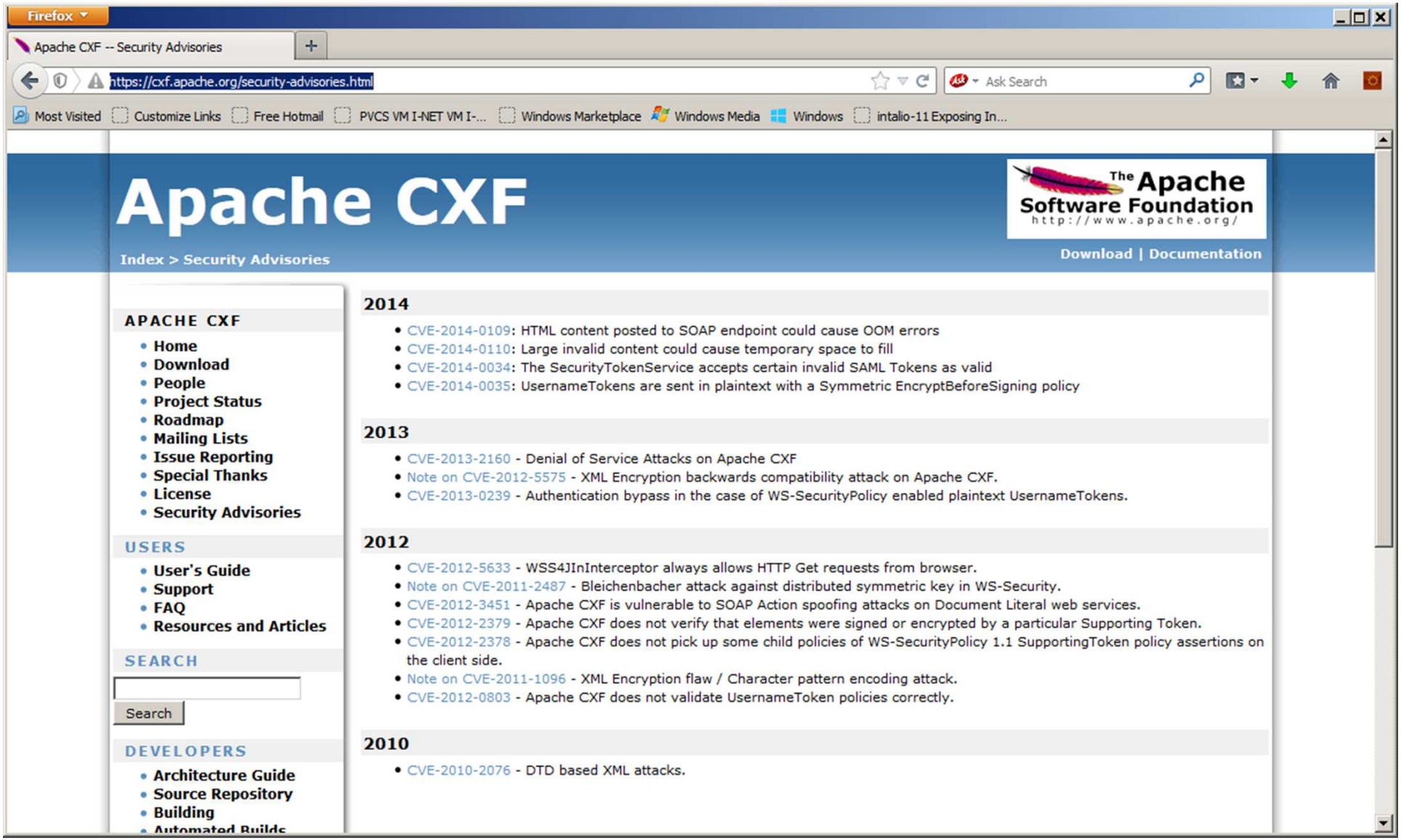
OAuth 2.0 in CXF



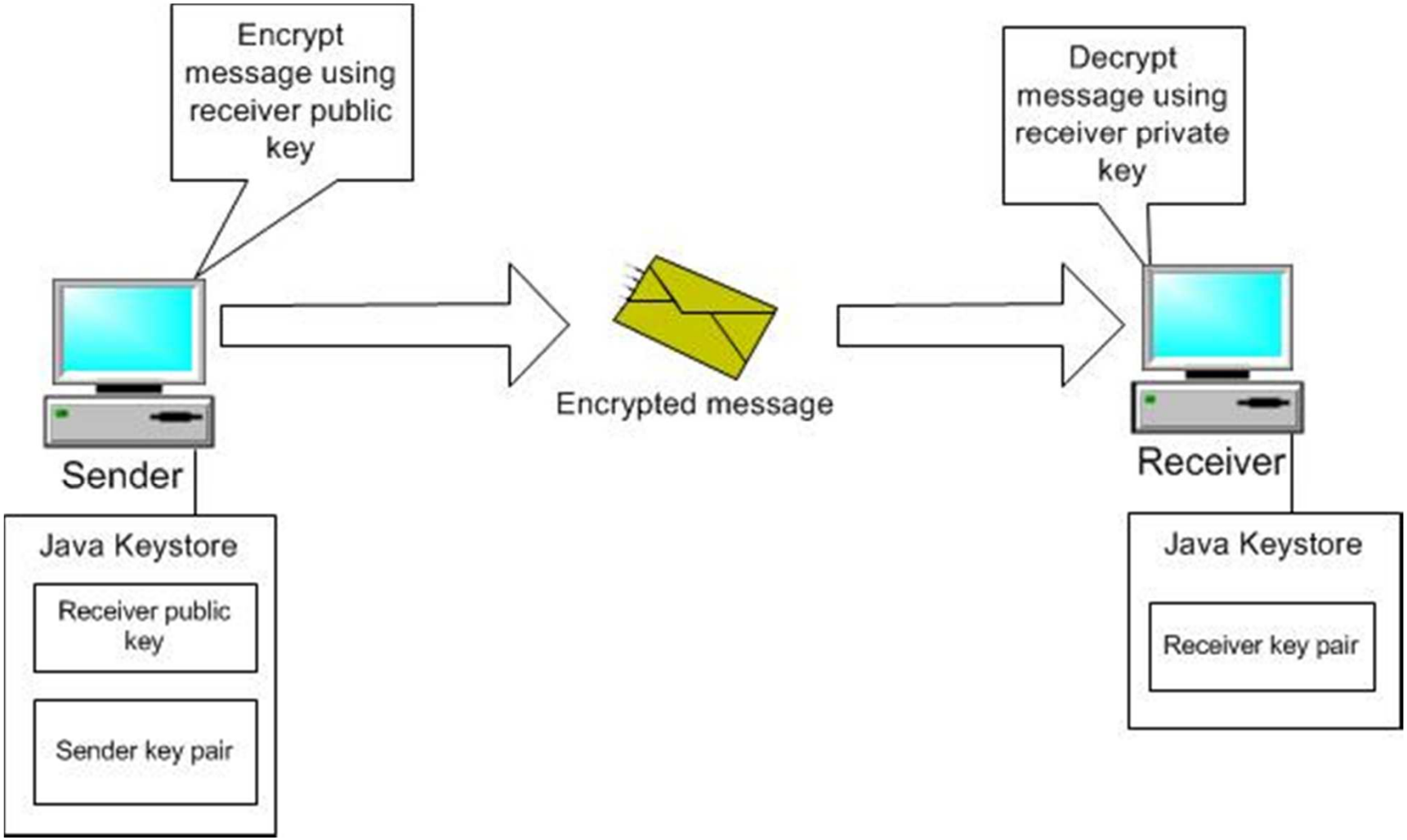
Recommendations

- Don't implement your own security
- Analyse and evaluate possible attacks
- Apply negative security tests
- Use black box testing tools: WS-Attacker (Ruhr-University Bochum, sourceforge)
- Check certificates, passwords policy, using algorithms and keys for low-level libraries and framewroks
- Default settings must be secure
- Check security issues for using frameworks

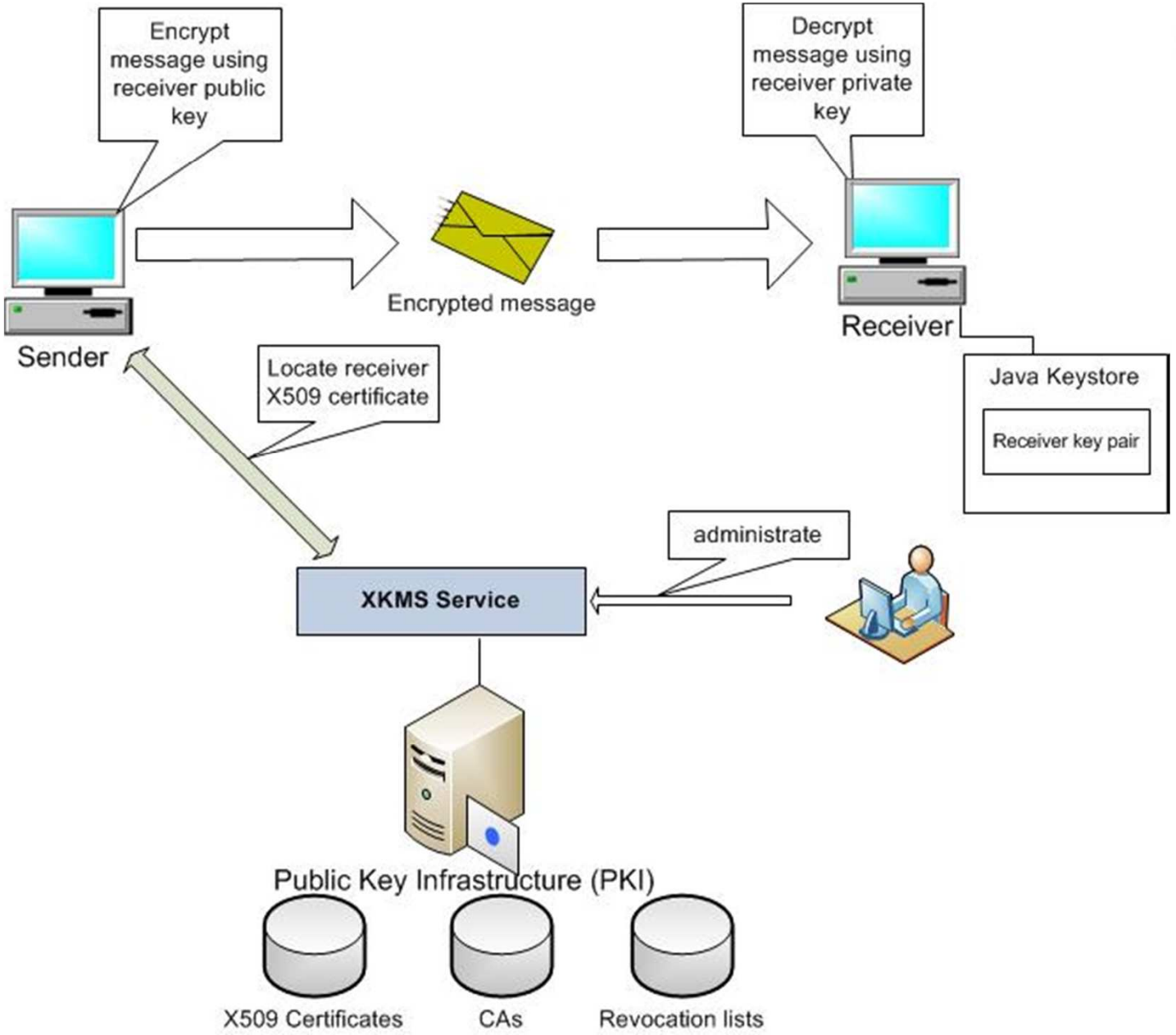
CXF Security Advisories



XKMS Use Case



XML Key Management Service



Attacks and Vulnerabilities

- Replay
- Injection (XPath, XML)
- Wrapping
- Spoofing (SOAPAction, WS-Addressing)
- XML DOS (Oversized XML, XML Bomb)
- Cross-site scripting (XSS: client side script injection)
- Cross-site request forgery (CSRF)

Signature Wrapping Attack


```
<soap:Envelope .>
  <soap:Header>
    <wsse:Security>
      <ds:Signature>
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
            Algorithm=".../xml-exc-c14n#" />
          <ds:SignatureMethod Algorithm="...#rsa-sha1" />
          <ds:Reference URI="#theBody">
            <ds:Transforms>
              <ds:Transform Algorithm=".../xml-exc-c14n#" />
            </ds:Transforms>
            <ds:DigestMethod Algorithm=".../xmldsig#sha1" />
            <ds:DigestValue>AbCdEfG0123456789...  </ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>AbCdEfG0123456789...  </ds:SignatureValue>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#X509Token" />
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </soap:Header>
  <soap:Body wsu:Id="theBody">
    <getQuote Symbol=”IBM” />
  </soap:Body>
</soap:Envelope>
```

Signature Wrapping Attack


```
<soap:Envelope .>
  <soap:Header>
    <wsse:Security>
      <ds:Signature>
        <ds:SignedInfo>
          <ds:Reference URI="#theBody">
            <ds:Transforms>
              <ds:Transform Algorithm=".../xml-exc-c14n#" />
            </ds:Transforms>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>AbCdEfG0123456789...</ds:SignatureValue>
       </ds:Signature>
    </wsse:Security>
    <!-- ... -->
    <!-- Original SOAP Body is placed below -->
    <!--.... -->
    <wrapper>
      <soap:Body wsu:Id="theBody">
        <getQuote Symbol=”IBM” />
      </soap:Body>
    </wrapper>
  </soap:Header>
  <!-- ... -->
  <!-- Maliciously modified SOAP Body is placed below -->
  <!--.... -->
  <soap:Body wsu:Id="#theBody">
    <getQuote Symbol=”Evil” />
  </soap:Body>
</soap:Envelope>
```

Configure SSL in CXF

Client:

```
WebClient client = WebClient.create(address, configLocation);
CustomerService proxy = JAXRSClientFactory.create(address,
        CustomerService.class, configLocation);
```

Service:

```
JAXRSServerFactoryBean bean = new JAXRSServerFactoryBean();
SpringBusFactory bf = new SpringBusFactory();
Bus bus = bf.createBus(configLocation);
bean.setBus(bus);
bean.setAddress(address);
bean.setServiceClass(CustomerService.class);
```

JWT

JSON: JWS, JWE, JWT, JOSE

- Possible with external libraries Jose4J, JsonCrypto
- Tight integration is in the pipe line

```
Payload:
{"iss":"joe",
 "exp":1300819380,
 "http://example.com/is_root":true}
```

```
Header:
{"typ":"JOSE",
 "alg":"HS256"
 "kid":"e9bc097a-ce51-4036-9562-d2ade882db0d"}
```

```
JWS JSON serialization:
 {"payload":
       "eyJpc3MiOiJqb2UiLA0KICJleHAiLy9leGF
        tcGxlLmNvbS9pc19yb290Ijp0cnVlfQ",
      "signatures":[
       {"protected":"eyJhbGciOiJSUzI1NiJ9",
        "header":
         {"kid":"2010-12-29"},
        "signature":
         "cC4hiUPoj9EetdgQGe77Rw ..."},
       {"protected":"eyJhbGciOiJFUzI1NiJ9",
        "header":
         {"kid":"e9bc097a-ce51-4036-9562-d2ade882db0d"},
        "signature":
         "DtEhU3ljbEg8L38VWAfUAqO-Kg6NU1Q ..."}]
}
```

```
Compact JWS:
 eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
 .
 eyJpc3MiOiJqb2UiLA0KICJleHAmh0dHA6Ly9leGFt
 cGxlLmNvbS9pc19yb290Ijp0cnVlfQ
 .
 dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```