IBM z/OS Connect Enterprise Edition

Security

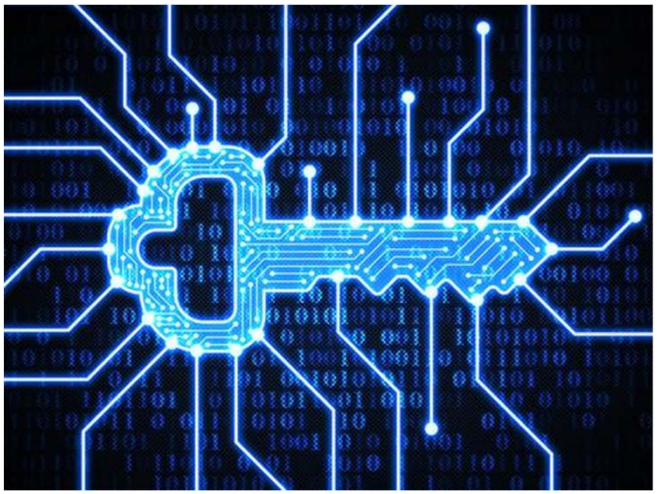
Mitch Johnson
mitchj@us.ibm.com
Washington System Center

Contents

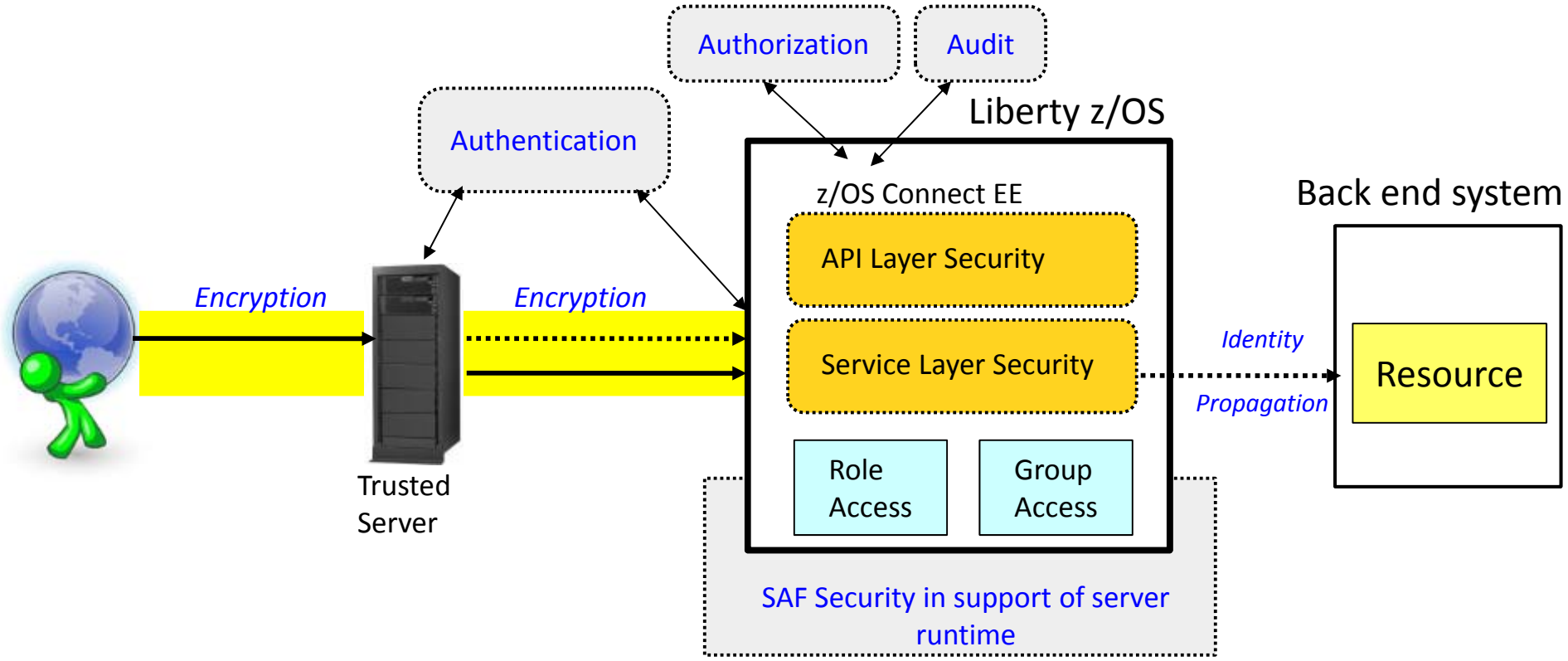
- Introduction
- API provider security
  - Authentication
  - Authorization
  - Audit
  - Encryption
  - Flowing identities to back end systems
- API requester security
  - What's different?
- More information

General considerations for securing REST APIs

- Know who is invoking the API (**Authentication)**
- Ensure that the data has not been altered in transit (**Data Integrity)** and ensure confidentiality of data in transit (**Encryption)**
- Control access to APIs (**Authorization)**
  - End user
  - Application
- Know who invoked the APIs (**Audit**)



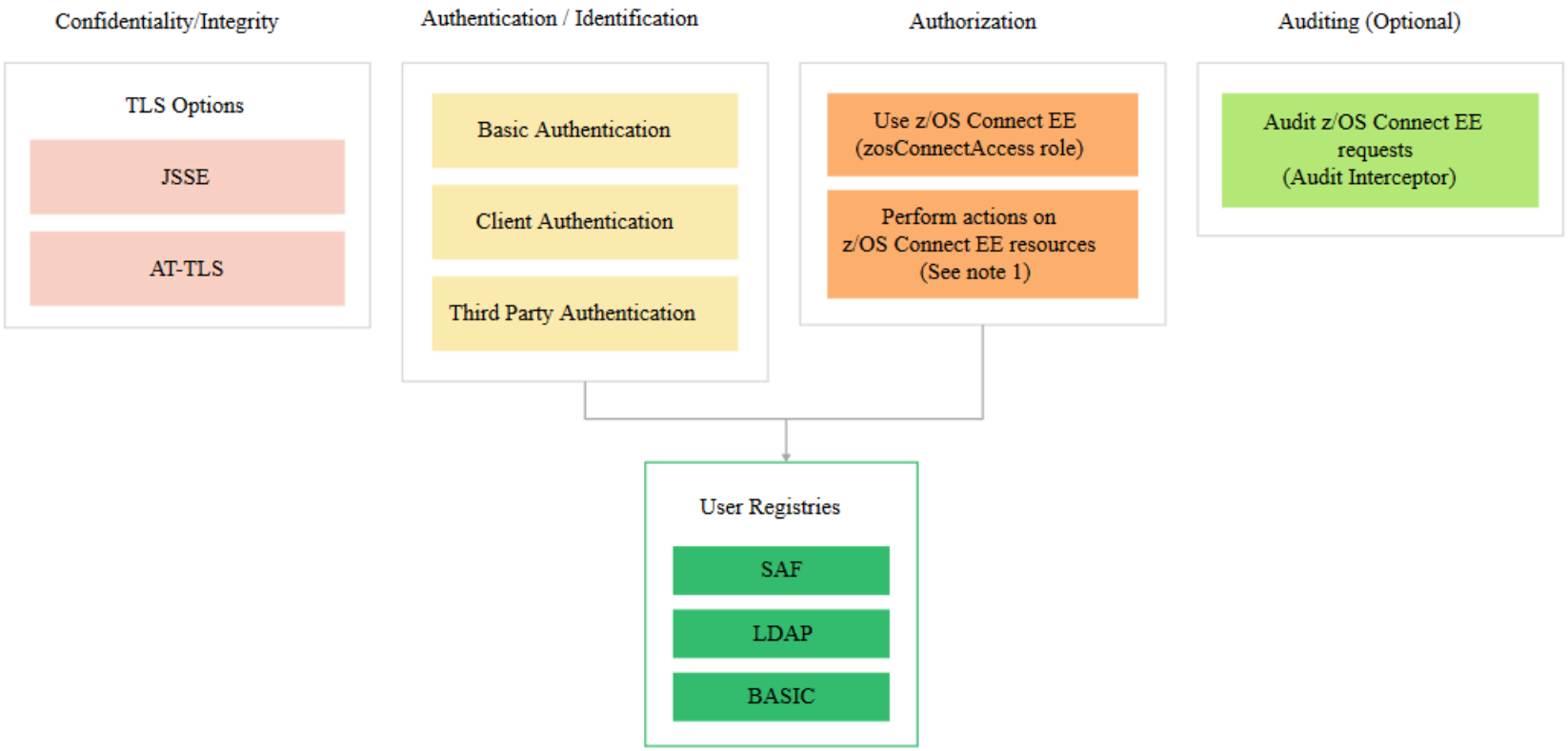
z/OS Connect EE API provider security overview



1. Authentication (basic, client certificates, 3rd party authentication)
2. Encryption (aka "SSL" or "TLS")
3. Authorization (role and group access)
4. Audit
5. Configuring security with SAF
6. Back end identity propagation (CICS, IMS, D)

See Dev Center article "Securing APIs with z/OS Connect EE" overview of z/OS Connect EE security

z/OS Connect EE security options

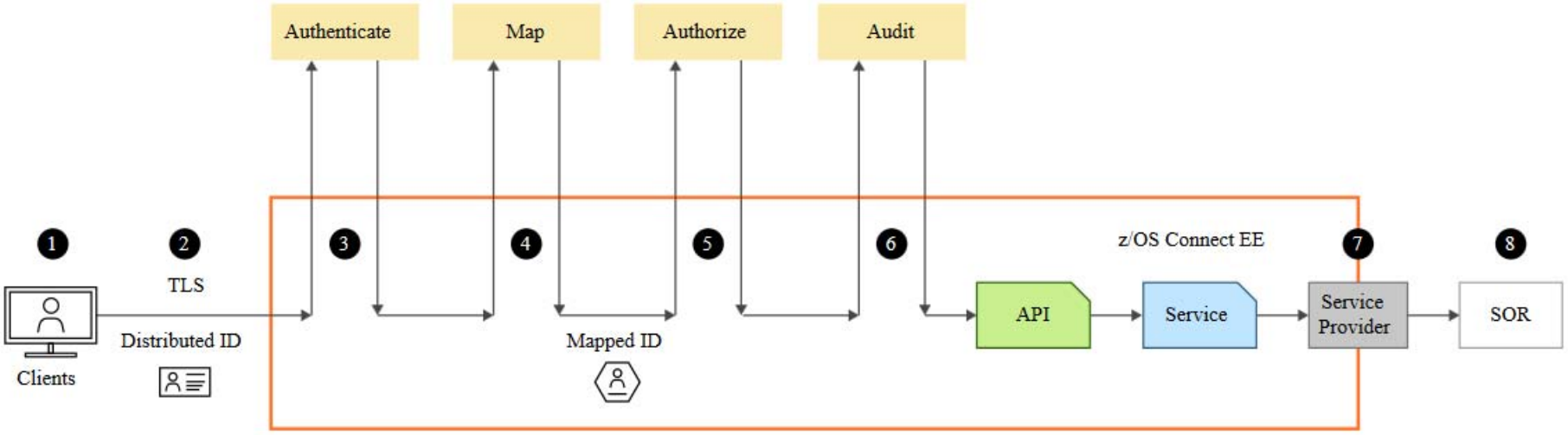


http://ibm.biz/zosconnect-security

The actions which can be controlled by authorization (see Note 1 in the diagram above) are: deploying, querying, updating, starting, stopping and deleting of APIs, services and API requesters.

Typical z/OS Connect EE security flow

z/OS Connect EE



1. The credentials provided by the client
2. Secure the connection to the z/OS Connect EE server
3. Authenticate the client. This can be within the z/OS Connect EE server or by requesting verification from a third party server
4. Map the authenticated identity to a user ID in the user registry
5. Authorize the mapped user ID to connect to z/OS Connect EE and optionally authorize user to invoke actions on APIs
6. Audit the API request
7. Secure the connection to the System of Record (SoR) and provide security credentials to be used to invoke the program or to access the data resource
8. The program or database request may run in the SoR under the mapped ID

Security is configured in server.xml

z/OS Connect EE

Excerpt from server.xml

```
<featureManager>
     <feature>appSecurity-2.0</feature>
     <feature>zosSecurity-1.0</feature>
     <feature>transportSecurity-1.0</feature>
</featureManager>


<sslDefault sslRef=“defaultSSLConfig"/>
<ssl id=“defaultSSLConfig"
     keyStoreRef=“defaultKeyStore"
     outboundSSLRef=“defaultKeyStore" />


<keyStore id=“defaultKeyStore" fileBased="false"
     location="safkeyring:///Keyring.LIBERTY"
     password=“password" readOnly="true"
     type="JCERACFKS"/>

<webAppSecurity allowFailOverToBasicAuth="true" />

<safRegistry id="saf"/>
<safAuthorization id="safAuth"/>
<safCredentials profilePrefix="BBGZDFLT"
                    unauthenticatedUser="WSGUEST"/>
```

Features

SSL repertoire

Key Store

Authentication

Authorization

Common challenges

- **End-to-end security** is hampered by the issue of how to provide secure access between middleware components that use disparate security technologies e.g. registries
  - This is a driver for implementing open security models like OAuth and OpenID Connect and standard tokens like JWT
- z/OS Connect security is implemented in many products including z/OS Connect, Liberty z/OS, SAF/RACF, CICS, IMS, DB2 …
  - And these are all documented in different places
- Often security is at odds with **performance**, because the most secure techniques often involve the most processing overhead especially if not configured optimally

Authentication

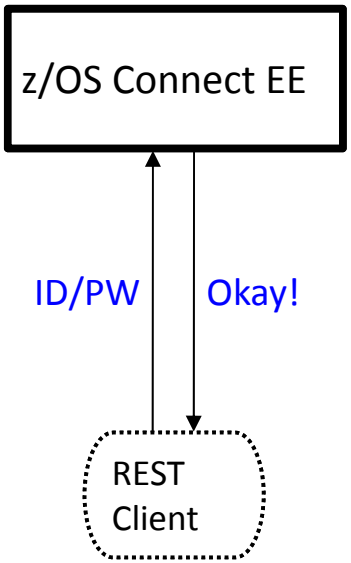
Obtaining an identity

Authentication

z/OS Connect EE

Several different ways this can be accomplished:

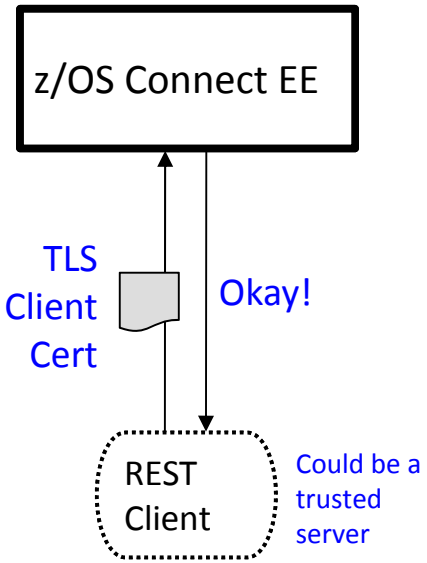
Basic Authentication



Server prompts for ID/PW
Client supplies ID/PW
Server checks registry:

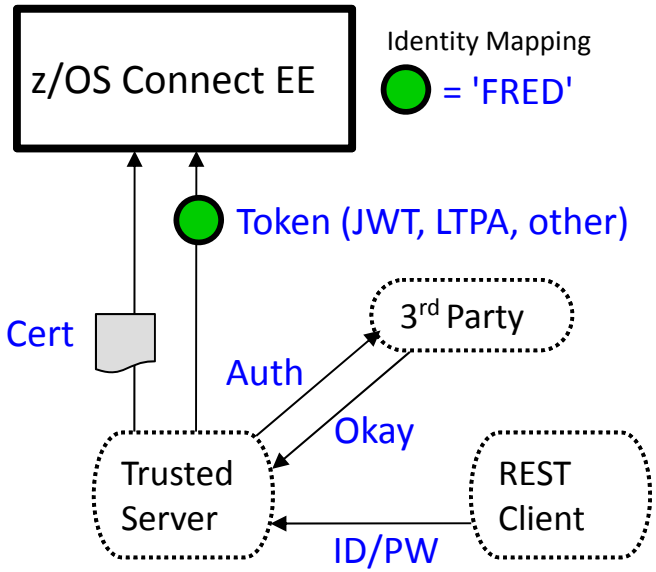
- Basic (server.xml)
- LDAP
- SAF

Client Certificate



Server prompts for cert.
Client supplies certificate
Server validates cert and maps to an identity

Third Party Authentication



Client authenticates to 3rd party sever
Client receives a trusted 3rd party token
Token flows to Liberty z/OS across trusted connection and is mapped to an identity

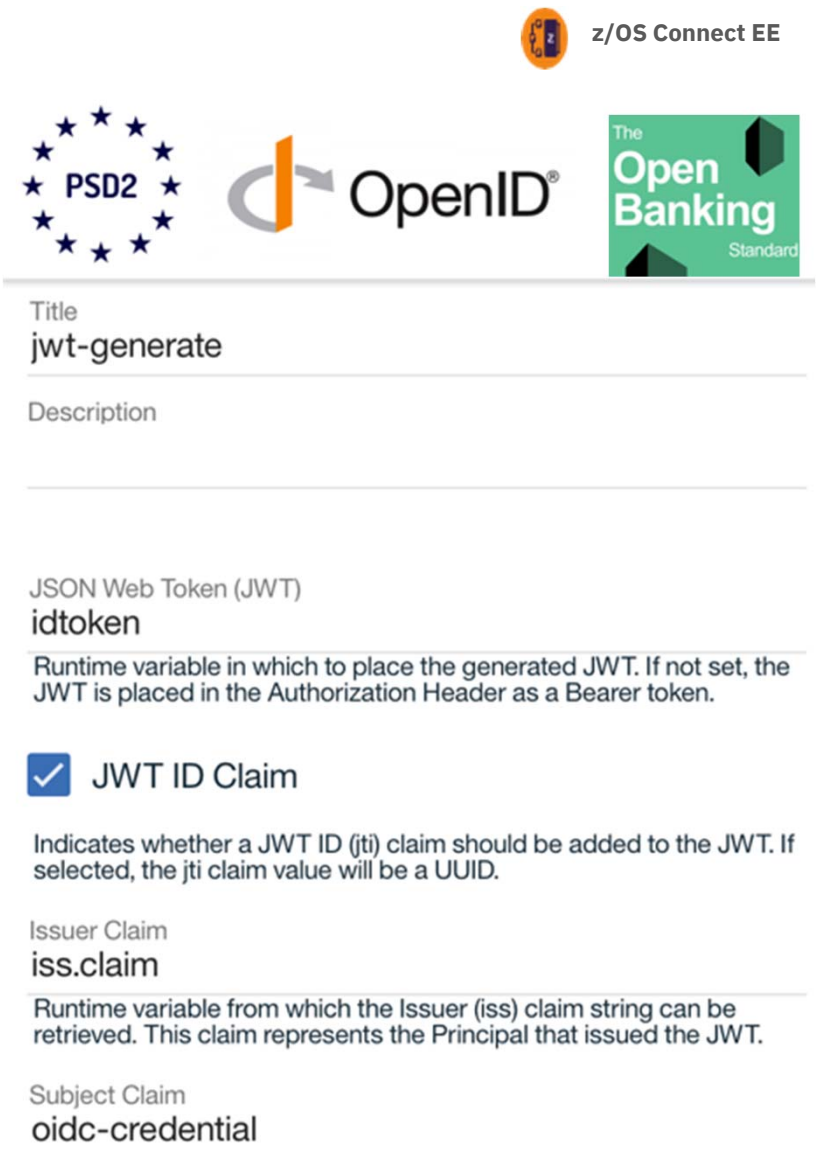
Security token types by z/OS Connect EE

z/OS Connect EE

Token type	How used	Pros	Cons
LTPA	Authentication technology used in IBM WebSphere	• Easy to use with WebSphere and DataPower	• IBM Proprietary token
SAML	XML-based security token and set of profiles	• Token includes user id and claims • Used widely with SoR applications	• Tokens can be heavy to process • No refresh token
OAuth 2.0 access token	Facilitates the authorization of one site to access and use information related to the user's account on another site	• Used widely for SoE applications e.g with Google, Facebook, Microsoft, Twitter …	• Needs introspection endpoint to validate token
JWT	JSON security token format	• More compact than SAML • Ease of client-side processing especially mobile	

OpenID Connect Overview

- **OpenID Connect** (OIDC) is built on top of OAuth 2.0
- Flexible user authentication for Single Sign-On (SSO) to Web, mobile and API workloads
- Addresses European **PSD2** and UK **OpenBanking** requirements for authorization and authentication

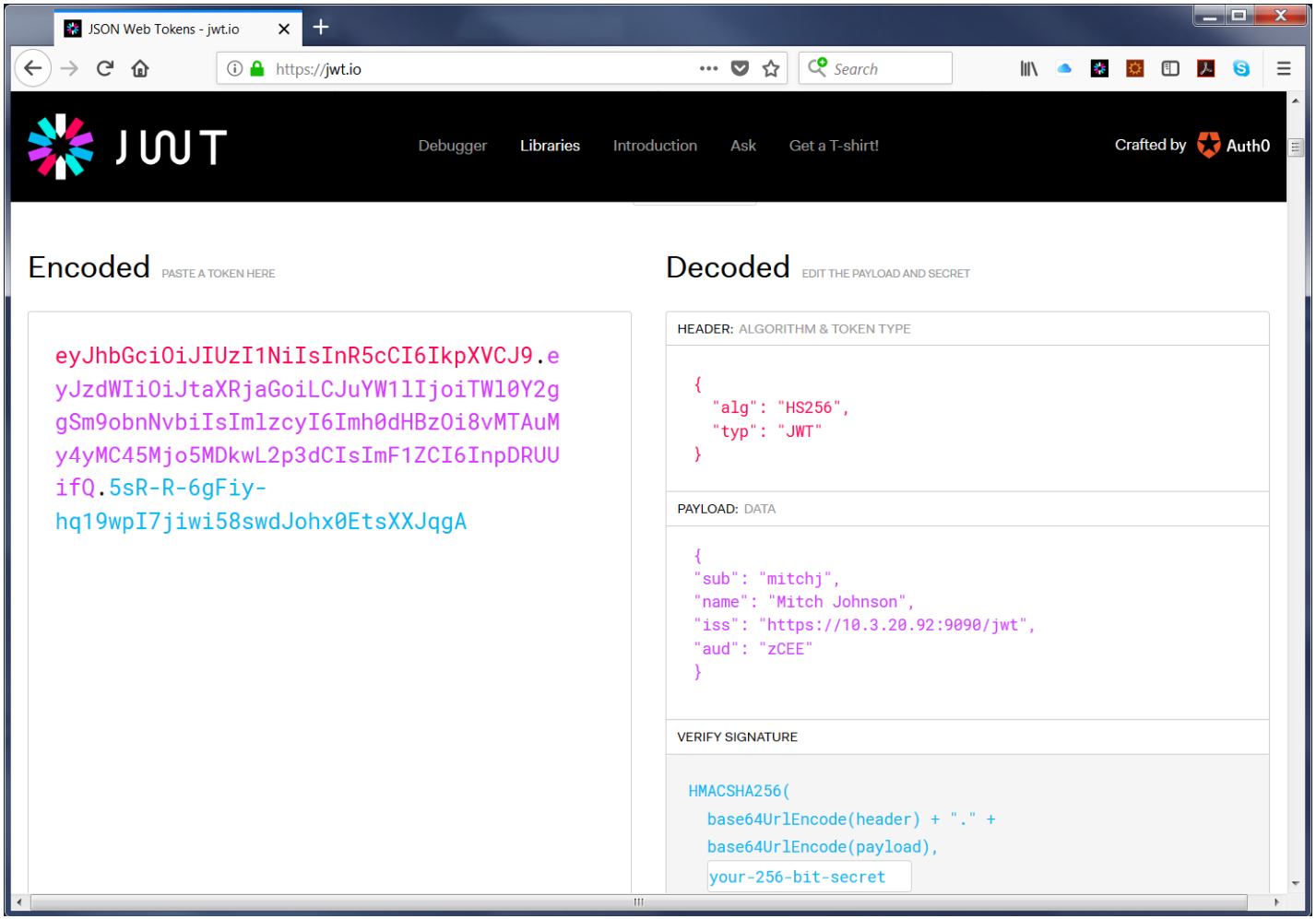


Why JWT with z/OS Connect EE?

- Token validation does **not** require an additional trip and can be validated locally by z/OS Connect server
- Parties can easily agree on a specific set of **custom** claims in order to exchange both authentication and authorization information
- Widely adopted by different Single Sign-On solutions and well known standards such as **OpenID Connect**
- **Message-level** security using signature standard
- JWT tokens are **lighter** weight than other XML based tokens e.g SAML

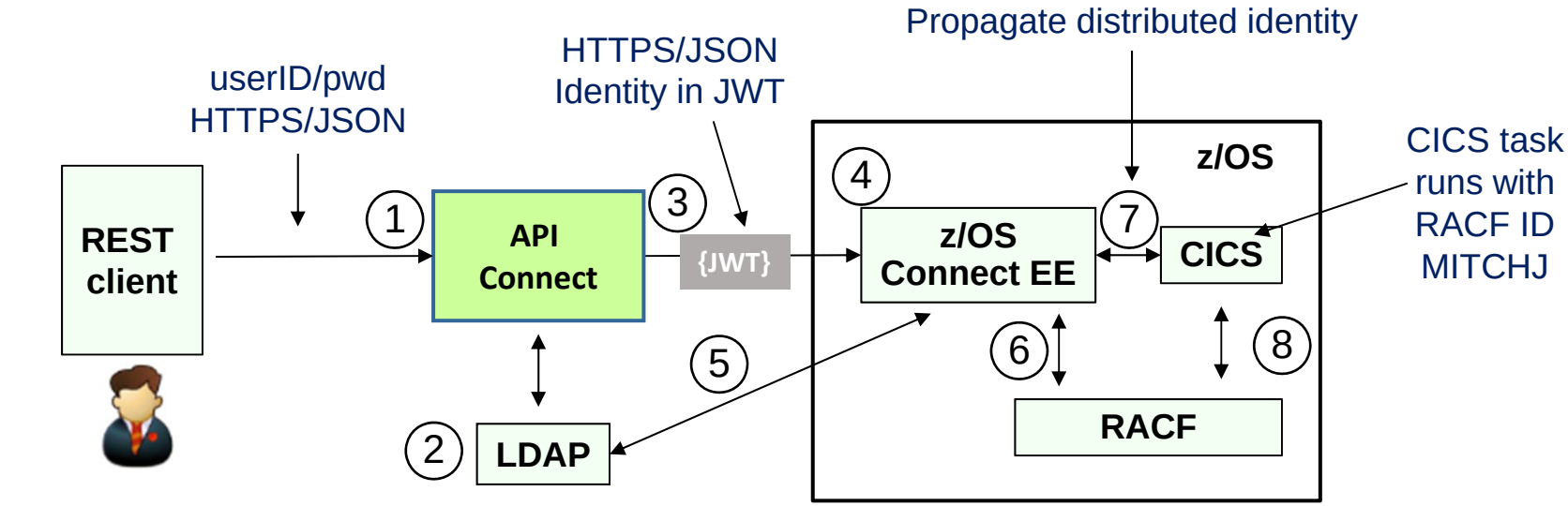
JWT (JSON Web Token)

- JWT is a compact way of representing claims that are to be transferred between two parties
- Normally transmitted via HTTP header
- Consists of three parts
  - Header
  - Payload
  - Signature



Example scenario – security flow

z/OS Connect EE




```
RACMAP ID(MITCHJ)  MAP +
   USERDIDFILTER(NAME('UID=johnson,OU=ats,O=ibm,C=us')) +
   REGISTRY(NAME('*'))
```

1. User authenticates with the managed API using a "distributed" identity and a password
2. LDAP is used as the user registry for distributed users and groups
3. API Connect generates a JWT and forwards the token with the request to z/OS Connect EE
4. z/OS Connect EE validates JWT
5. z/OS Connect EE looks up user in LDAP registry
6. z/OS Connect EE calls RACF to map distributed ID to RACF user ID and authorizes access to API
7. z/OS Connect EE CICS service provider propagates distributed ID to CICS
8. CICS calls RACF to map distributed ID to RACF user ID and performs resource authorization checks

JWT used in scenario


```
{
 "alg": "RS256"
  "typ":  "JWT"

}
{
 "sub": "johnson",
 "iss": "  https://10.3.20.92:9090/jwt",
 "aud": "zCEE"
}
```

- The header contains an **alg** (algorithm) element value **RS256**
  - **RS256** (RSA Signature with SHA-256) is an asymmetric algorithm which uses a **public/private** key pair
  - **ES512** (Elliptic Curve Digital Signature Algorithm with SHA-512) link for more info
  - **HS256** (HMAC with SHA-256) is a symmetric algorithm with only one (**secret**) key
- The **iss** (issuer) claim identifies the principal that issued the JWT
- The **sub** (subject) claim **johnson** identifies the principal that is the subject of the JWT
- The **aud** (audience) claim **zCEE** identifies the recipients that the JWT is intended for (stands for

Configuring authentication with JWT

z/OS Connect EE

z/OS Connect EE can perform user authentication with JWT using the support that is provided by the *openidConnectClient-1.0* feature. The ***<openidConnectClient>*** element is used to accept a JWT token as an authentication token

```
<openidConnectClient id="RS" clientId="RS-JWT-ZCEE" inboundPropagation="required"
    signatureAlgorithm="RS256" trustStoreRef="JWTTrustStore"
    trustAliasName="JWTapicSign" userIdentityToCreateSubject="sub"
    mapIdentityToRegistryUser="true"
    issuerIdentifier="https://10.3.20.92:9090/jwt" authnSessionDisabled="true"
    audiences="zCEE"/>
```

- ***inboundPropagation*** is set to required to allow z/OS Connect EE to use the received JWT as an authentication token
- ***signatureAlgorithm*** specifies the algorithm to be used to verify the JWT signature
- ***trustStoreRef*** specifies the name of the keystore element that defines the location of the validating certificate
- ***trustAliasName*** gives the alias or label of the certificate to be used for signature validation
- ***userIdentityToCreateSubject*** indicates the claim to use to create the user subject
- ***mapIdentityToRegistryUser*** indicates whether to map the retrieved identity to the registry user
- ***issuerIdentifier*** defines the expected issuer
- ***authnSessionDisabled*** indicates whether a WebSphere custom cookie should be generated for the session
- ***audiences*** defines a list of target audiences

See Dev Center article "Using a JWT with z/OS Connect EE" for full description of scenario

Using authorization filters with z/OS Connect EE

z/OS Connect EE

Authentication filter can be used to filter criteria that are specified in the **authFilter** element to determine whether certain requests are processed by certain providers, such as OpenID Connect, for authentication.

```
<openidConnectClient id="RS" clientId="RS-JWT-ZCEE" inboundPropagation="required"
signatureAlgorithm="RS256" trustStoreRef="JWTTrustStore"
trustAliasName="JWTapicSign" userIdentityToCreateSubject="sub"
mapIdentityToRegistryUser="true" issuerIdentifier="https://10.3.20.92:9090/jwt"
authnSessionDisabled="true" audiences="zCEE" authFilterRef="API Gateway"/>

<authFilter id="API Gateway">
     <remoteAddress id="ApiAddress" ip="10.7.1.*" matchType="equals"/>
</authFilter>
<authFilter id="PhoneBook">
      <requestUrl id="URL"  urlPattern="/phoneBook/*" matchType="equals"/>
</authFilter>
```

Some alternative filter types

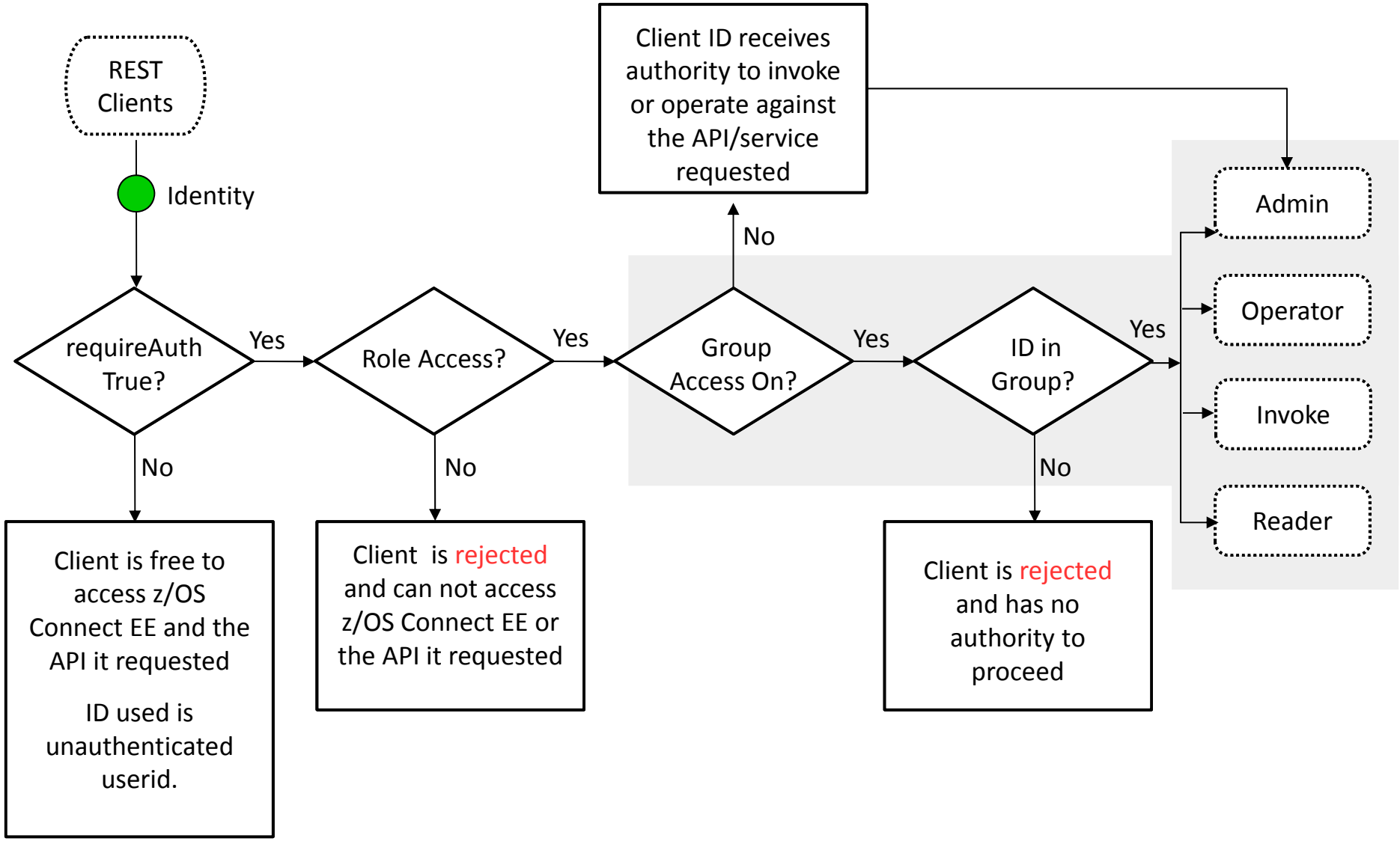
- A ***remoteAddress*** element is compared against the TCP/IP address of the client that sent the request.
- The ***host*** element is compared against the "Host" HTTP request header, which identifies the target host name of the request.
- The ***requestUrl*** element is compared against the URL that is used by the client application to make the request.

Authorization

Once we have an identity

Security flow with z/OS Connect EE

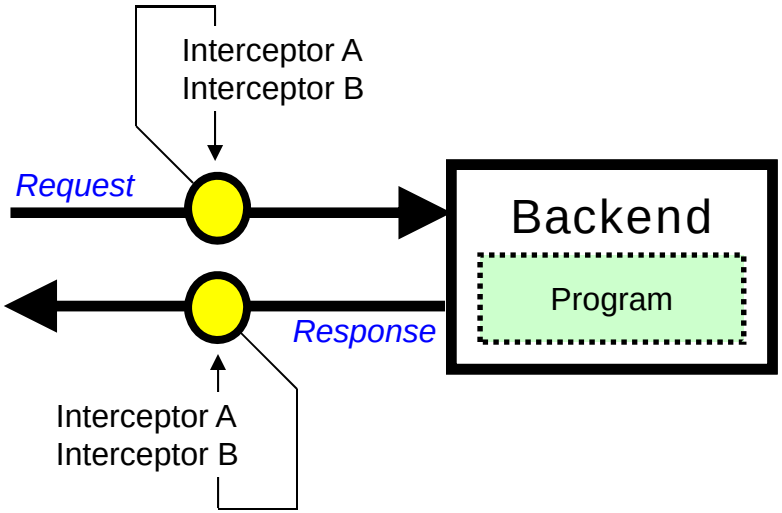
z/OS Connect EE



Overview of z/OS Connect interceptors

z/OS Connect EE

The interceptor framework provides a way to call code to do pre-invoke work and then again to do post-invoke work:



In `server.xml` you can:

- Define 'global interceptors,' which apply to all configured APIs and services
- Define interceptors specific to a given configured API or service

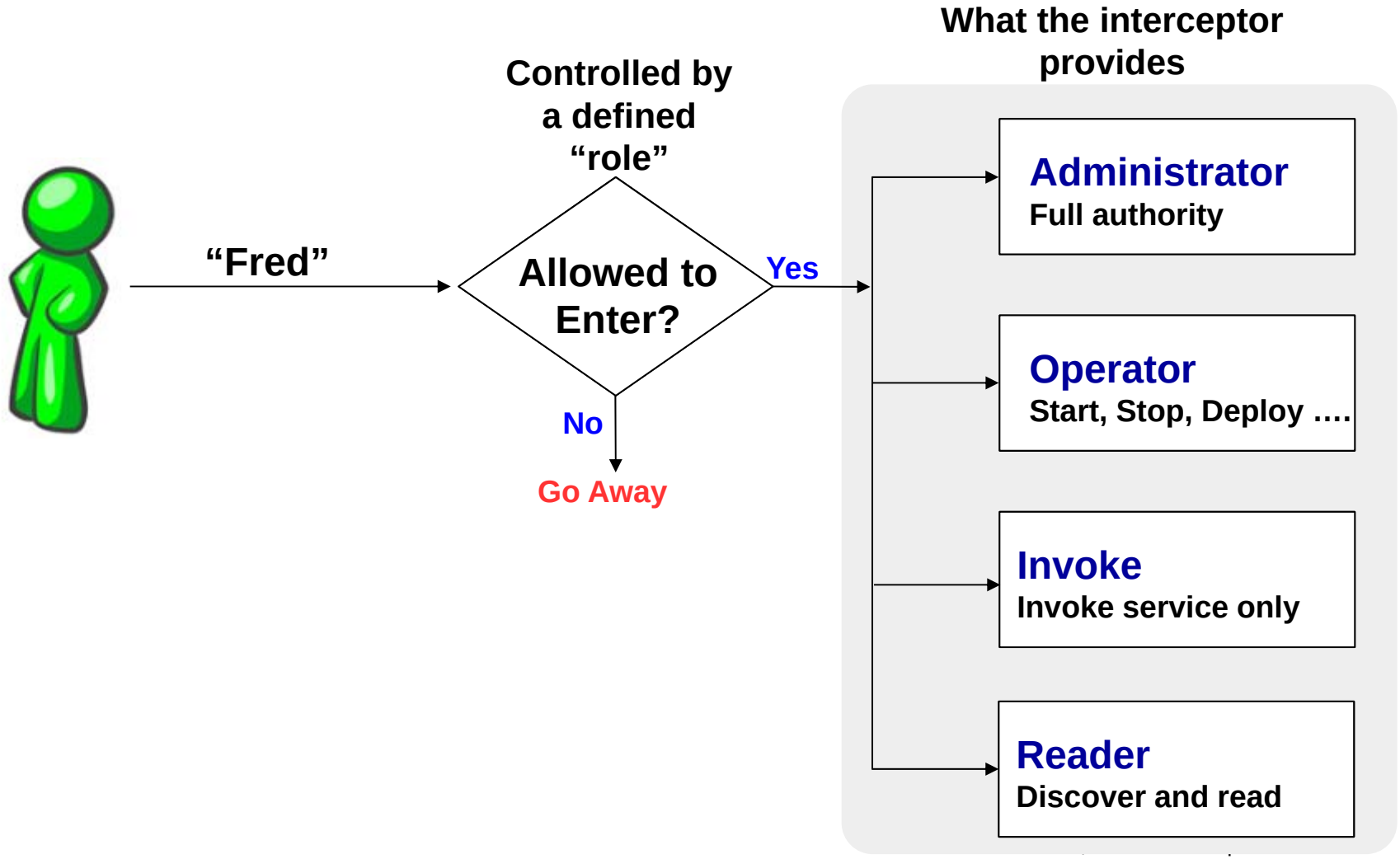
z/OS Connect comes with an authorization interceptor (which user can access which API or service) and an audit interceptor (for SMF recording)

It is also possible to write your own interceptor and have it called as part of request/response processing

Authorization interceptor

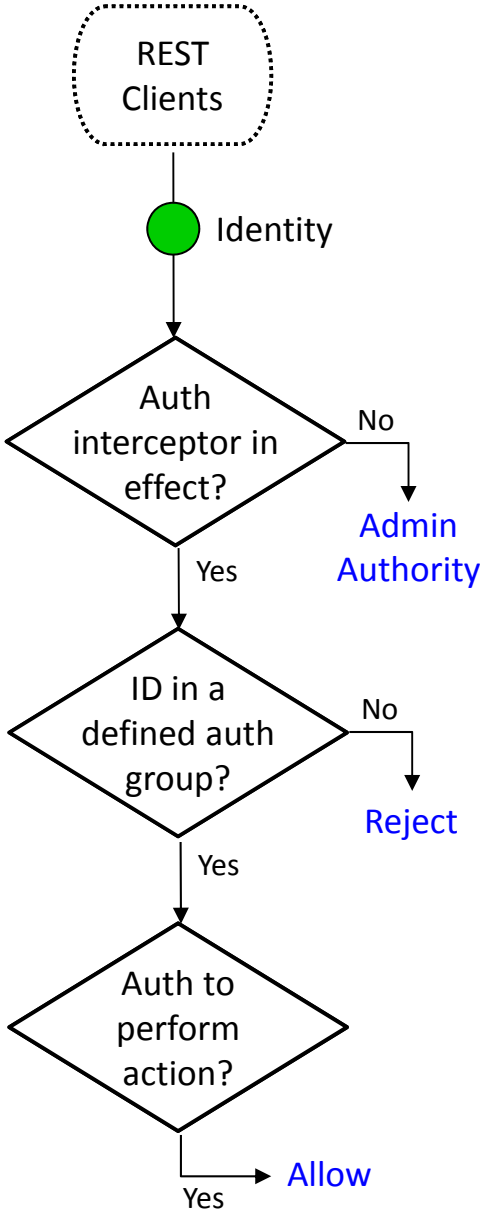
z/OS Connect EE

The “authorization interceptor” is a supplied piece of interceptor code that will check to see if the user has the authority to perform the action requested:



Configuring authorization

z/OS Connect EE



Definition of groups at the global level. Can also be defined at API and service level.

Pointer to interceptor list

```
<zosconnect_zosConnectManager
    globalAdminGroup="GMADMIN"
    globalInvokeGroup="GMINVOKE"
    globalInterceptorsRef="interceptorList_g" />

<zosconnect_zosConnectInterceptors
   id="interceptorList_g"
   interceptorRef="auth"/>

<zosconnect_authorizationInterceptor id="auth"/>
```

An interceptor "list" that defines just the "auth" interceptor.

Element that defines authorization interceptor to z/OS Connect EE.

Define at global (shown here), API or service layer

Override, opt-out … many ways to configure this

See Techdoc WP102439 "Interceptor Scenarios" PDF for examples of variations on configuring interceptors at different levels in z/OS Connect.

Configuring interceptors - Example

z/OS Connect EE

Interceptors defined as **global** apply to all the APIs defined to the instance of z/OS Connect (unless the global definition is overridden). Interceptors defined as API-level apply only to that API. The authorization interceptor works on the principle of user membership in a group.

```
<zosconnect_zosConnectManager globalInterceptorsRef="interceptorList_g"
globalAdminGroup="GMADMIN" globalInvokeGroup="GMINVOKE"/>

<zosconnect_authorizationInterceptor id="auth"/>
<zosconnect_auditInterceptor id="audit"/>

<zosconnect_zosConnectInterceptors id="interceptorList_g" interceptorRef="auth"/>
<zosconnect_zosConnectInterceptors id="interceptorList_s" interceptorRef="audit"/>

<zosconnect_zosConnectAPIs location="">
  <zosConnectAPI name="catalog" interceptorsRef="interceptorList_s" />
</zosconnect_zosConnectAPIs>
```

Audit

Audit (SMF) Interceptor

The audit interceptor writes SMF 123.1 records. Below is an example of some of the information captured:

- System Name
- Sysplex Name
- Job Name
- Job Prefix
- Address Space Stoken

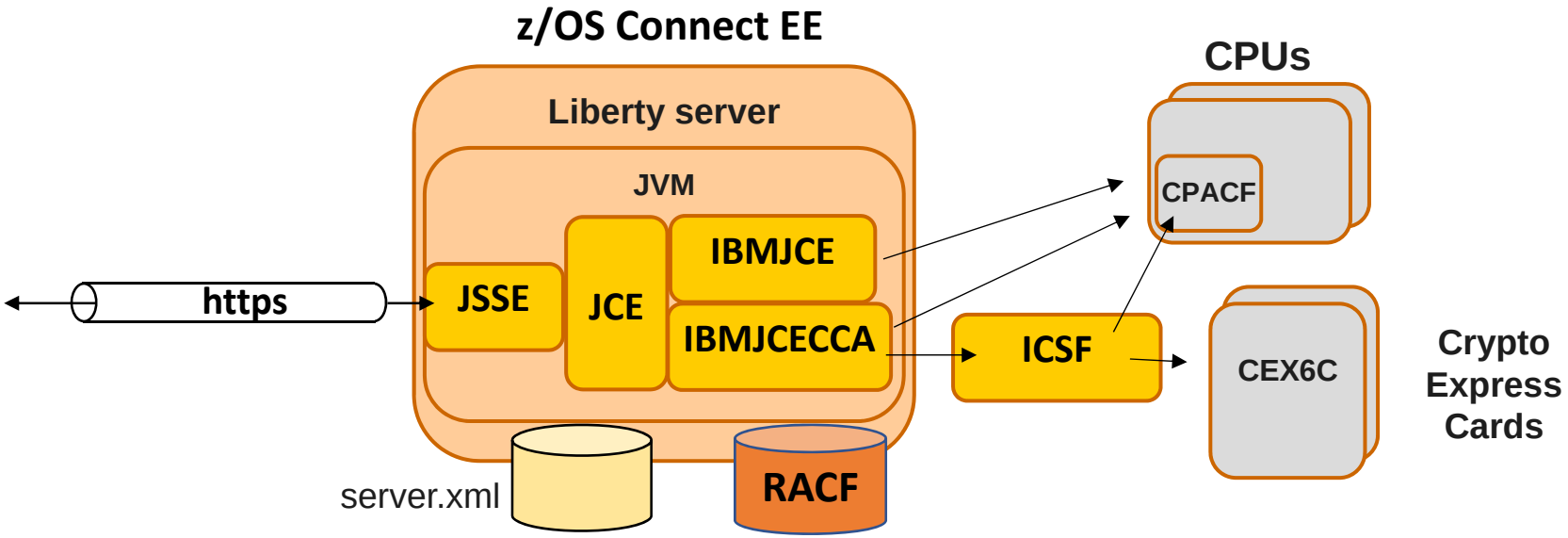
Server Identification Section

- Arrival Time
- Completion Time
- Target URI
- Input JSON Length
- Response JSON Length
- Method Name
- API or Service Name
- Userid
- Mapped user name

User Data Section

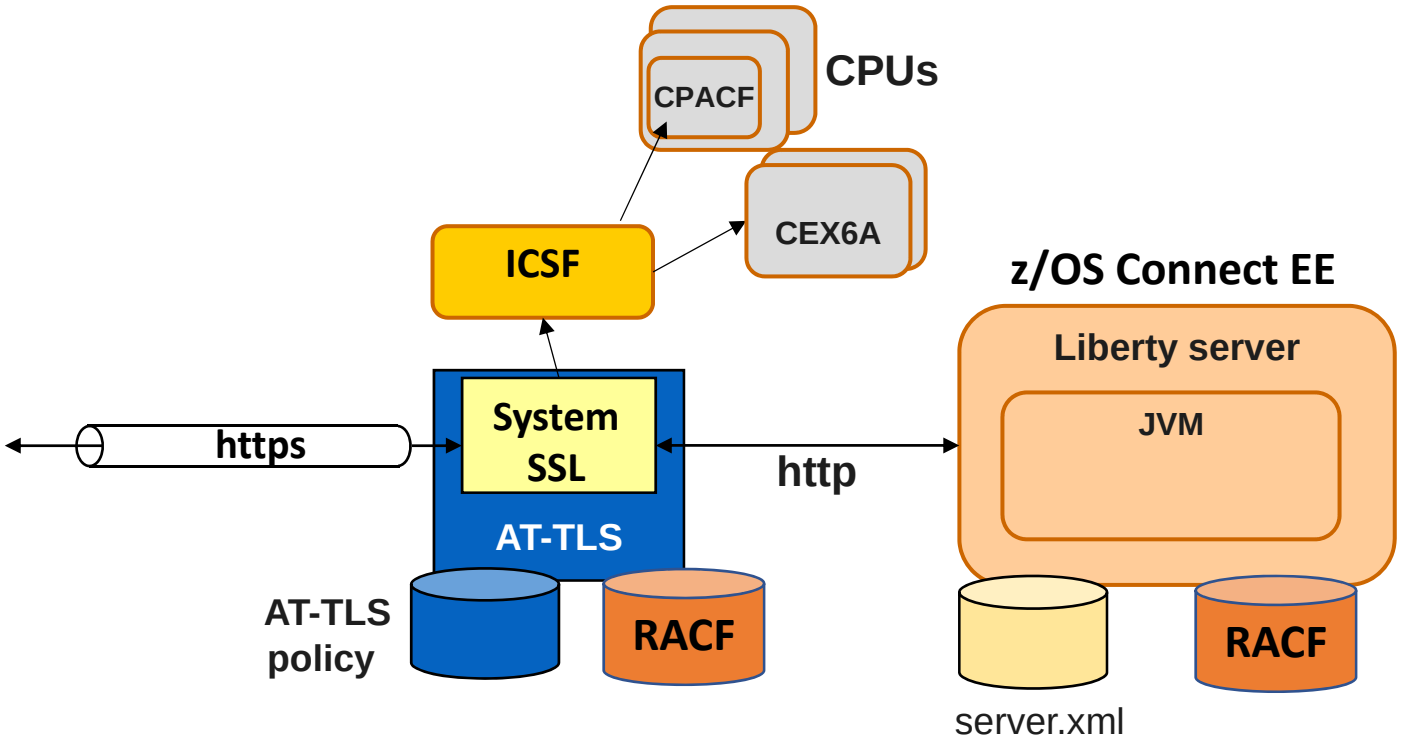
Encryption

Using JSSE with z/OS Connect EE



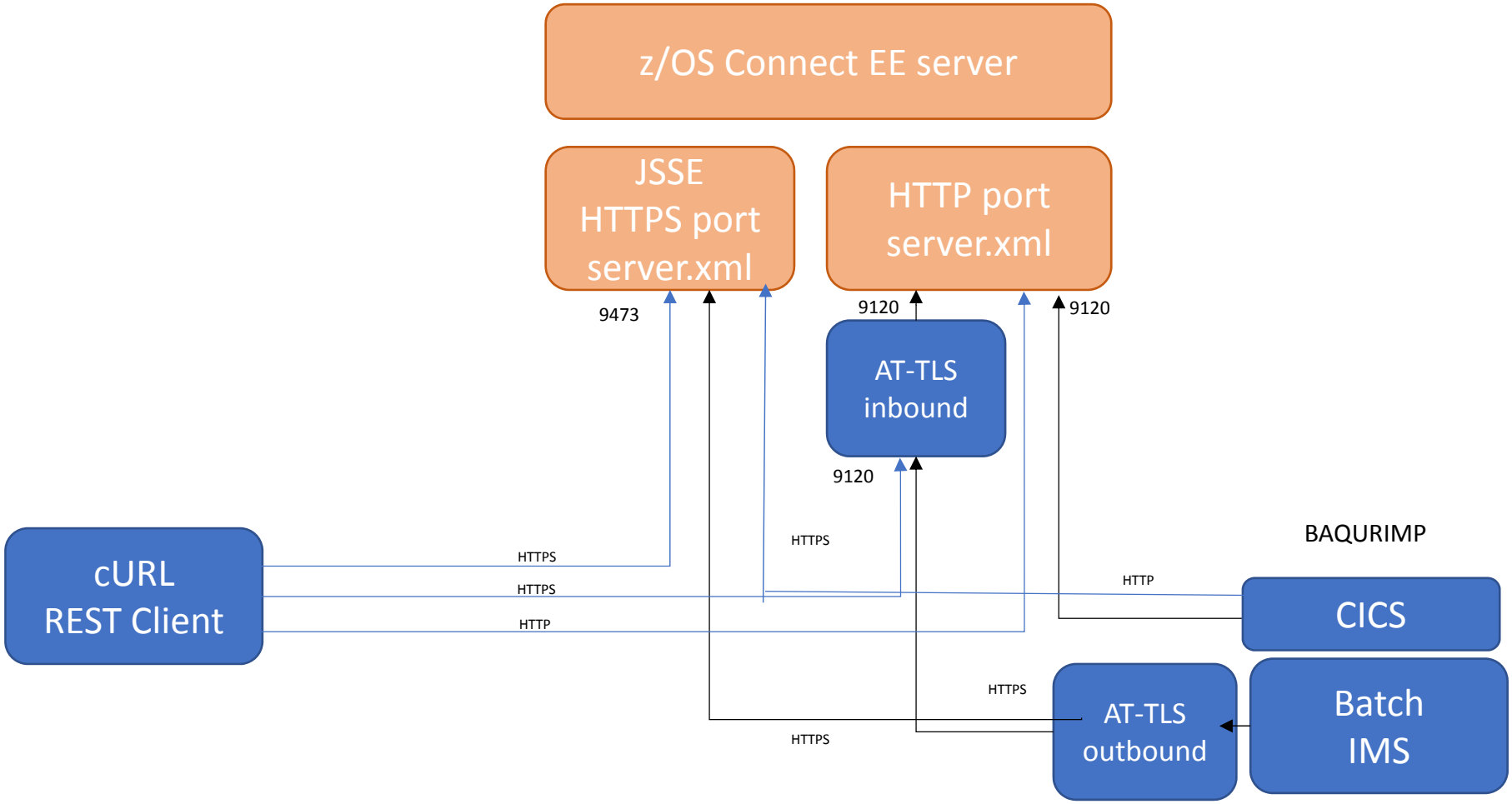
- z/OS Connect EE support for SSL/TLS is based on **Liberty server** support
- **Java Secure Socket Extension** (JSSE) API provides framework and Java implementation of SSL and TLS protocols used by Liberty HTTPS support
- **Java Cryptography Extension** (JCE) is standard extension to the Java Platform that provides implementation for cryptographic services
- **IBM Java SDK** for z/OS provides two different JCE providers, **IBMJCE** and **IBMJCECCA**

Using AT-TLS with z/OS Connect EE



- **Application Transparent TLS** (AT-TLS) creates a secure session on behalf of z/OS Connect
- Only define http ports in server.xml (z/OS Connect does not know that TLS session exists)
- Define TLS protection for all applications (including z/OS Connect) in **AT-TLS policy**
- AT-TLS uses **System SSL** which exploits the CPACF and Crypto Express cards via ICSF

AT-TLS Inbound Scenarios



JSSE and AT-TLS comparison

z/OS Connect EE

Capability	Description	JSSE	AT-TLS
1-way SSL	Verification of z/OS Connect certificate by client	Yes	Yes
2-way SSL	Verification of client certificate by z/OS Connect	Yes	Yes
SSL client authentication	Use of client certificate for authentication	Yes	**No**
Support for requireSecure option on APIs	Requires that API requests are sent over HTTPS	Yes	**No**
Persistent connections	To reduce number of handshakes	Yes	Yes
Re-use of SSL session	To reduce number of full handshakes	Yes	Yes
Shared SSL sessions	To share SSL sessions across cluster of z/OS Connect instances	**No**	Yes
zIIP processing	Offload TLS processing to zIIP	Yes	**No**
CPACF	Offload symmetric encryption to CPACF	Yes	Yes
CEX6	Offload asymmetric operations to Crypto Express cards	Yes	Yes

Configuring TLS Encryption with JSSE

Cyphers

- During the TLS handshake, the TLS protocol and data exchange cipher are negotiated
- Choice of cipher and key length has an impact on performance
- You can restrict the protocol (SSL or TLS) and ciphers to be used
- Example setting server.xml file

```
<ssl id="DefaultSSLSettings"
keyStoreRef="defaultKeyStore" sslProtocol="TLSv1.2"
enabledCiphers="TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384"/>
```

- This configures use of TLS 1.2 and two supported ciphers
- It is recommended to control what ciphers can be used in the server rather than the client

Persistent connections

- Persistent connections can be used to avoid too many handshakes
- Configured by setting `the keepAliveEnabled` attribute on the httpOptions element to **true**
- Example setting server.xml file

```
<httpEndpoint host="*" httpPort="80" httpsPort="443"
id="defaultHttpEndpoint" httpOptionsRef="httpOpts"/>

<httpOptions id="httpOpts" keepAliveEnabled="true"
maxKeepAliveRequests="500" persistTimeout="1m"/>
```

- This sets the connection timeout to **1 minute** (default is 30 seconds) and sets the maximum number of persistent requests that are allowed on a single HTTP connection to **500**
- It is recommended to set a maximum number of persistent requests when connection workload balancing is configured
- It is also necessary to configure the client to support persistent connections

SSL sessions

- When connections timeout, it is still possible to avoid the impact of full handshakes by reusing the SSL session id
- Configured by setting the `sslSessionTimeout` attribute on the sslOptions element to an amount of time
- Example setting server.xml file

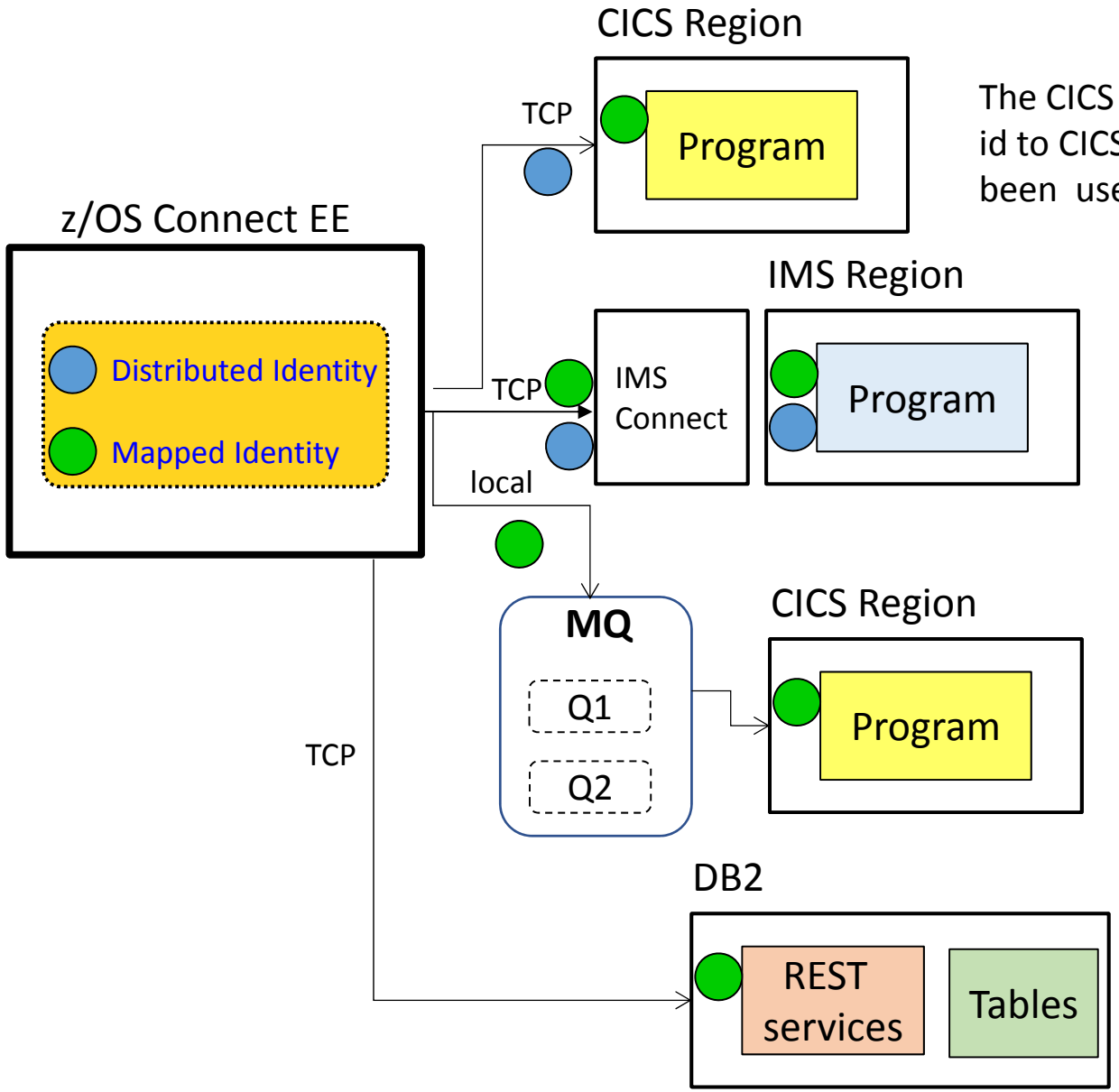
```
<httpEndpoint host="*" httpPort="80" httpsPort="443"
id="defaultHttpEndpoint" httpOptionsRef="httpOpts“
sslOptionsRef="mySSLOptions"/>

<httpOptions id="httpOpts" keepAliveEnabled="true"
maxKeepAliveRequests=“100" persistTimeout="1m"/>

<sslOptions id="mySSLOptions" sslRef="DefaultSSLSettings"
sslSessionTimeout=“10m”/>
```

- This sets the timeout limit of an SSL session to **10 minutes** (default is 8640ms)
- SSL session ids are not shared across z/OS Connect servers

Flowing identities to back end systems

Flowing an identity to the back end

z/OS Connect EE



The CICS SP propagates the distributed id to CICS (or sends the SAF id if it has been used for authentication)

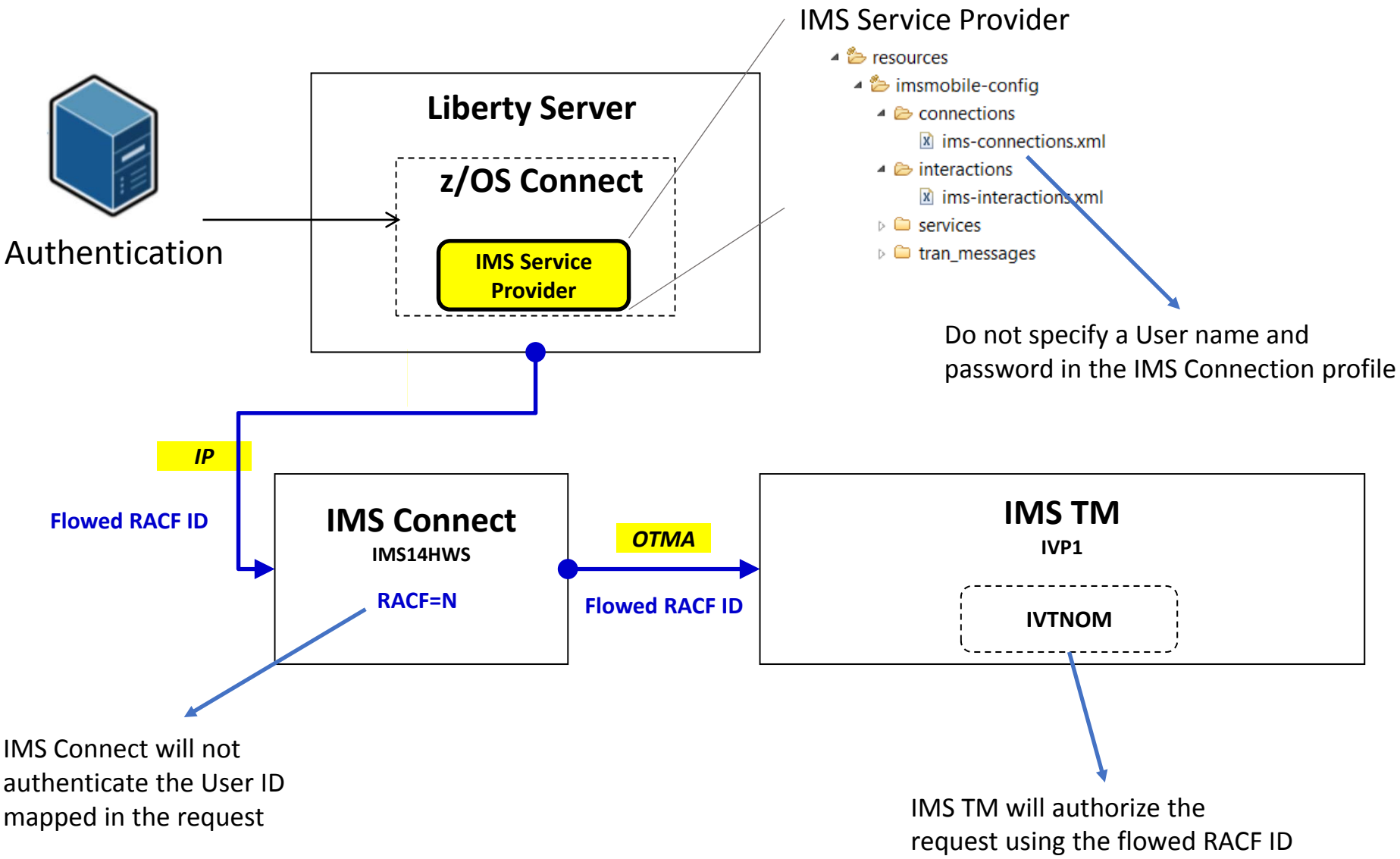
The IMS SP asserts the mapped id to IMS but also sends the distributed id for audit purposes

The MQ SP asserts the mapped id to CICS

The REST client SP can request a Passticket and send with mapped ID to DB2 (available today in V3.0.15)

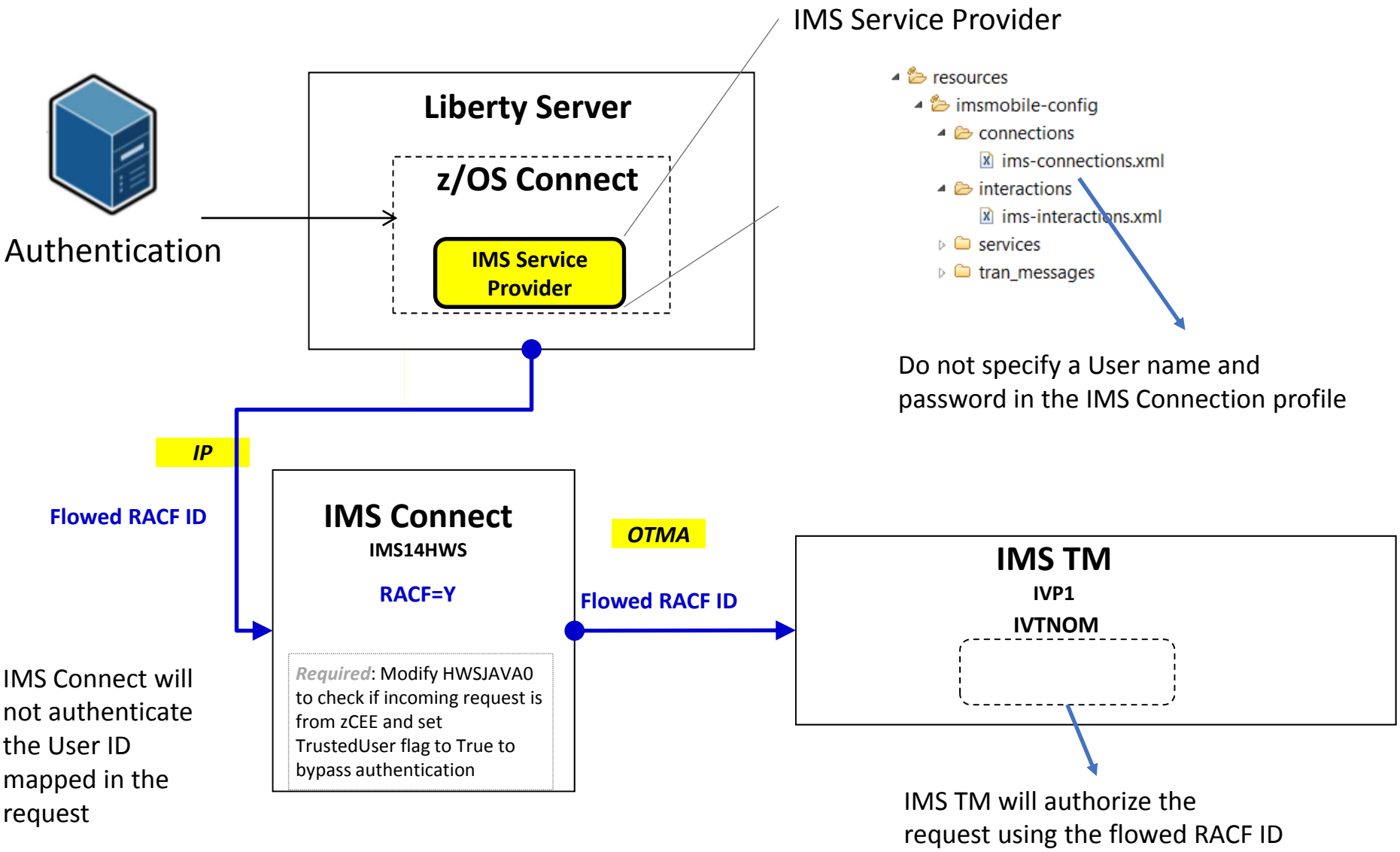
Flowing a RACF ID to IMS

z/OS Connect EE



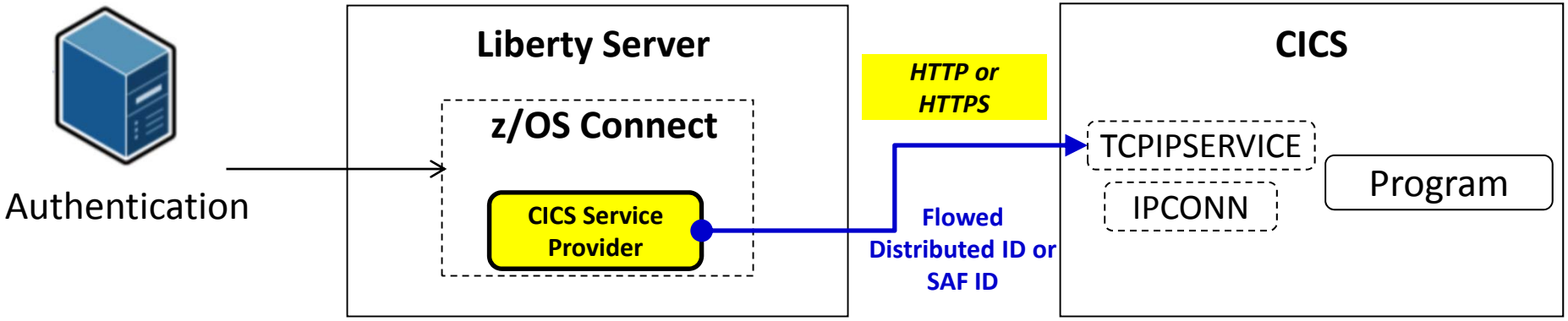
Flowing a RACF ID to IMS

z/OS Connect EE



Flowing a user ID with CICS service provider

z/OS Connect EE



IPIC connections enforce **bind** security to prevent an unauthorized client system from connecting to CICS, **link** security to restrict the resources that can be accessed over a connection to a CICS system, and **user** security to restrict the CICS resources that can be accessed by a user

Distributed identities can be propagated to CICS and then mapped to a RACF user ID by CICS. You can then view the distinguished name and realm for a distributed identity in the association data of the CICS task. **Important**: If the z/OS Connect EE server is not in the same sysplex as the CICS system, you must use an IPIC SSL connection that is configured with client authentication.

If a SAF ID is used for authentication (e.g basicauth with a SAF registry) then the SAF ID is passed to CICS.

CICS IPCONN

z/OS Connect EE

```
DEFINE IPCONN(ZOSCONN) GROUP(SYSPGRP)
     APPLID(ZOSCONN)
     NETWORKID(ZOSCONN)
     TCPIPSERVICE(ZOSCONN)
LINKAUTH(SECUSER)
     USERAUTH(IDENTIFY)
     IDPROP(REQUIRED)
```

Must match `zosConnectApplid` set in `zosconnect_cicsIpicConnection`

Must match `zosConnectNetworkid` set in `zosconnect_cicsIpicConnection`

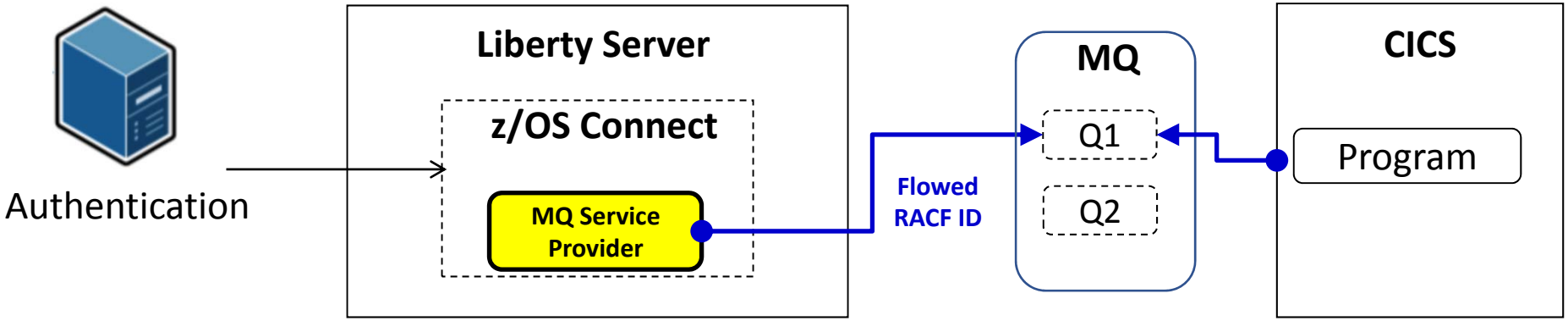
Specify name of TCPIPSERVICE

Requests run under the flowed user ID

```
<zosconnect_cicsIpicConnection id="cscvinc"
      host="wg31.washington.ibm.com"
      zosConnectNetworkid="ZOSCONN"
      zosConnectApplid="ZOSCONN"
      port="1491"/>
```

Flowing a user ID with MQ service provider

z/OS Connect EE

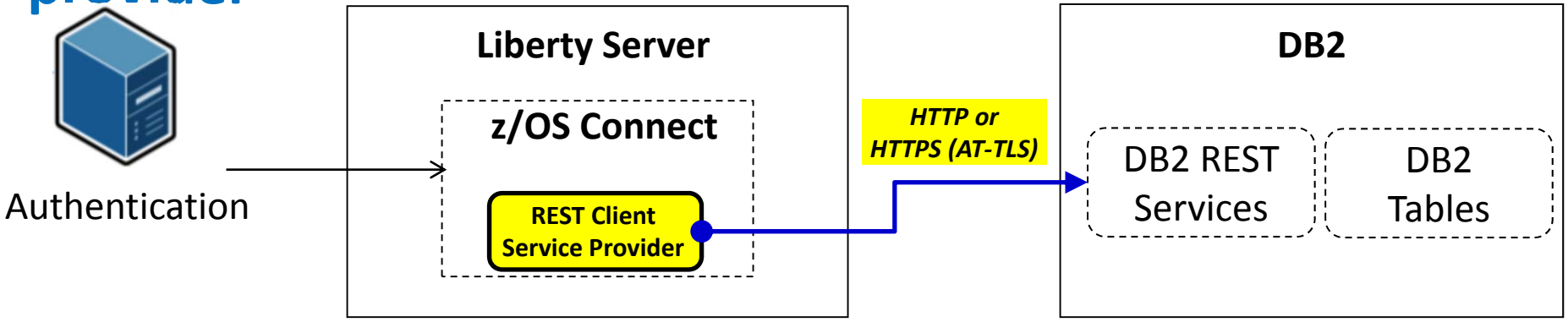


Configuration attributes on the `mqzOSConnectService` element, and the `properties.wmqJMS` sub-element of the `jmsConnectFactory` element affect which user ID and optional password are presented to the queue manager.

Set **`useCallerPrincipal=true`** to flow the authenticated RACF user ID

Setting the user ID for the REST client service provider

z/OS Connect EE



Authentication options:

1. **User ID / password**
2. **TLS Client Certificate**
3. **Passticket support (In Open Beta currently)**

```
<zosconnect_zosConnectServiceRestClientBasicAuth
...
userName="EMPLOY1"
password="{xor}GhIPExAGDwg="/>
```

Specify a user name and password to be used in the HTTP header with the DB2 REST Service

```
<zosconnect_zosConnectServiceRestClientConnection
...
sslCertsRef="sslCertificates"/>
```

TLS client authentication

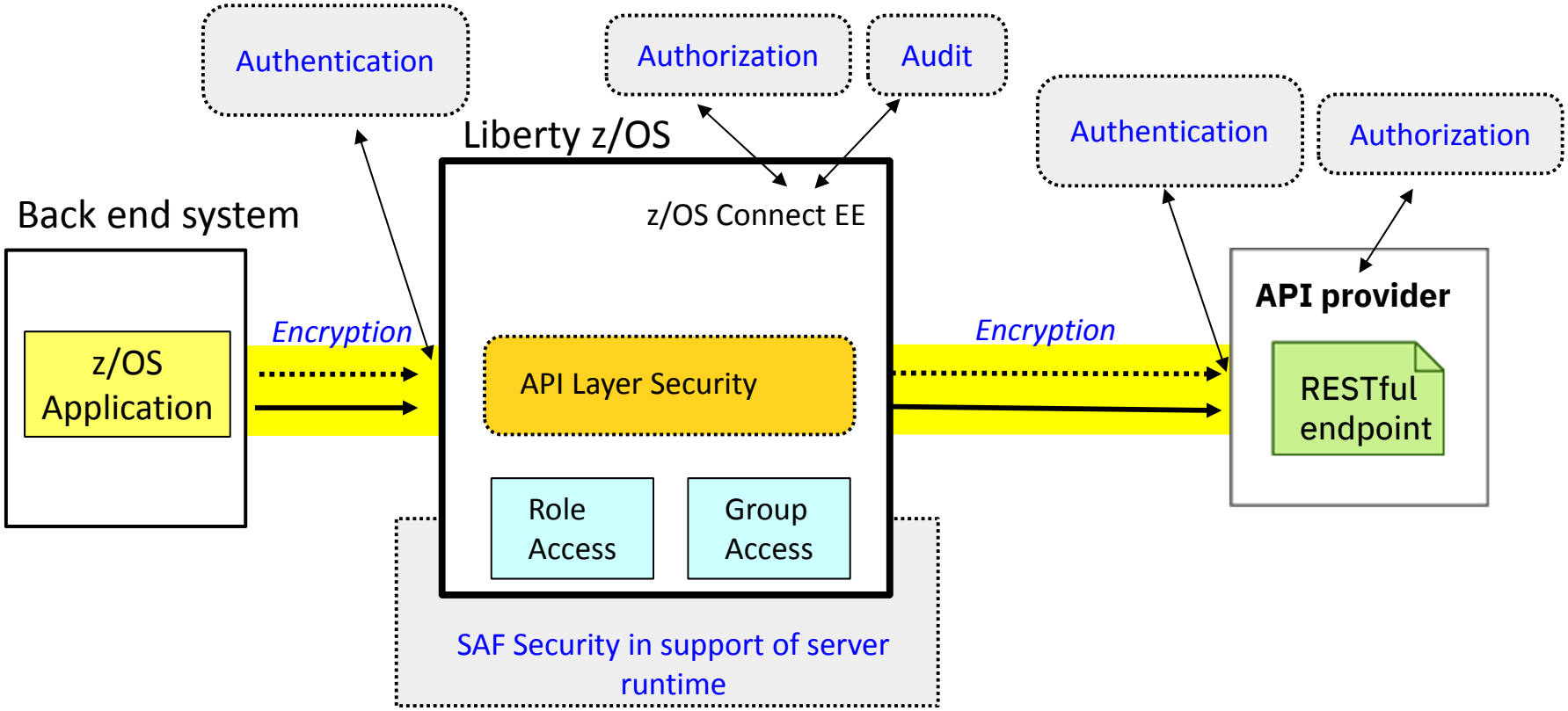
```
<zosconnect_zosConnectServiceRestClientBasicAuth
...
applName="applName"/>
```

z/OS Connect requests a PassTicket from RACF

What's different for API Requester?

API requester security – overview

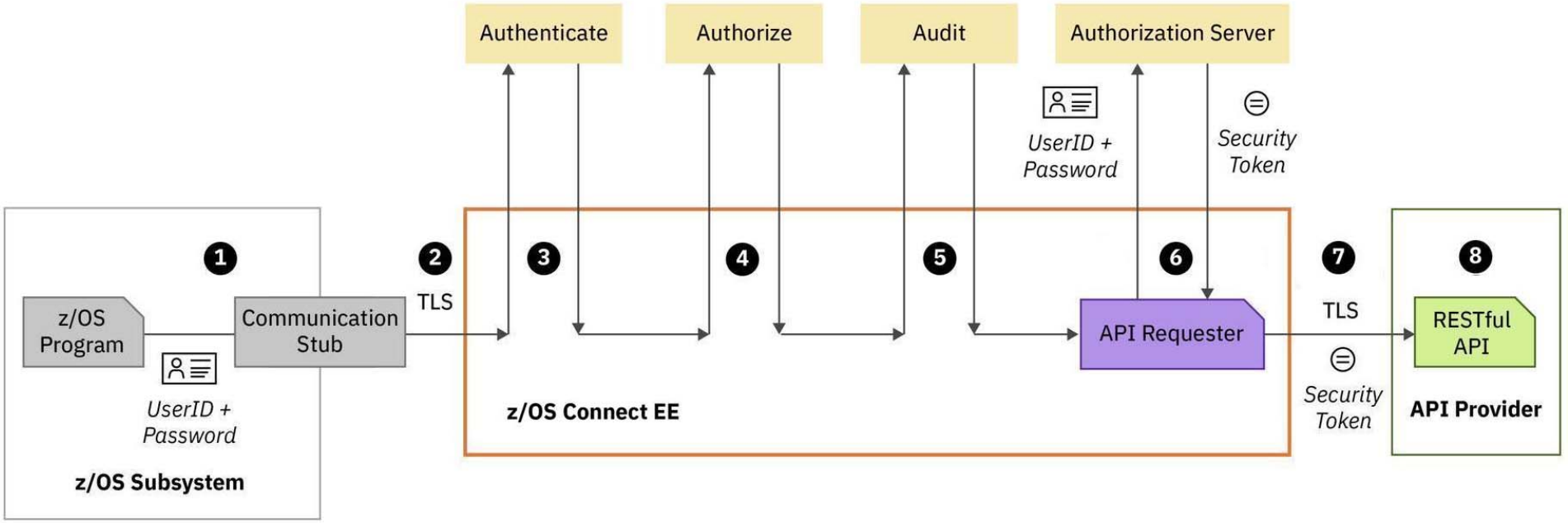
z/OS Connect EE



1. Authentication (basic, client certificate)
2. Encryption (aka "SSL" or "TLS")
3. Authorization (OAuth)
4. Audit
5. Configuring security with SAF

Typical z/OS Connect EE security flow

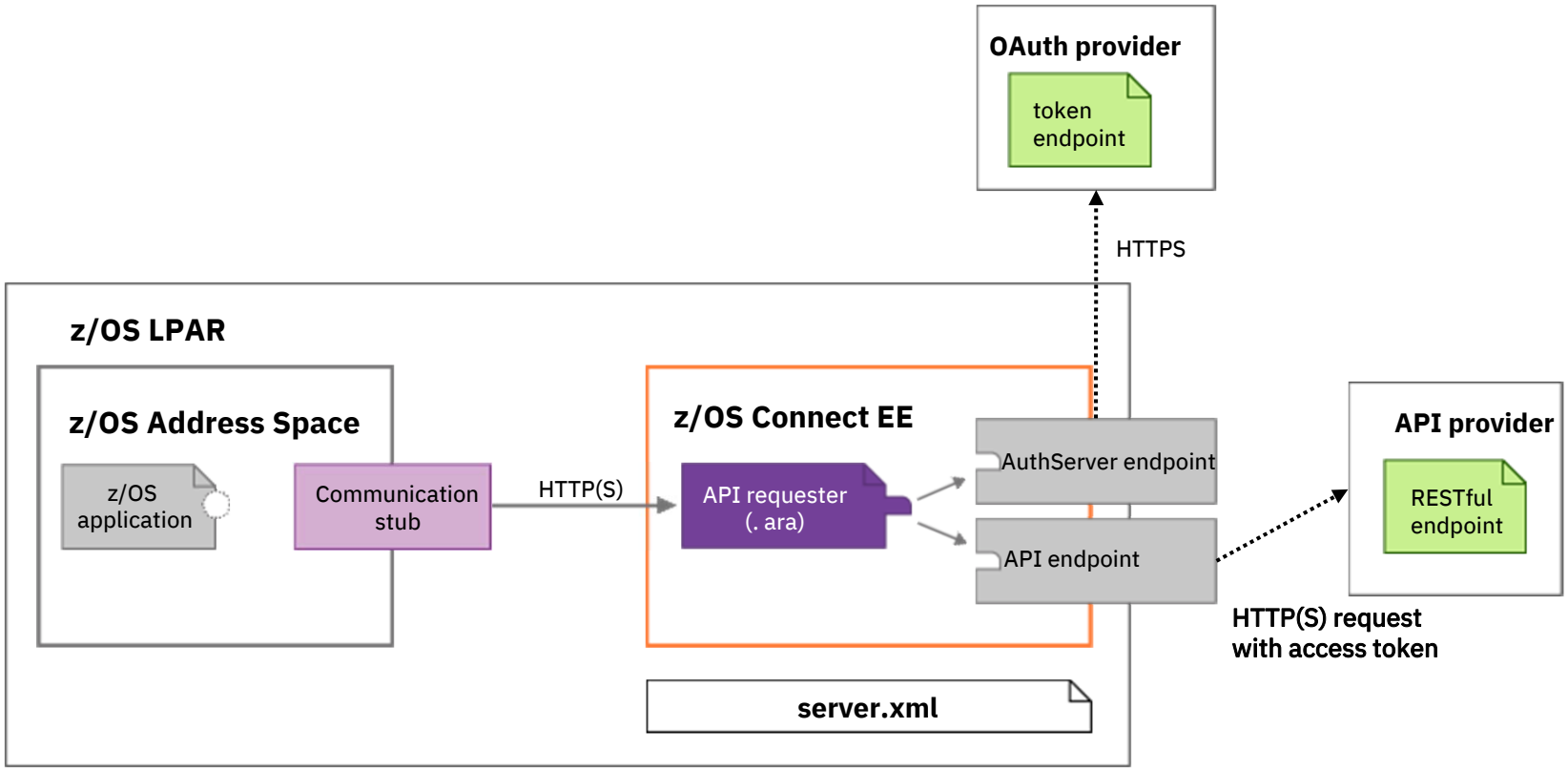
z/OS Connect EE



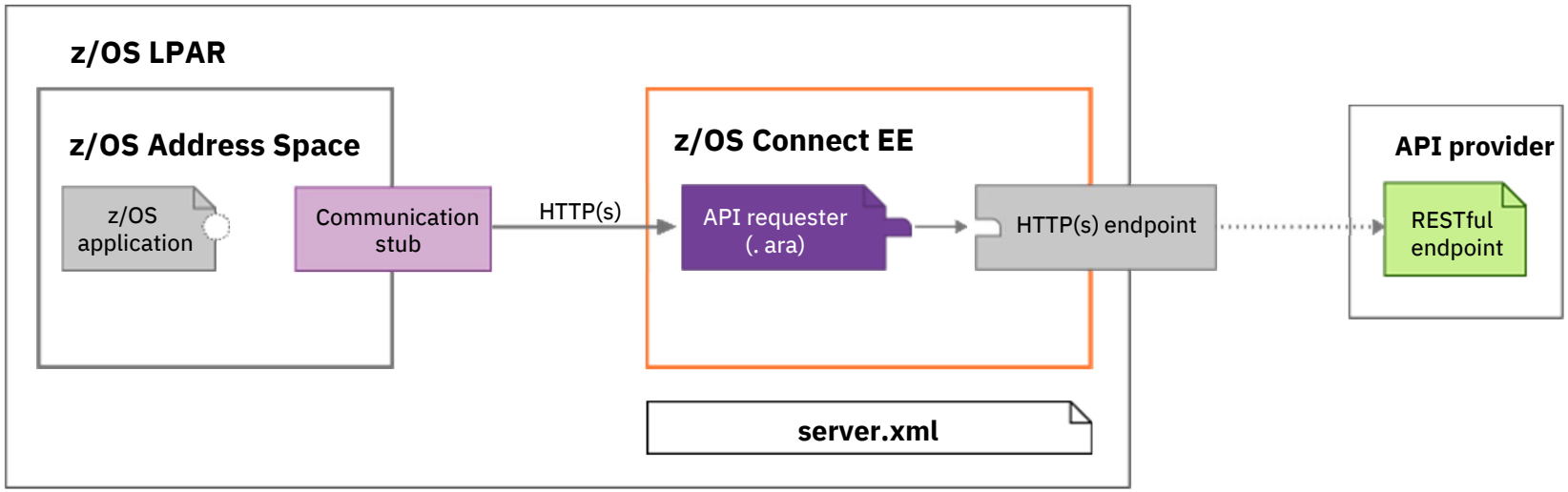
1. A user ID and password can be used for basic authentication by the z/OS Connect EE server
2. Connection between the CICS, IMS, or z/OS application and the z/OS Connect EE server can use TLS
3. Authenticate the CICS, IMS, or z/OS application.
4. Authorize the authenticated user ID to connect to z/OS Connect EE and to perform specific actions on z/OS Connect EE API requesters
5. Audit the API requester request
6. Pass the user ID and password credentials to an authorization server to obtain a security token.
7. Secure the connection to the external API provider, and provide security credentials such as a security token to be used to invoke the RESTful API
8. The RESTful API runs in the external API provider

OAuth 2.0 support

z/OS Connect EE



Encryption



Options:

1. AT-TLS
2. CICS TLS (System SSL)

1. JSSE
2. AT-TLS

Configuring OAuth support

z/OS Connect EE

For **OAuth**, two grant types are supported:

- Resource Owner Password Credential [a.k.a. password]
- Client Credentials [a.k.a. client_credentials]

The access token is a way for the API provider to validate the client application rights to invoke its APIs.

```
<zosconnect_endpointConnection id="orderDispatchAPI"
    host="https://154.2.45.123" port="443"
    authenticationConfigRef="myOAuthConfig"/>

<zosconnect_oAuthConfig id="myOAuthConfig"
    grantType="client_credentials"
    authServerRef="myOAuthProvider" />

<zosconnect_authorizationServer id="myOAuthProvider"
    tokenEndpoint="https://154.2.45.123/oauth2/token"
    basicAuthRef="myAppID"/>

<zosconnect_authData id="myAppID" user="myClientID"
    password="myClientSecret" />
```

optional

Summary

Summary

- Define clear security requirements before deciding on a security design

- Security design needs to consider
  - Authentication
  - Encryption
  - Authorization
  - Audit
  - Protection against attack
  - Rate limiting

- Because z/OS Connect EE is based on Liberty it benefits from a wide range of Liberty security capabilities

- z/OS Connect EE has it's own security capabilities in the form of the authorization and audit interceptors

- Look at the security solution end to end, including the security capabilities of the API Gateway

More information

z/OS Connect EE