zOSSEC1 – IBM z/OS Connect EE Security

A dive into Liberty and z/OS Connect Security

Mitch Johnson
mitchj@us.ibm.com
emitchj@gmail.com

Wildfire
Technical Hands-On Workshops
IBM

IBM ATG – IBM Z

Topics

- OMVS, Liberty, and RACF Security Review
- z/OS Connect Security Overview
- Authentication
  - –Basic Authentication
  - –Mutual Authentication using TLS
  - –Third Party Tokens
- Encryption and Message Integrity using TLS
- Authorization
- Propagating identities to z/OS subsystems
- z/OS Connect API Requester and third-party tokens

Disclaimer

- The information in this presentation was derived from various product Knowledge Centers (KC).

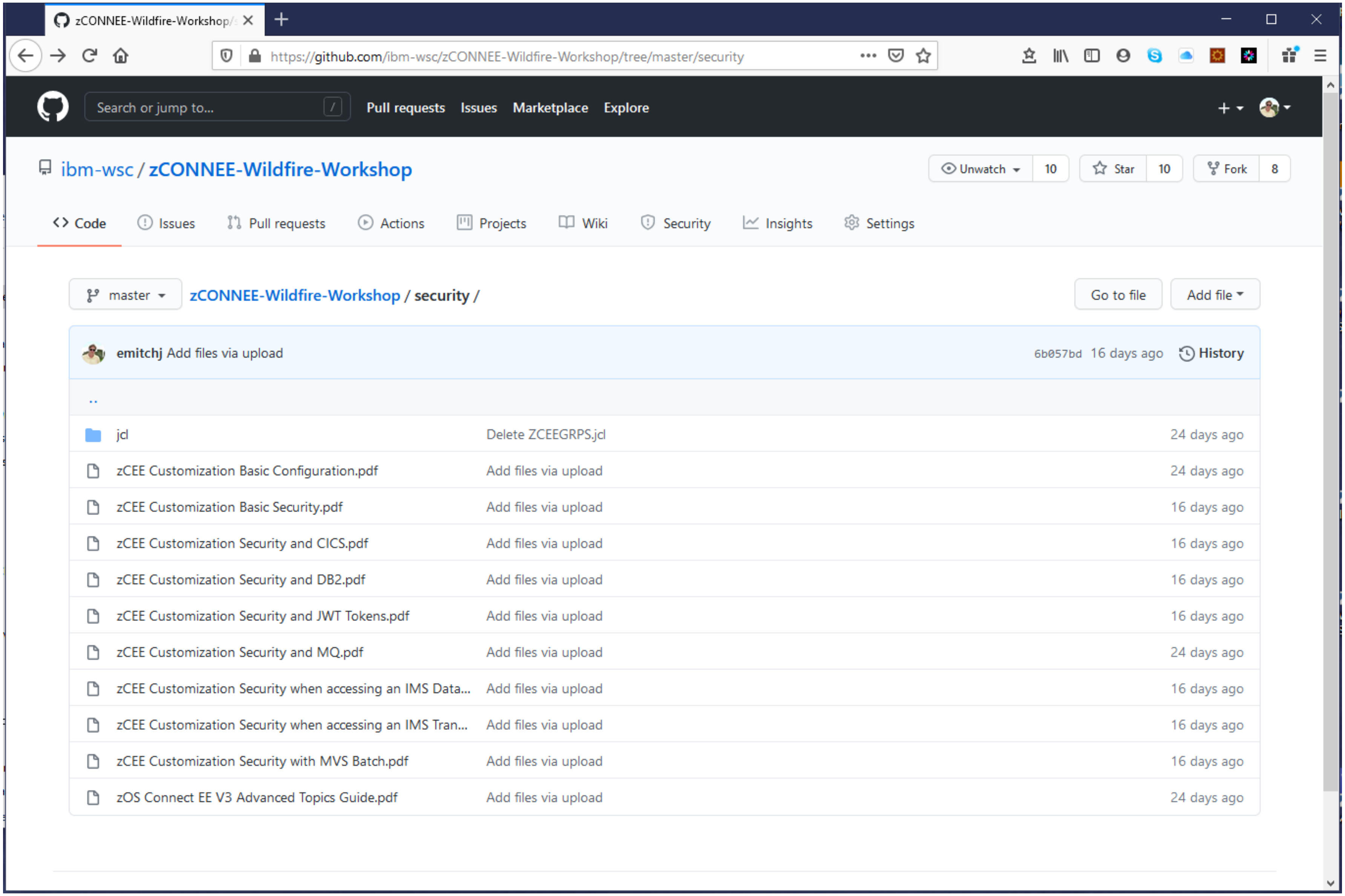
- Additional information included in this presentation was been distilled from years of experience implementing security using RACF with z/OS products like CICS, IMS, Db2, MQ, etc. as well as Java runtimes environments like WebSphere Application Server and Liberty.

- There will be additional information on slides that will designated as Tech/Tips. These contain information which will be at perhaps at least be interesting and hopefully, useful to the reader.

- A Liberty or z/OS Connect icon will appear on slides where the information is specific to these products. Don’t hesitate to ask questions as to why the icon does or does not appear on certain slides.

- The examples, tips, etc. present in this material are based on firsthand experiences and are not necessarily sanctioned by z/OS Connect development.

Detailed examples of what is cover are available



zCONNEE-Wildfire-Workshop/
https://github.com/ibm-wsc/zCONNEE-Wildfire-Workshop/tree/master/security
Search or jump to...
Pull requests
Issues
Marketplace
Explore
ibm-wsc / zCONNEE-Wildfire-Workshop
Unwatch 10
Star 10
Fork 8
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
master
zCONNEE-Wildfire-Workshop / security /
Go to file
Add file
emitchj Add files via upload
6b057bd 16 days ago
History
..

jcl	Delete ZCEEGRPS.jcl	24 days ago
zCEE Customization Basic Configuration.pdf	Add files via upload	24 days ago
zCEE Customization Basic Security.pdf	Add files via upload	16 days ago
zCEE Customization Security and CICS.pdf	Add files via upload	16 days ago
zCEE Customization Security and DB2.pdf	Add files via upload	16 days ago
zCEE Customization Security and JWT Tokens.pdf	Add files via upload	16 days ago
zCEE Customization Security and MQ.pdf	Add files via upload	24 days ago
zCEE Customization Security when accessing an IMS Data...	Add files via upload	16 days ago
zCEE Customization Security when accessing an IMS Tran...	Add files via upload	16 days ago
zCEE Customization Security with MVS Batch.pdf	Add files via upload	16 days ago
zOS Connect EE V3 Advanced Topics Guide.pdf	Add files via upload	24 days ago

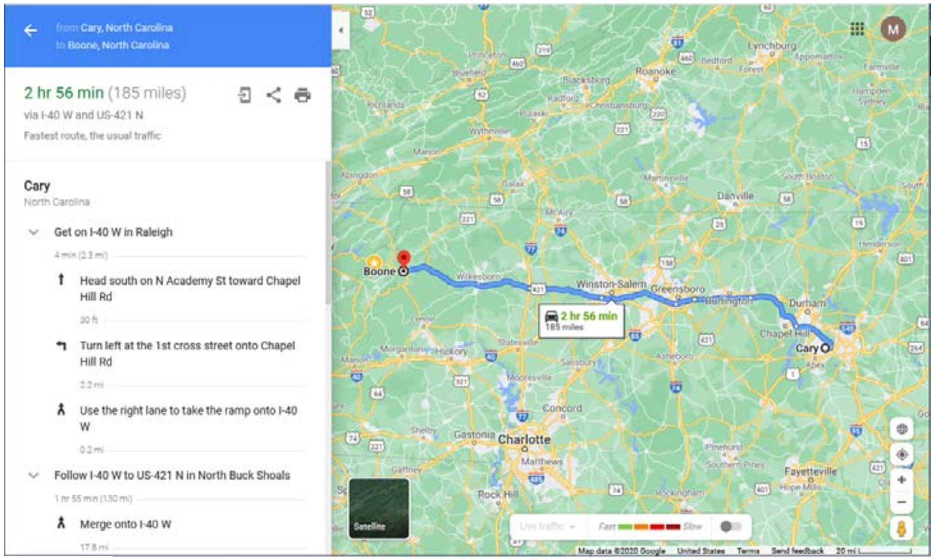
https://github.com/ibm-wsc/zCONNEE-Wildfire-Workshop/tree/master/exercises

Map of the journey to security

Sometimes it seems like implementing security is starting with a map like this:

"*Here be dragons*" (hic sunt dracones in Latin) means dangerous or unexplored territories.

When we would really like to have is a Google map with step-by-step directions:



from Cary, North Carolina
to Boone, North Carolina

2 hr 56 min (185 miles)
via I-40 W and US-421 N
Fastest route, the usual traffic

Cary
North Carolina

Get on I-40 W in Raleigh
4 min (2.3 mi)

Head south on N Academy St toward Chapel Hill Rd
30 ft

Turn left at the 1st cross street onto Chapel Hill Rd
2.2 mi

Use the right lane to take the ramp onto I-40 W
0.2 mi

Follow I-40 W to US-421 N in North Buck Shoals
1 hr 55 min (130 mi)

Merge onto I-40 W
17.8 mi

But when securing Liberty and z/OS Connect, there is probably not a direct route to your destination. There will a series waypoints or intermediate steps on the journey which must be considered.

The waypoints to be considered involve:

- Providing secure access between middleware components that use disparate security technologies e.g., registries like LDAP, SAF, TLS etc. to propagate security credentials from a client all the way to the targeted resource.

➢This is a driver for implementing open security models like OAuth and OpenID Connect and standard tokens like JWT

- Integrating security involves different products including z/OS Connect, WebSphere Liberty Profile on z/OS with CICS, IMS, Db2, MQ,… probably for the first time in your environment.

➢ Security for of these components are all documented in different places

- Considering that security is often at odds with **performance**, the more secure techniques often mean more processing overhead, especially if not configured optimally

➢Security is usually not a choice, but mandatory

A single simple step-by-step linear map is not always possible, the best approach may be to build a solution using components, one step at a time based on the waypoints involved and the ultimate goal. Providing security by understanding these waypoints and the corresponding options is the focus of this presentation.

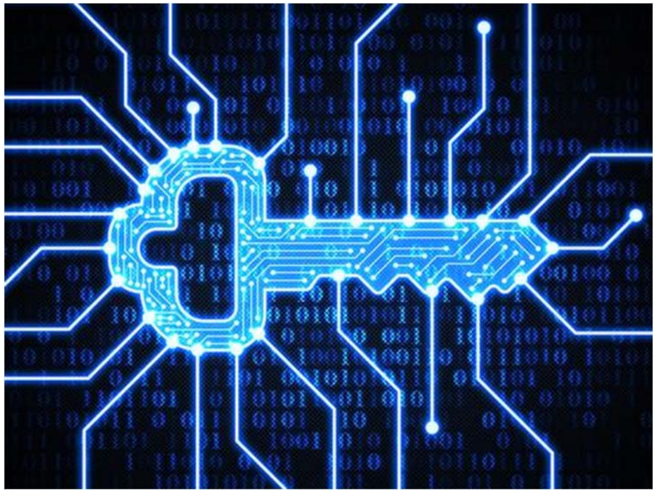
General security terms or considerations

Security involves

- Identifying who or what is requesting access (**Authentication)**
  - Basic Authentication
  - Mutual Authentication using TLS
  - Third Party Tokens

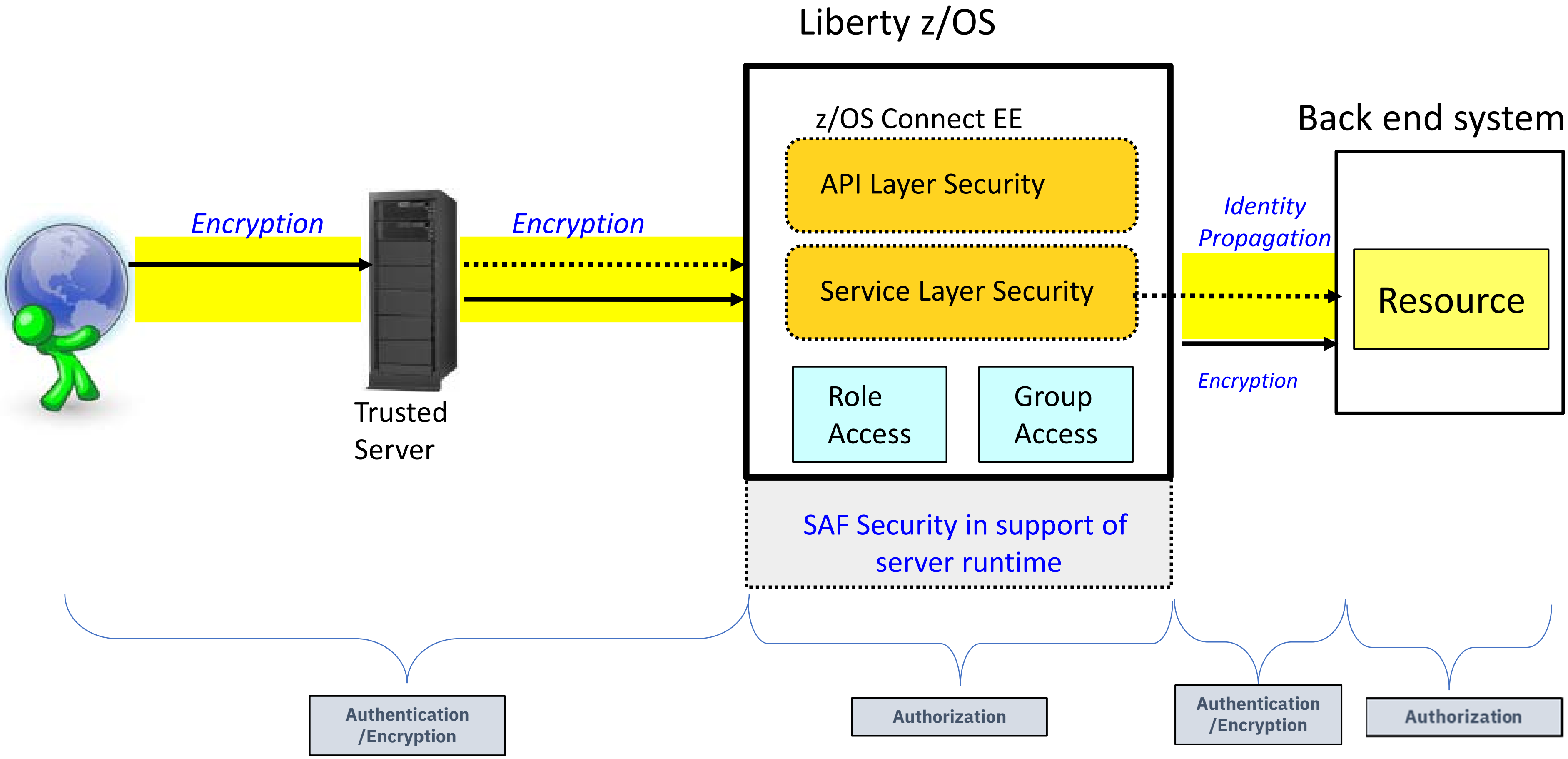
- Ensuring that the message has not been altered in transit (**Data Integrity)** and ensuring the confidentiality of the message in transit (**Encryption)**
  - TLS (encrypting messages and generating/sending a digital signature)

- Controlling access (**Authorization)**
  - Is the authenticated identity authorized to access to z/OS Connect
  - Is the authenticated identity authorized to access a specific API, Services, etc.

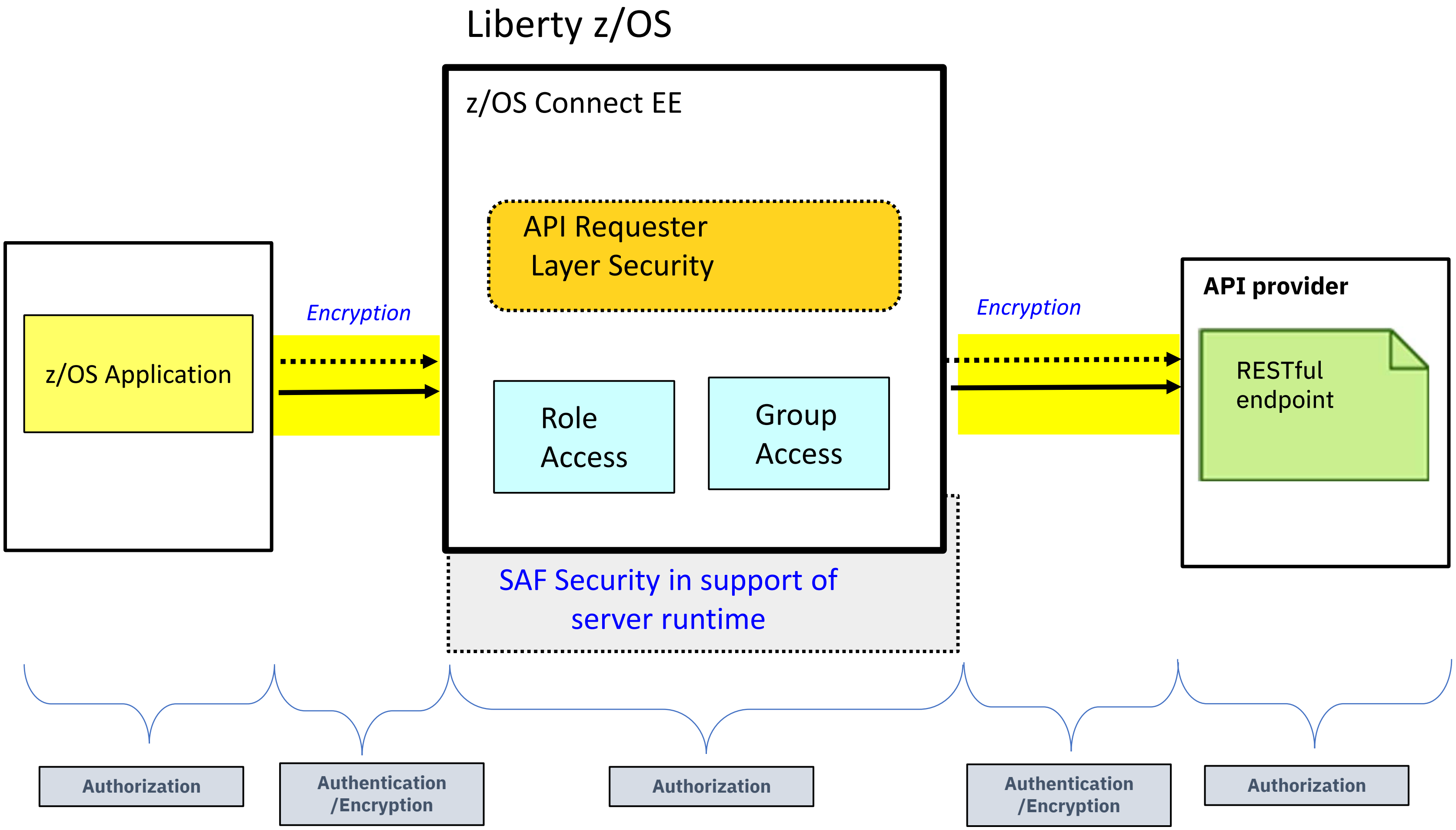


This presentation is focused in these areas.

API provider security overview (waypoints)



API requester security – overview (waypoints)



But before starting, let's begin with a review of some basic OMVS, Liberty and RACF security/configuration details and options related to security

OMVS security - Unix file permissions

	Owner			Group			Other		
	Read	Write	Execute	Read	Write	Execute	Read	Write	Execute
Bit	1	1	1	1	0	1	0	0	0
Base-2 Value	[4]	[2]	[1]	[4]	[2]	[1]	[4]	[2]	[1]
	4 +	2 +	1 =	4 +	0 +	1 =	0 +	0 +	0 =

Owner

7 The owner has READ, WRITE and EXECUTE

The **owner** of the file or directory

chmod –R* u+rwx zceesrv1

Group

5 The group has READ and EXECUTE, but not WRITE

IDs that are part of the **group** for the file or directory

chmod g+rwx server.xml

Other

0 Others have nothing

IDs that are not the owner and not part of the group; that is, **other**

chmod –R* o+rx resources
chmod –R* o-w resources/security

mitchj@us.ibm.com

-R* indicates recursion

Default server configuration directories and files

ID=**LIBSERV**
Group=**LIBGRP**

```
export JAVA_HOME=<path_to_64_bit_Java>
export WLP_USER_DIR=/var/zosconnect
./server create zceesrv1
```

```
/var/zosconnect          750  LIBSERV LIBGRP
  /servers               750  LIBSERV LIBGRP
    /zceesrv1            750  LIBSERV LIBGRP
      /logs              777  LIBSERV LIBGRP
       ⛁ messages.log    666  LIBSERV LIBGRP
      /resources         750  LIBSERV LIBGRP
        /zosconnect      750  LIBSERV LIBGRP
         /apis           750  LIBSERV LIBGRP
         /apiRequesters  750  LIBSERV LIBGRP
         /rules          750  LIBSERV LIBGRP
         /services       750  LIBSERV LIBGRP
     ▢server.xml         640  LIBSERV LIBGRP
     ▢server.env         640  LIBSERV LIBGRP
      /workarea          750  LIBSERV LIBGRP
```

It will create the directories and files under the *<WLP_USER_DIR>* and assign ownership based on the ID and Group that created the server

There are a few potential issues with this in a production setting:

- If you have multiple people with a need to change configuration files, do you share the password of LIBSERV? (answer: **no**)
  Sharing passwords is a bad practice. Better to take advantage SAF SURROGAT so permitted users can switch to the owning ID so they can make changes
- If you have multiple people with a need to read or update configuration files, do you simply connect them to LIBGRP? (answer: **no**)
  The owner group may be granted access to other resources (on z/OS SAF profiles notably: SERVER) and you do not want others inheriting that. Better to make the configuration group be something different from the owner group and grant READ through that group.

CWWKB0121I: The server process UMASK value is set to 0000

- sets permission bit for new files deployed using the RESTful APIs to rw-rw-rw (666 x'OR 000)

Default server configuration directories and files

ID=**LIBSERV**
Group=**LIBGRP**

```
export JAVA_HOME=<path_to_64_bit_Java>
export WLP_USER_DIR=/var/zosconnect
./server create zceesrv1
```

```
/var/zosconnect             754  LIBSERV LIBGRP
  /servers                  754  LIBSERV LIBGRP
    /zceesrv1               754  LIBSERV LIBGRP
      /logs                 774  LIBSERV LIBGRP
        messages.log        644  LIBSERV LIBGRP
      /resources            750  LIBSERV LIBGRP
        /zosconnect         750  LIBSERV LIBGRP
          /apis             760  LIBSERV LIBGRP
          /apiRequesters    760  LIBSERV LIBGRP
          /rules            760  LIBSERV LIBGRP
          /services         760  LIBSERV LIBGRP
     server.xml             460  LIBSERV ADMGRP
     server.env             460  LIBSERV ADMGRP
      /workarea             750  LIBSERV LIBGRP
```

It will create the directories and files under the *<WLP_USER_DIR>* and assign ownership based on the ID and Group that created the server

There are a few potential issues with this in a production setting:

- If you have multiple people with a need to change configuration files, do you share the password of LIBSERV? (answer: **no**)
  Sharing passwords is a bad practice. Better to take advantage SAF SURROGAT so permitted users can switch to the owning ID so they can make changes
- If you have multiple people with a need to read or update configuration files, do you simply connect them to LIBGRP? (answer: **no**)
  The owner group may be granted access to other resources (on z/OS SAF profiles notably: SERVER) and you do not want others inheriting that. Better to make the configuration group be something different from the owner group and grant READ through that group.

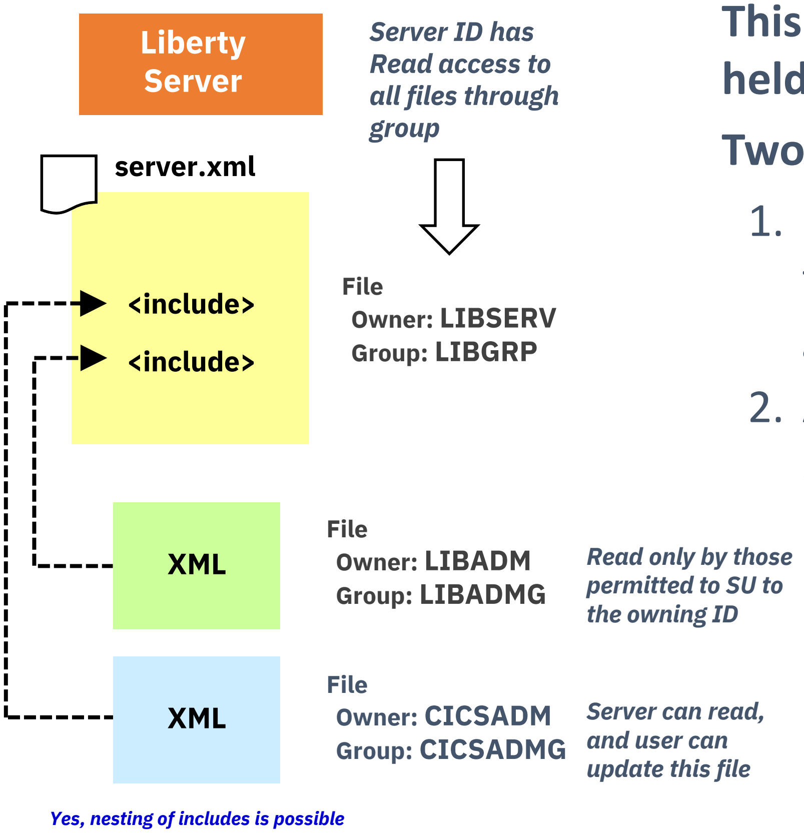
CWWKB0121I: The server process UMASK value is set to 0000

- sets permission bit for new files deployed using the RESTful artifacts to rw-rw-rw (666 x'OR 000)

~~Often you may be tempted to use command ***chmod –R 777***~~ *

Access for Owner, Group, Others depend on UID and GID as stored with the directory or file, not the actual SAF identity or group. This has implications when moving entire filesystems from one LPAR to another using utility ADRDSSU

Liberty supports server XML “include” file processing



Server ID has Read access to all files through group

Read only by those permitted to SU to the owning ID

Server can read, and user can update this file

Yes, nesting of includes is possible

This allows portions of the configuration to be held in files outside the main server.xml file

Two primary uses:

1. Hold sensitive configuration information in file that is READ to select people, but not the read group
2. Allow a user to update their portion of the server configuration, but not other parts of it

For the second use-case it is important to insure the user can not override configuration in the main XML. Use the "onConflict" tag in the <include> element:

<include location="myIncludeFile.xml" onConflict="IGNORE"/>

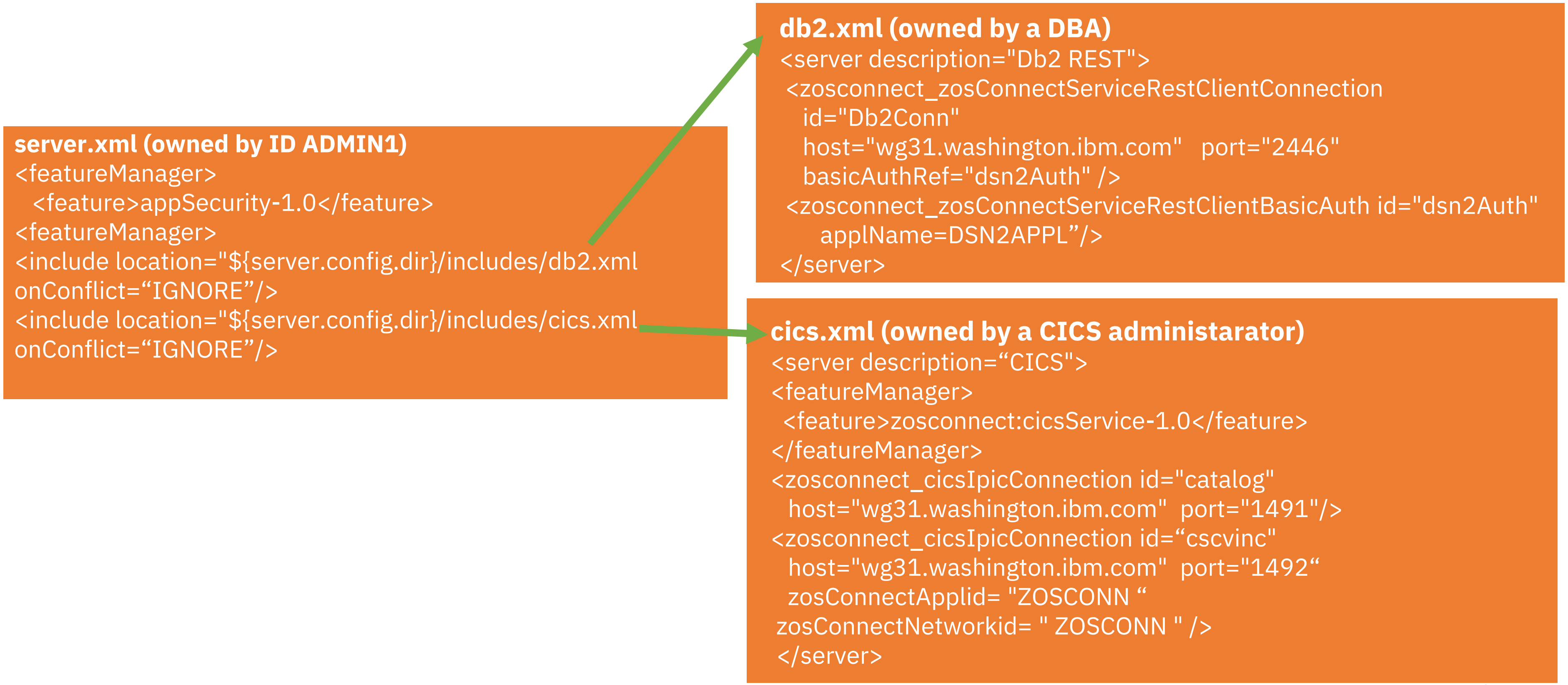
This tells Liberty to ignore XML elements in include file that are also found in the main server.xml

It does not prevent them from injecting configuration elements not found in the main server.xml. If there is a concern about that, don't use include processing.

For example, using “include” to manage the server XML

- Setup a server.xml using ‘include’ statements and allow application admins to write to those included files, but not the server.xml itself.
- Control what configuration can be overridden in included files using the ‘onConflict’ option provided with the include element (see Ignore, Replace, Merge).

https://www.ibm.com/support/knowledgecenter/en/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_config_include.html



server.xml (owned by ID ADMIN1)

```
<featureManager>
  <feature>appSecurity-1.0</feature>
<featureManager>
<include location="${server.config.dir}/includes/db2.xml
onConflict=“IGNORE”/>
<include location="${server.config.dir}/includes/cics.xml
onConflict=“IGNORE”/>
```

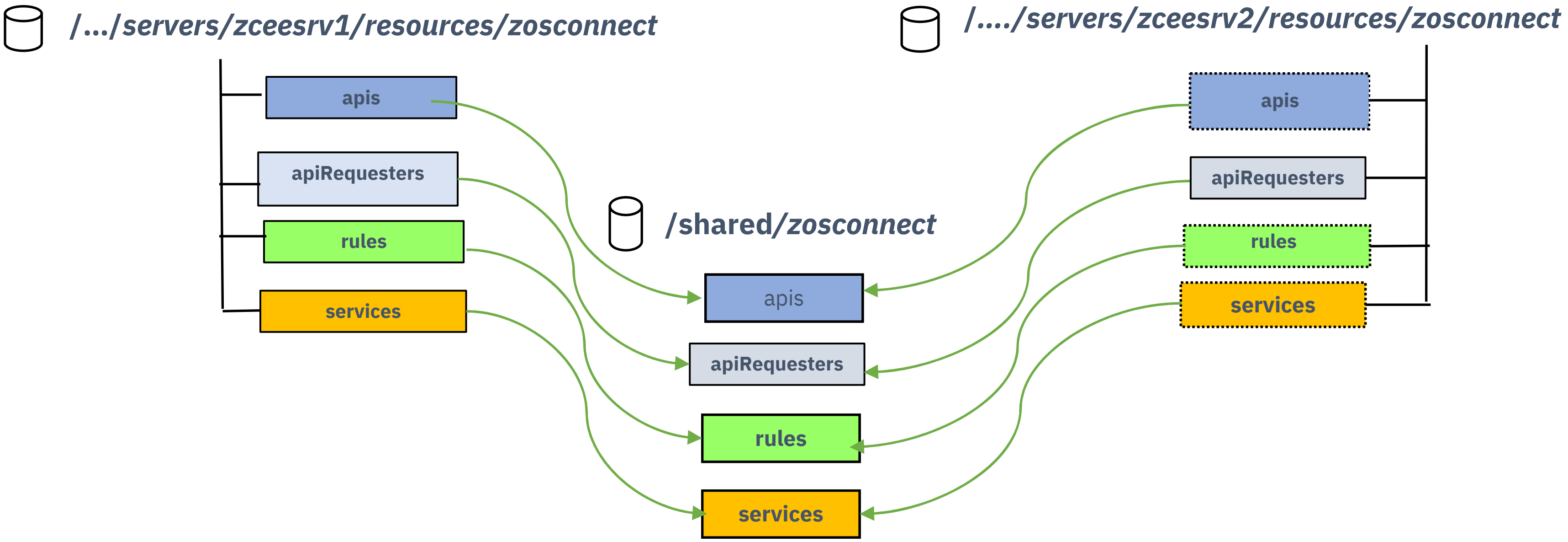
db2.xml (owned by a DBA)

```
<server description="Db2 REST">
 <zosconnect_zosConnectServiceRestClientConnection
   id="Db2Conn"
   host="wg31.washington.ibm.com"   port="2446"
   basicAuthRef="dsn2Auth" />
 <zosconnect_zosConnectServiceRestClientBasicAuth id="dsn2Auth"
     applName=DSN2APPL”/>
</server>
```

cics.xml (owned by a CICS administarator)

```
<server description=“CICS">
<featureManager>
 <feature>zosconnect:cicsService-1.0</feature>
</featureManager>
<zosconnect_cicsIpicConnection id="catalog"
  host="wg31.washington.ibm.com"  port="1491"/>
<zosconnect_cicsIpicConnection id=“cscvinc"
  host="wg31.washington.ibm.com"  port="1492“
  zosConnectApplid= "ZOSCONN “
 zosConnectNetworkid= " ZOSCONN " />
 </server>
```

Remember the use of symbolic links to share artifacts

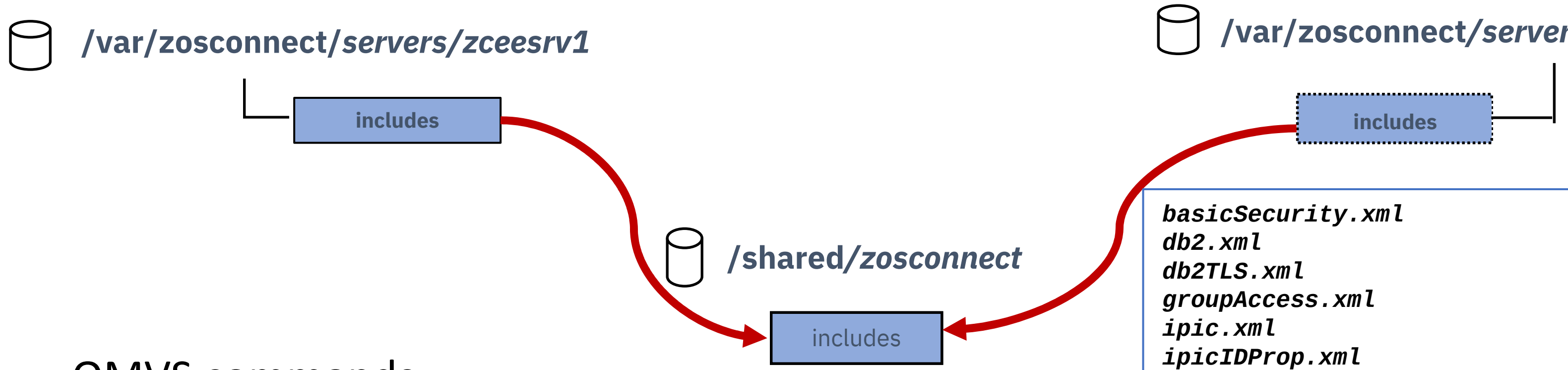


Use symbolic links

```
cd /shared/zosconnect
ln -s /. . ./servers/zceesrv1/resources/zosconnect/apis apis
ln -s /. . ./servers/zceesrv2/resources/zosconnect/apis apis

ln -s /. . ./servers/zceesrv1/resources/zosconnect/apiRequesters apiRequesters
ln -s /. . ./servers/zceesrv2/resources/zosconnect/apiRequesters apiRequesters

ln -s /. . ./servers/zceesrv1/resources/zosconnect/rules rules
ln -s /. . ./servers/zceesrv2/resources/zosconnect/rules rules

ln -s /. . ./servers/zceesrv1/resources/zosconnect/services services
ln -s /. . ./servers/zceesrv2/resources/zosconnect/services services
```

```
F ZCEESRV1,ZCON,REFRESH
F ZCEESRV2,ZCON,REFRESH
```

Sharing security XML configuration files between servers


/var/zosconnect/servers/zceesrv1
includes
/var/zosconnect/servers/zceesrv2
includes
/shared/zosconnect
includes

```
basicSecurity.xml
db2.xml
db2TLS.xml
groupAccess.xml
ipic.xml
ipicIDProp.xml
keyring.xml
keystore.xml
keyringMutual.xml
keyringOutboundMutual.xml
safSecurity.xml
```

OMVS commands

```
Use symbolic links

cd /shared/zosconnect
ln -s /var/zosconnect/servers/zceesrv1/includes includes
ln -s /var/zosconnect/servers/zceesrv2/includes includes
```

Each server.xml file includes these statements

```
<include location="${server.config.dir}/includes/safSecurity.xml"/>
<include location="${server.config.dir}/includes/ipicIDProp.xml"/>
<include location="${server.config.dir}/includes/keyringOutboundMutual.xml”/>
<include location="${server.config.dir}/includes/groupAccess.xml"/>
<include location="${server.config.dir}/includes/shared.xml"/>
<include location="${server.config.dir}/includes/oauth.xml"/>
```

These includes can be in a common file which then can be shared among multiple servers

F BAQSTRT,REFRESH,CONFIG

Server XML configuration files – bootstrap.properties file

zceesrv1’s bootstrap.properties

```
httpPort=9080
httpsPort=9443
ipicPort=1491
cicsHost=wg31.washington.ibm.com
network=ZOSCONN1
applid=ZOSCONN1
```

zceesrv2’s bootstrap.properties

```
httpPort=9090
httpsPort=9453
ipicPort=1492
cicsHost=wg31.washington.ibm.com
network=ZOSCONN2
applid=ZOSCONN2
```

server.xml

```
 <!-- To access this server from a remote client add a host attribute to the following
element, e.g. host="*" -->
    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="${httpPort}"
                  httpsPort="${httpsPort}" />
```

ipicIDProp.xml

```
<zosconnect_cicsIpicConnection id="catalog"
  host="${cicsHost}" port="${ipicPort}"
  zosConnectNetworkid="${network}"  zosConnectApplid="${applid}"/>

<zosconnect_cicsIpicConnection id="cscvinc"
 host="${cicsHost}" port="${ipicPort}"
 zosConnectNetworkid="${network}"  zosConnectApplid="${applid}"/>

<zosconnect_cicsIpicConnection id="miniloan"
 host="${cicsHost}" port="${ipicPort}"
 zosConnectNetworkid="${network}"  zosConnectApplid="${applid}"/>
```

BAQR0660E Failed to establish CICS connection IPIC Capability exchange error

Sharing security XML configuration files – variables

variablesBBG.xml

```
<variable name= "unauthenticatedUser" value= "WSGUEST" />
<variable name="profilePrefix" value=“BBGZDFLT" />
```

variablesEMJ.xml

```
<variable name= "unauthenticatedUser" value=“ZCGUEST" />
<variable name="profilePrefix" value="EMJZDFLT" />
```

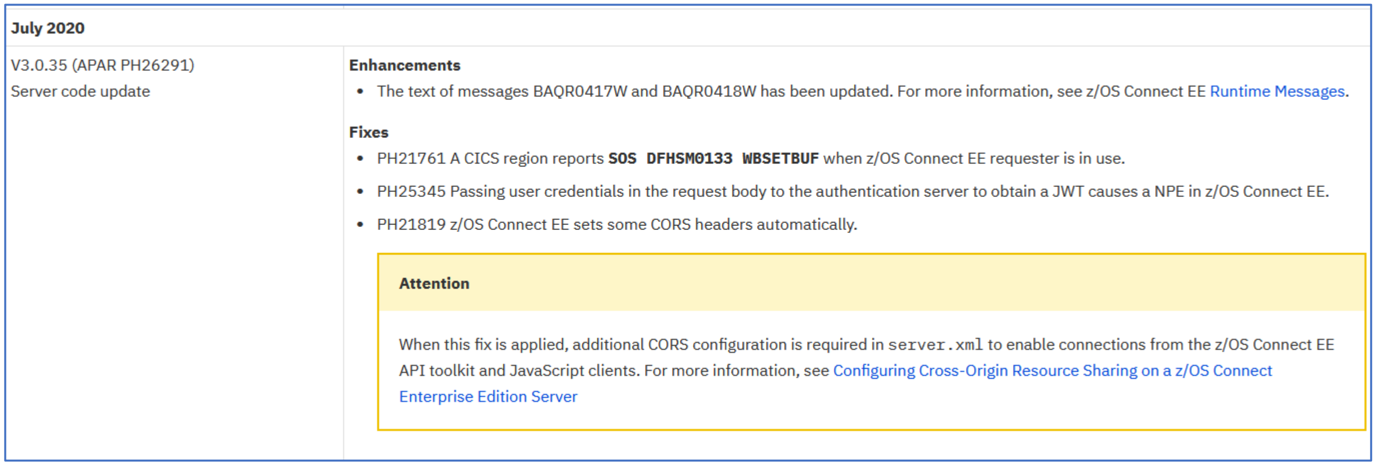
safSecurity.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="SAF security">

    <!-- Enable features -->
    <featureManager>
        <feature>appSecurity-2.0</feature>
        <feature>zosSecurity-1.0</feature>
    </featureManager>

    <webAppSecurity allowFailOverToBasicAuth="true" />

    <safRegistry id="saf" />
    <safAuthorization racRouteLog="ASIS" />
    <safCredentials unauthenticatedUser="${unauthenticatedUser}"
       profilePrefix="${profilePrefix}"/>

</server>
```

server.xml

```
<server description="new server">
<include location="${server.config.dir}/includes/safSecurity.xml"/>
<include location="${server.config.dir}/variablesEMJ.xml"/>

     <!-- Enable features -->
     <featureManager>
         <feature>zosconnect:zosConnect-2.0</feature>
         <feature>zosconnect:zosConnectCommands-1.0</feature>
     </featureManager>
```

A practical example-PTF V3.0.35 included a CORS update



July 2020

V3.0.35 (APAR PH26291)
Server code update

Enhancements

- The text of messages BAQR0417W and BAQR0418W has been updated. For more information, see z/OS Connect EE Runtime Messages.

Fixes

- PH21761 A CICS region reports **SOS DFHSM0133 WBSETBUF** when z/OS Connect EE requester is in use.
- PH25345 Passing user credentials in the request body to the authentication server to obtain a JWT causes a NPE in z/OS Connect EE.
- PH21819 z/OS Connect EE sets some CORS headers automatically.

Attention

When this fix is applied, additional CORS configuration is required in `server.xml` to enable connections from the z/OS Connect EE API toolkit and JavaScript clients. For more information, see Configuring Cross-Origin Resource Sharing on a z/OS Connect Enterprise Edition Server

cors.xml

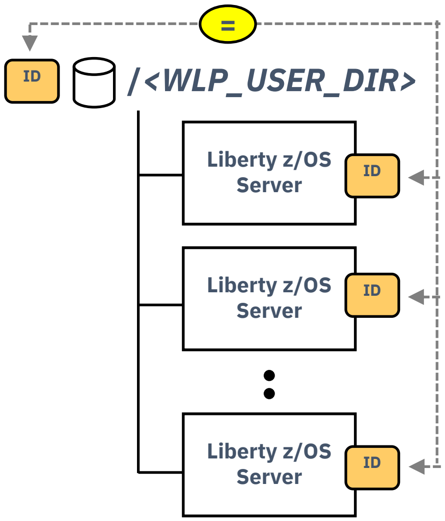
```
<?xml version="1.0" encoding="UTF-8"?>
<server description="CORS entries">

    <!-- add cors to allow cross origin access, e.g. when using swagger doc from zOS Connect Enterprise
Edition  -->
    <cors id="defaultCORSConfig"
          domain="/"
          allowedOrigins="*"
          allowedMethods="GET, POST, PUT, DELETE, OPTIONS"
          allowedHeaders="Origin, Content-Type, Authorization, Cache-Control, Expires, Pragma"
          allowCredentials="true"
          maxAge="3600"/>

</server>
```

server.xml

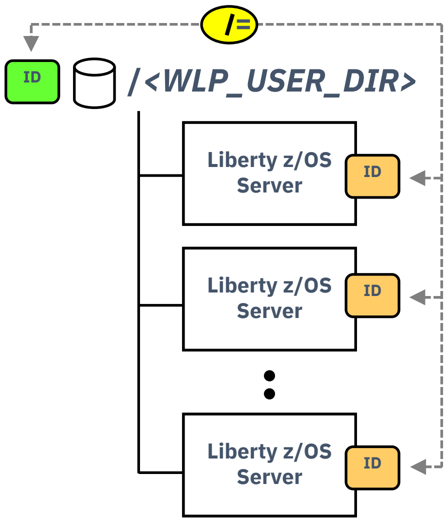
```
<include location="${server.config.dir}/includes/cors.xml"/>
```

z/OS Security – Range of options – Started Task IDs

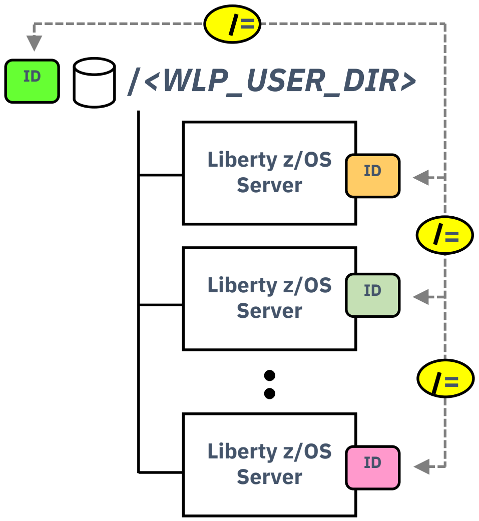
On z/OS, the best practice for Liberty servers in production is that they run as ‘Started Tasks’ (STCs).



- Multiple servers
- All have same STC ID
- STC ID = File Owner ID



- Multiple servers
- All have same STC ID
- STC ID ǂ File Owner ID



- Multiple servers
- Different STC IDs
- STC IDs ǂ File Owner ID

Should all servers sharing WLP_USER_DIR share the same STC ID?
It is a matter of the degree of identity isolation that is required

z/OS : Assigning ID to started tasks: SAF STARTED

The first question here is whether you wish to have a common started task ID that is shared among servers, or if you wish each server to have a unique ID

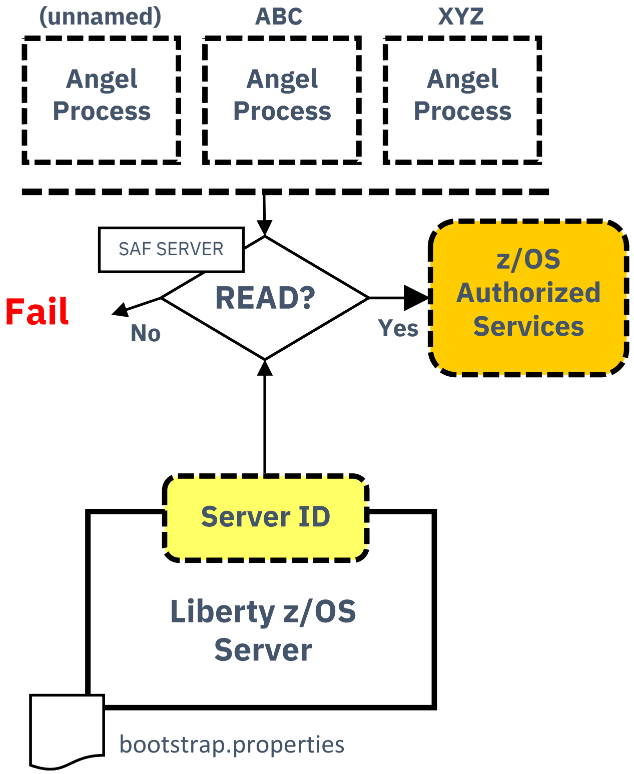
Then the second question is whether servers under a WLP_USER_DIR will share a common JCL start proc, or use unique start procs for each server

	Common ID	*Unique IDs*
Common Proc	START ZCEEPROC,JOBNAME=<server>,PARMS='<server>' STATED ZCEEPROC.* 1	START ZCEEPROC,JOBNAME=<jobname>,PARMS='<server>' STARTED ZCEEPROC.<jobname> 3
Unique Proc per server	START ZCEESRV,*JOBNAME*=<server>,*PARMS*='<server>' STARTED ZCEE*.* 2	START ZCEESRV1,JOBNAME=<server>,PARMS='<server>' STARTED ZCEESRV1.* 4

Note: Using unique JCL procedure eliminates the need to specify JOBNAME or PARMS on the start commands

It's possible to use a combination of the above, even under the same WLP_USER_DIR. So there's no "one best answer" here. What's best is what's best for you.

z/OS : The Angel process – what is this about?



(unnamed)
ABC
XYZ
Angel Process
Angel Process
Angel Process
SAF SERVER
Fail
No
READ?
Yes
z/OS Authorized Services
Server ID
Liberty z/OS Server
bootstrap.properties

```
com.ibm.ws.zos.core.angelRequired=true
com.ibm.ws.zos.core.angelName=<name>
```

The Angel Process is a started task that is used to protect access to z/OS authorized services. This is done with SAF SERVER profiles.

- Authorized services include: WOLA, SAF, WLM, RRS, DUMP
- The ability to start multiple Angel processes on an LPAR was introduced in 16.0.0.4. This is called "Named Angels". It provides a way to separate Angel usage between Liberty servers:
  - An Angel process can be started with a NAME='*<name>*' parameter (or it can be started as a "default" without a name). The name may be up to 54 characters.
  - Liberty servers can be pointed at a specific Angel with a bootstrap property

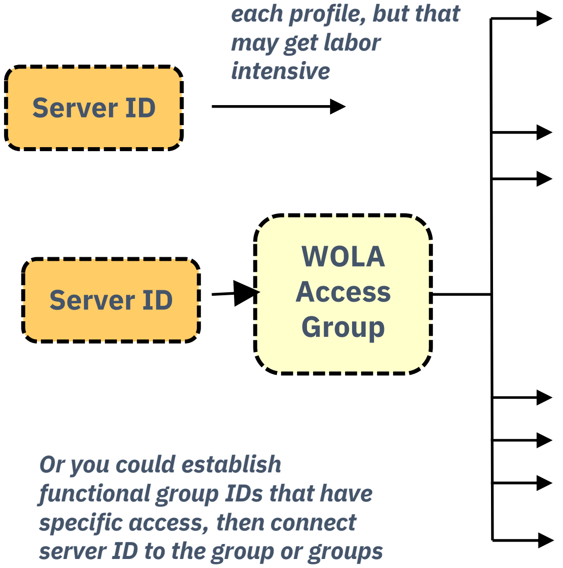
Best practice:

- **You may create separate named Angels for isolation of Test and Production, but do not take this practice too far. A few Angels, yes; dozens, no.**
- **Establish automation routines to start the Angels at IPL**
- **Grant SAF GROUP access to the SERVER profiles, then connect server IDs as needed**

Current list of Liberty Features

https://www.ibm.com/support/knowledgecenter/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/rwlp_feat.html

z/OS : SAF SERVER profiles related to the Angel



You can grant server IDs direct READ to each profile, but that may get labor intensive

Or you could establish functional group IDs that have specific access, then connect server ID to the group or groups to get the access.

BBG.ANGEL	enables access to the unnamed Angel process
BBG.ANGEL.***<angel_name>***	enables access to a specific named Angel
BBG.AUTHMOD.BBGZSAFM	enables access to authorized services
BBG.AUTHMOD.BBGZSCFM	enables loading of authorized client services
BBG.AUTHMOD.BBGZSAFM.SAFCRED*	enables use of SAF authorized services
BBG.AUTHMOD.BBGZSAFM.ZOSWLM	enables use of WLM authorized services
BBG.AUTHMOD.BBGZSAFM.TXRRS	enables use of RRS services (transaction)
BBG.AUTHMOD.BBGZSAFM.ZOSDUMP	enables use of SVCDUMP services
BBG.AUTHMOD.BBGZSAFM.LOCALCOM	enables use of WOLA
BBG.AUTHMOD.BBGZSAFM.WOLA	
BBG.AUTHMOD.BBGZSCFM.WOLA	
BBG.AUTHMOD.BBGZSAFM.PRODMGR	enables use of IFAUSAGE services
BBG.AUTHMOD.BBGZSAFM.ZOSAIO	enables use TCP asynchronous I/O services
*BBG.SECPFX.***BBGZDFLT***	enables use of the profilePrefix for SAF authentication request

- **Establish all the SERVER profiles ahead of time. Existence of profile does not grant access; READ to it does.**
- **Determine what access a server needs and grant only that; check "is available" messages in messages.log to verify**

Tech/Tip: The SAFLOG parameters was added in a recent Liberty drop. If this parameter is set to Y, additional security related messages will be written to the JES messages and console if a Liberty does not have authorization to use an angel-controlled privileged function. See URL https://www.ibm.com/support/knowledgecenter/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/rwlp_newinrelease.html

Tech/Tip: z/OS : SAF SURROGAT Resources

RACF Surrogate access allows a designated administrative identity the ability to invoke commands and perform functions as if they were running under the identity that will be used for the z/OS Connect server started task. This may be useful because identities associated with started task are normally restricted and cannot be used for accessing TSO or OMVS shells,

Use the following examples as guides and create the surrogate resources and permit access. In these examples, ***LIBSERV*** represents the RACF identity under which the z/OS Connect server will be running and ***adminUser*** represent the administrative RACF identity.

Define a SURROGAT profile for the server's SAF identity

```
RDEFINE SURROGAT BPX.SRV.LIBSERV
```

Define a SURROGAT submit profile to allow job submission as the server's SAF identity

```
RDEFINE SURROGAT LIBSERV.SUBMIT
```

Permit an administrative identity to act as a surrogate of the Liberty task identity

```
PERMIT BPX.SRV.LIBSERV CLASS(SURROGAT) ID(adminGrp) ACC(READ)
PERMIT LIBSERV.SUBMIT CLASS(SURROGAT) ID(adminGrp) ACC(READ)
```

Refresh the SURROGAT in storage profiles

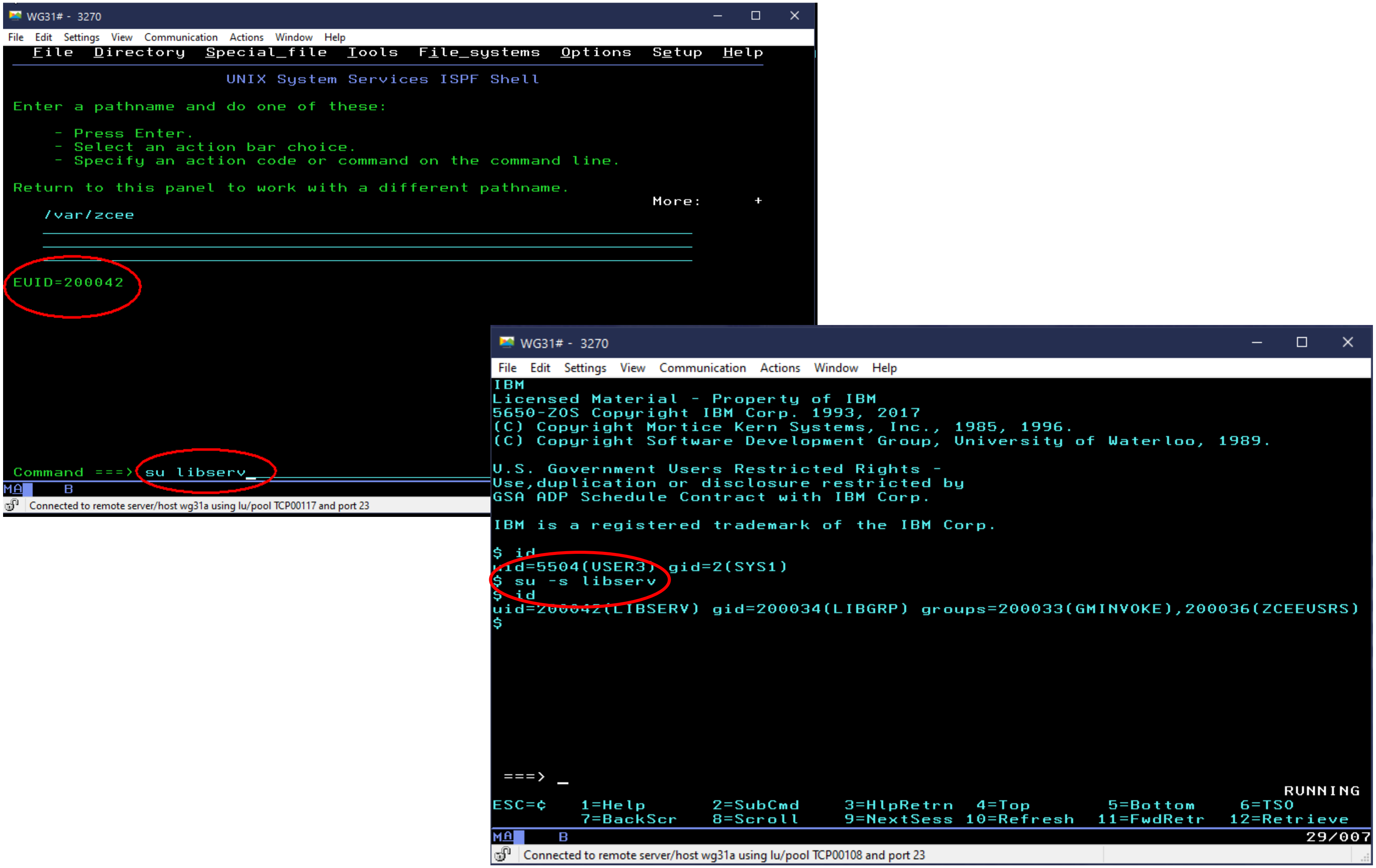
```
SETROPTS RACLIST(SURROGAT) REFRESH
```

Now user identity *adminUser* can submit JCL with the *USER=LIBSERV* parameter on the job card or use the OMVS switch user command (*su –s LIBSERV*) to execute OMVS scripts or commands as LIBSERV.

Tech/Tip: z/OS : A JCL example of using SURROGAT access

```
//MYSERVER JOB 'ZCEE',CLASS=A,REGION=0M,NOTIFY=&SYSUID,USER=LIBSERV
//****************************************************************
//*  SET SYMBOLS
//****************************************************************
//EXPORT EXPORT SYMLIST=(*)
// SET JAVAHOME='/usr/lpp/java/J8.0_64'
// SET ZCEEPATH='/usr/lpp/IBM/zosconnect/v3r0'
// SET SERVER= 'baqstrt'
// SET TEMPLATE='zosconnect:default'
// SET WLPUSER='/var/zosconnect'
// SET USER='LIBSERV'
// SET GROUP='LIBGRP'
//****************************************************************
//*  Step ZCEESRVR - Use the zosconnect command to create a server
//****************************************************************
//ZCEESRVR EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSERR   DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//SYSTSIN  DD *,SYMBOLS=EXECSYS
 BPXBATCH SH +
 export JAVA_HOME=&JAVAHOME; +
 export WLP_USER_DIR=&WLPUSER; +
 &ZCEEPATH/bin/zosconnect create &SERVER +
 --template=&TEMPLATE
```

Tech/Tip: z/OS : An ISPF/OMVS example of using SURROGAT access



```
 File  Directory  Special_file  Tools  File_systems  Options  Setup  Help

                    UNIX System Services ISPF Shell

Enter a pathname and do one of these:

    - Press Enter.
    - Select an action bar choice.
    - Specify an action code or command on the line.

Return to this panel to work with a different pathname.
                                                              More:     +
   /var/zcee

EUID=200042

Command ===> su libserv
```

Connected to remote server/host wg31a using lu/pool TCP00117 and port 23

```
IBM
Licensed Material - Property of IBM
5650-ZOS Copyright IBM Corp. 1993, 2017
(C) Copyright Mortice Kern Systems, Inc., 1985, 1996.
(C) Copyright Software Development Group, University of Waterloo, 1989.

U.S. Government Users Restricted Rights -
Use,duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

$ id
uid=5504(USER3) gid=2(SYS1)
$ su -s libserv
$ id
uid=200042(LIBSERV) gid=200034(LIBGRP) groups=200033(GMINVOKE),200036(ZCEEUSRS)
$




 ===> _
                                                                  RUNNING
ESC=¢    1=Help       2=SubCmd     3=HlpRetrn   4=Top       5=Bottom     6=TSO
         7=BackScr    8=Scroll     9=NextSess  10=Refresh  11=FwdRetr   12=Retrieve
                                                                          29/007
```

Connected to remote server/host wg31a using lu/pool TCP00108 and port 23

Tech/Tip: z/OS : SAF UNIXPRIV/FACILITY Resources

An alternative to using a surrogate access is to permit the identity under which the customization will be done to enhanced Unix privileges. Specially, permitting the identity to Unix privileges SUPERUSER.FILESYS, SUPERUSER.FILESYS.CHANGEPERMS and SUPERUSER.FILESYS.CHOWN.

- *Permit an administrative identity to write to any local directory or file*
  PERMIT SUPERUSER.FILESYS CLASS(UNIXPRIV)
  ID(adminUser) ACC(CONTROL)
- *Permit an administrative identity to change permission bit of any local directory or file*
  PERMIT SUPERUSER.FILESYS.CHANGEPERMS CLASS(UNIXPRIV)
  ID(adminUser) ACC(READ)
- *Permit an administrative identity to change the ownership of any directory or file*
  PERMIT SUPERUSER.FILESYS.CHOWN CLASS(UNIXPRIV)
  ID(*adminUser*) ACC(READ)
- *Permit an administrative identity switch to root (su –s root)*
  PERMIT BPX.SUPERUSER CLASS(FACILITY) ID(adminUser) ACC(READ)

- *Refresh the UNIXPRIV and/or FACILITY instorage profiles*
  SETROPTS RACLIST(UNIXPRIV,FACILITY) REFRESH

Do not use these commands if you do not understand the implications.

https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.4.0/com.ibm.zos.v2r4.bpxb200/usspriv.htm

Tech/Tip: z/OS : SAF APPL and EJBRole Resources

Connect z/OS Connect users to a common group

```
CONNECT (FRED,USER1,JOHNSON) GROUP(ZCEEUSRS)
```

Define a APPL profile for the server's SAF profilePrefix and permit access

```
RDEFINE APPL BBGZDFLT UACC(NONE) OWNER(SYS1)
PERMIT BBGZDFLT CLASS(APPL) ACCESS(READ) +
     ID(WSGUEST#, ZCEEUSRS)
SETROPTS RACLIST(APPL) REFRESH
```

Define an EJBROLE profile for the server's SAF profilePrefix and permit access

```
RDEFINE EJBROLE BBGZDFLT.zos.connect.access.roles.zosConnectAccess +
     OWNER(SYS1) UACC(NONE)
PERMIT BBGZDFLT.zos.connect.access.roles.zosConnectAccess +
   CLASS(EJBROLE) ID(ZCEEUSRS) ACCESS(READ)
```

Refresh the SURROGAT in storage profiles

```
SETROPTS RACLIST(EJBROLE) REFRESH
```

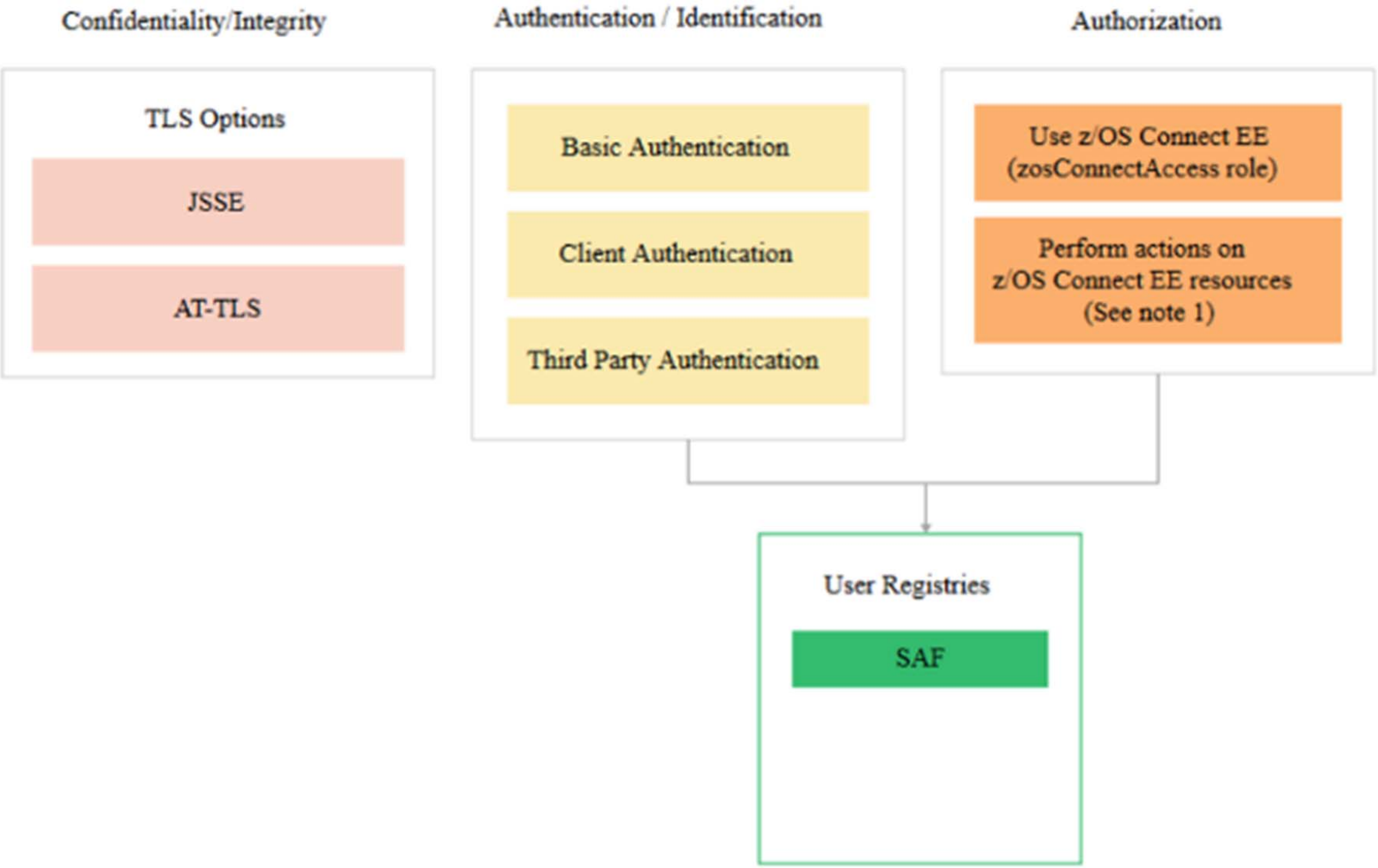
```
<safCredentials unauthenticatedUser="WSGUEST" profilePrefix="BBGZDFLT" />
```

# https://www.ibm.com/support/knowledgecenter/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_config_security_saf.html

z/OS Connect Security

Overview

Liberty and z/OS Connect EE security options


Confidentiality/Integrity
TLS Options
JSSE
AT-TLS
Authentication / Identification
Basic Authentication
Client Authentication
Third Party Authentication
Authorization
Use z/OS Connect EE (zosConnectAccess role)
Perform actions on z/OS Connect EE resources (See note 1)
User Registries
SAF

The actions which can be controlled by authorization are deploying, querying, updating, starting, stopping and deleting of APIs, services and API requesters.

z/OS Connect Security server XML Configuration

```
   <zosconnect_zosConnectManager
            requireAuth="true"
            requireSecure="true"/>

   <zosconnect_zosConnectAPIs>
      <zosConnectAPI name="catalog"
            requireAuth="true"
            requireSecure="true"/>
   </zosconnect_zosConnectAPIs>

   <zosconnect_services>
     <service id="selectByEmployee"
         name="selectEmployee"
         requireAuth="true"
         requireSecure="true"/>
  </zosconnect_services>

  <zosconnect_apiRequesters>
      requireAuth="true"
      <apiRequester name="cscvincapi_1.0.0"
          requireAuth="true"
          requireSecure="true"/>
   </zosconnect_apiRequesters>

   <zosconnect_zosconnect_service
     id="selectByEmployee"
     name="selectEmployee"
     requireAuth="true"
     requireSecure="true"/>
```

Globally, requires that users specify security credentials to be authenticated order to access APIs, services and API requesters, unless overridden on the specific resource definitions.

Requires that users specify security credentials to be authenticated in order to access the API.

Requires that users specify security credentials to be authenticated in order to access the service. This attribute is only relevant when the service is invoked directly, otherwise only the API requireAuth attribute is relevant.

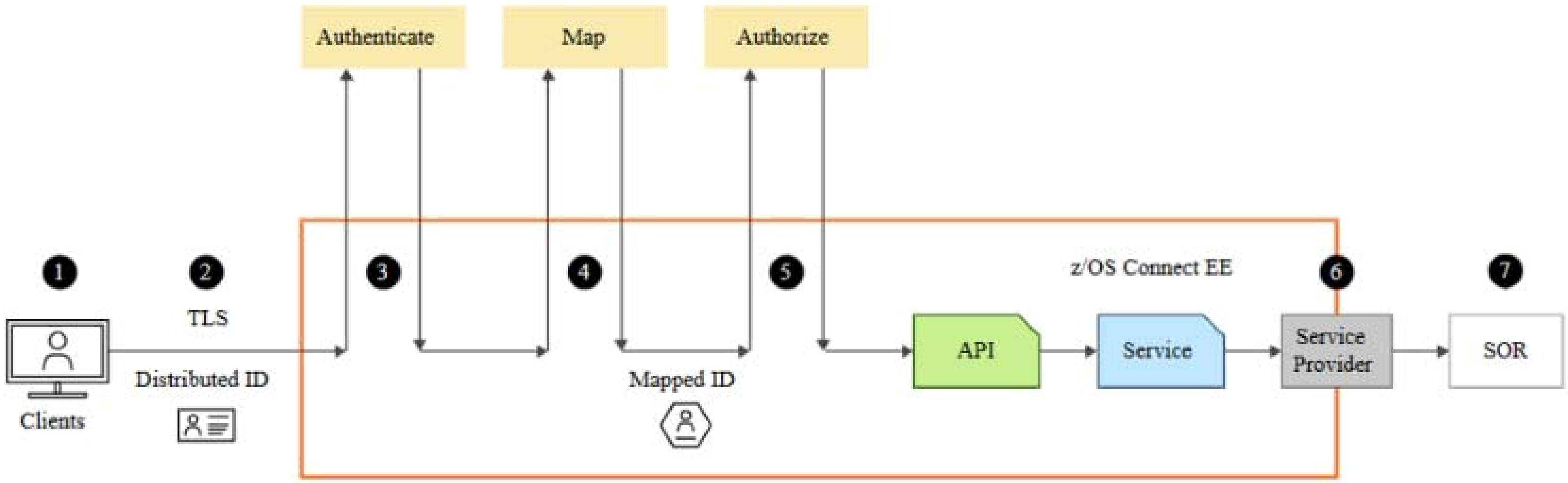
Requires that users specify security credentials to be authenticated in order to access all API requesters. If the requireAuth attribute is not set, the global setting on the zosconnect_zosConnectManager element is used instead, unless the requireAuth attribute is overridden on the specific API requester.

Requires that users specify security credentials to be authenticated in order to access the service.

Require that requests are sent overt HTTPS to access APIs, services and API requesters, unless overridden on the specific resource definitions

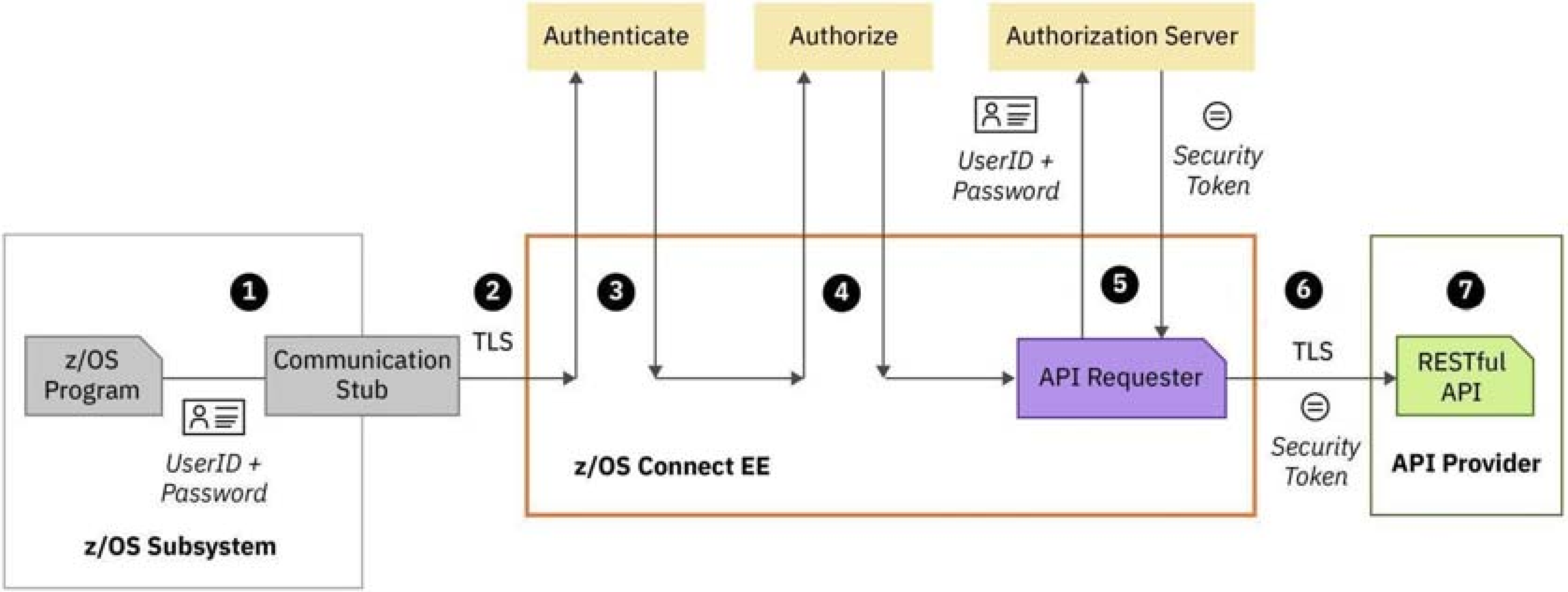
Note that these are z/OS Connect configuration elements documented in the z/OS Connect KC, i.e., the *zosconnect* prefix is the queue.

Typical z/OS Connect EE API Provider security flow



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. 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
7. The program or database request may run in the SoR under the mapped ID

Typical z/OS Connect EE API Requester security flow



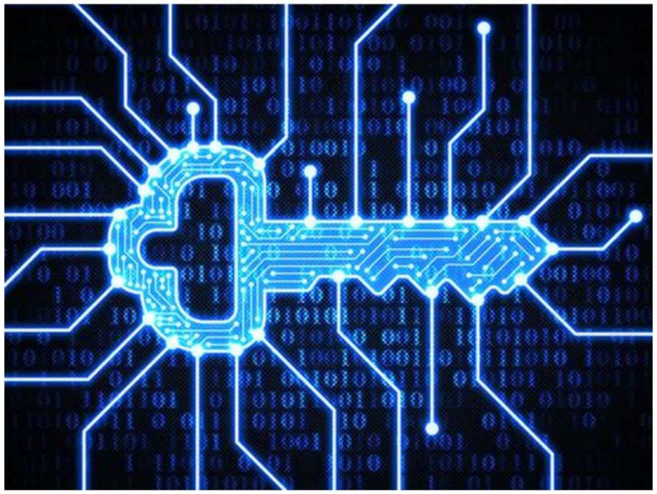
1. A user ID and password can be used for basic authentication by the z/OS Connect EE server
2. Connection between the CICS, IMS, or z/OS application and the z/OS Connect EE server can use TLS
3. Authenticate the CICS, IMS, or z/OS application.
4. Authorize the authenticated user ID to connect to z/OS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

General security terms or considerations

- Identifying who or what is requesting access (**Authentication)**
  - Basic Authentication
  - Mutual Authentication using TLS
  - Third Party Tokens

- Ensuring that the message has not been altered in transit (**Data Integrity)** and ensuring the confidentiality of the message in transit (**Encryption)**
  - TLS (encrypting messages and generating/sending a digital signature)

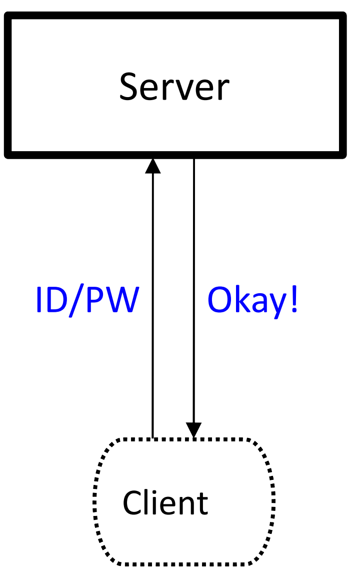
- Controlling access (**Authorization)**
  - Is the authenticated identity authorized to access to z/OS Connect
  - Is the authenticated identity authorized to access a specific API, Services, etc.



Authentication

Several different ways this can be accomplished:

Basic Authentication



Server

ID/PW

Okay!

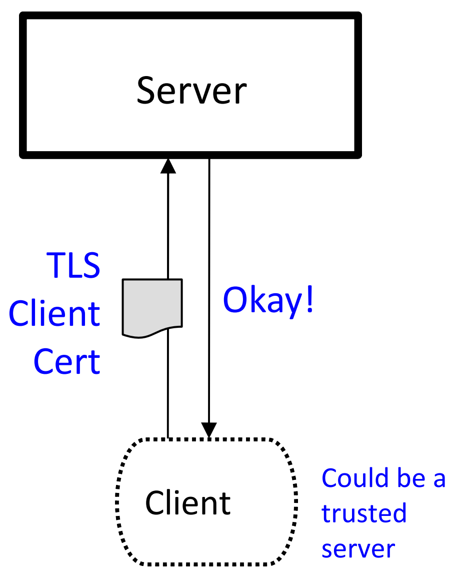
Client

Client supplies ID/PW or ID/PassTicket

Server checks registry:

- Basic (server.xml)
- SAF

Client Certificate



Server

TLS Client Cert

Okay!

Client

Could be a trusted server

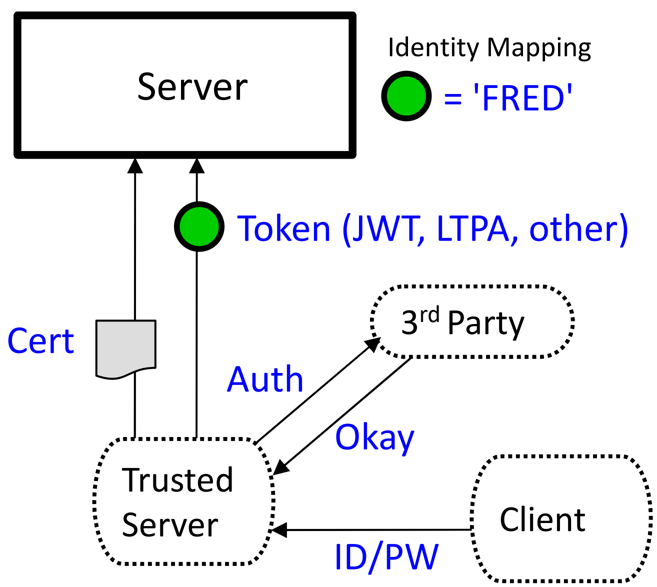
Client supplies certificate

Server validates cert and maps to an identity

Registry options:

- SAF

Third Party Authentication



Identity Mapping

● = 'FRED'

Server

● Token (JWT, LTPA, other)

Cert

Auth

Okay

3rd Party

Trusted Server

ID/PW

Client

Client authenticates to 3rd party sever

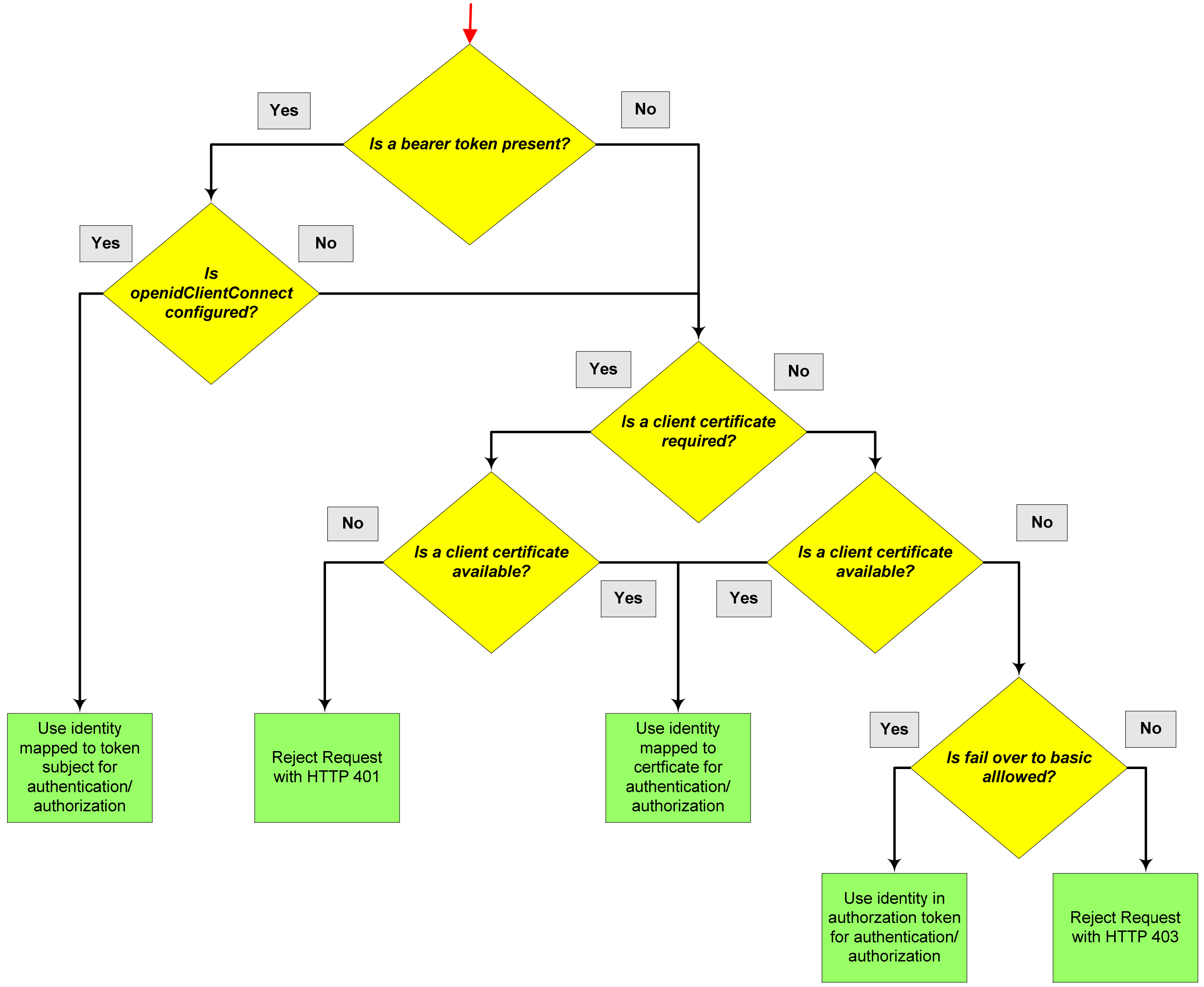
Client receives a trusted 3rd party token

Token flows to server and is mapped to an identity

Registry options:

- We may know these detail.

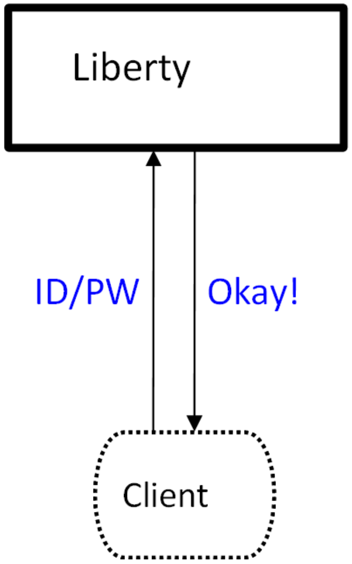
Authentication/Authorization Precedence


Is a bearer token present?
Yes
No
Is openidClientConnect configured?
Yes
No
Is a client certificate required?
Yes
No
Is a client certificate available?
No
Yes
Is a client certificate available?
Yes
No
Is fail over to basic alllowed?
Yes
No
Use identity mapped to token subject for authentication/ authorization
Reject Request with HTTP 401
Use identity mapped to certficate for authentication/ authorization
Use identity in authorzation token for authentication/ authorization
Reject Request with HTTP 403

Authentication - Basic Authentication

Several different ways this can be accomplished:

Basic Authentication

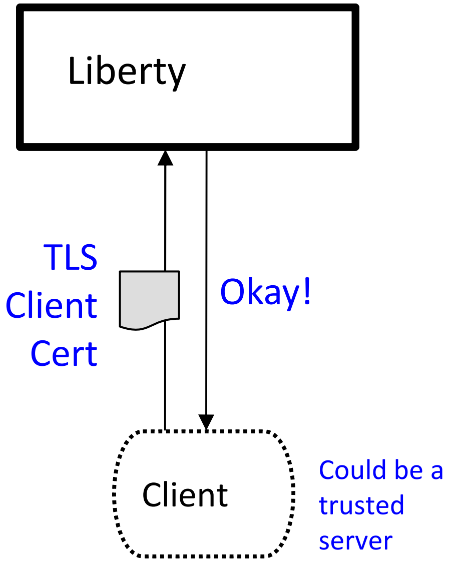


Client supplies ID/PW or ID/PassTicket

Server checks registry:

- **Basic (server.xml)**
- **SAF**

Client Certificate



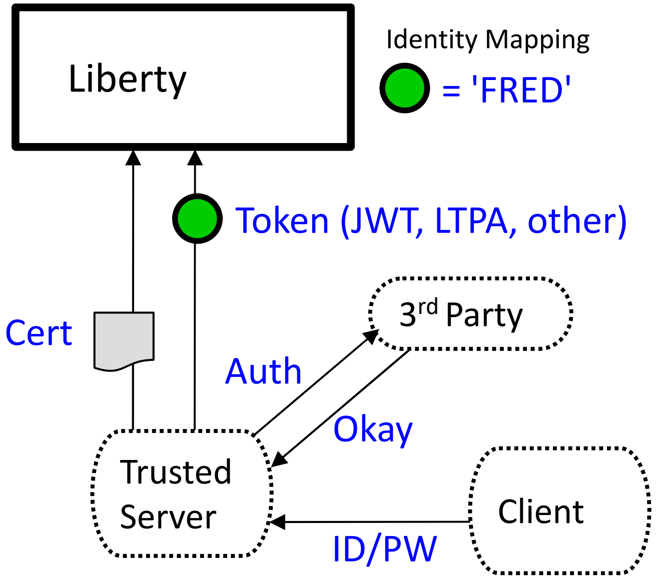
Client supplies certificate

Server validates cert and maps to an identity

Registry options:

- **SAF**

Third Party Authentication



Client authenticates to 3rd party sever

Client receives a trusted 3rd party token

Token flows to server and is mapped to an identity

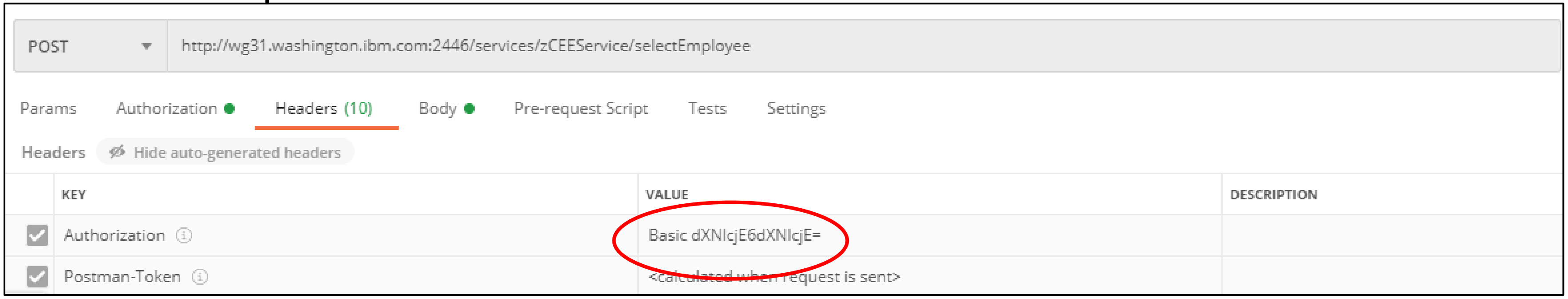
Registry options:

- **We may know these detail.**

Basic authentication – REST Client

❑ When sending a request to a Liberty server running z/OS Connect, basic authentication information (identity and password) is provided in the HTTP header in a Basic *Authorization* token with the identity and password encoded or formatted using Base64.

- An example with Postman:



POST ▾ http://wg31.washington.ibm.com:2446/services/zCEEService/selectEmployee

	KEY	VALUE	DESCRIPTION
☑	Authorization ⓘ	Basic dXNlcjE6dXNlcjE=	
☑	Postman-Token ⓘ	<calculated when request is sent>	

- An example with cURL:

curl -X GET --***user fred:fredpwd*** --header "Content-Type: application/json …

❑ Corresponding required server XML configuration:

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

     <webAppSecurity allowFailOverToBasicAuth="true" />

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

Note that these are Liberty configuration elements documented in the Liberty KC, i.e., no *zosconnect* prefix.

Basic authentication – COBOL API Requester

❑ A MVS batch or IMS requester application sends basic authentication information (identity and password) by using environment variables.

- BAQUSERNAME
- BAQPASSWORD

❑ The variables can be provided in JCL using CEEOPTS DD statement:

```
//CEEOPTS  DD *
    POSIX(ON),
    ENVAR("BAQURI=wg31.washington.ibm.com",
   "BAQPORT=9080",
   "BAQUSERNAME=USER1",
   "BAQPASSWORD=USER1")
```

❑ Or, provided by using a CEEROPT module:

```
CEEROPT CSECT
CEEROPT AMODE ANY
CEEROPT RMODE ANY
CEEXOPT POSIX=((ON),OVR),                                              +
               ENVAR=(('BAQURI=wg31.washington.ibm.com',               +
               'BAQPORT=9120',                                         +
               'BAQUSERNAME=USER1',                                    +
               'BAQPASSWORD=USER1'),OVR),                              +
               RPTOPTS=((ON),OVR)
        END
```

Tech/Tip: This is good opportunity to use a pass ticket rather than a password

A PassTicket provides an alternative to a password

- ❑ A PassTicket is generated by or for a client by using a secured sign-on key (whose value is masked or encrypted) to encrypt a valid *RACF identity* combined with the *application name* of the targeted resource. Also embedded in the PassTicket is a time stamp (based on the current Universal Coordinated Time (UCT)) which sets the time when the PassTicket will expire (usually 10 minutes).

- ❑ Access to PassTickets is managed using the RACF PTKTDATA class.

- ❑ For z/OS Connect, a RACF PassTicket can be used for basic authentication when connecting from any REST client on any platform to a z/OS Liberty server and for requests from a z/OS Connect server accessing IMS and Db2.

- ❑ ***PassTickets do not have to be generated on z/OS using RACF services.*** IBM has published the algorithm used to generate a PassTickets, see manual *z/OS Security Server RACF Macros and Interfaces, SA23-2288-40. Github has examples using Java, Python and other example are available on other sites*.

RACF resources for using PassTickets

❑ First define a PTKTDATA resource using the *applName* assigned to the target subsystem:

RDEFINE PTKTDATA applName SSIGNON(KEYMASK(keymaskValue))
APPLDATA('NO REPLAY PROTECTION')

Where:

applName is an application name assigned to the resource

kemaskValue is the value of the secured sign-on application key, a 64-bit hex value

replayProtection indicates if a pass ticket can be reused

❑ Access to using PassTickets is controlled by another PTKTDATA resource, *IRRPTAUTH.applName.identity.* UPDATE access is required. For example, to use PassTickets to access a z/OS Connect server the resources below need to be defined and access permitted.

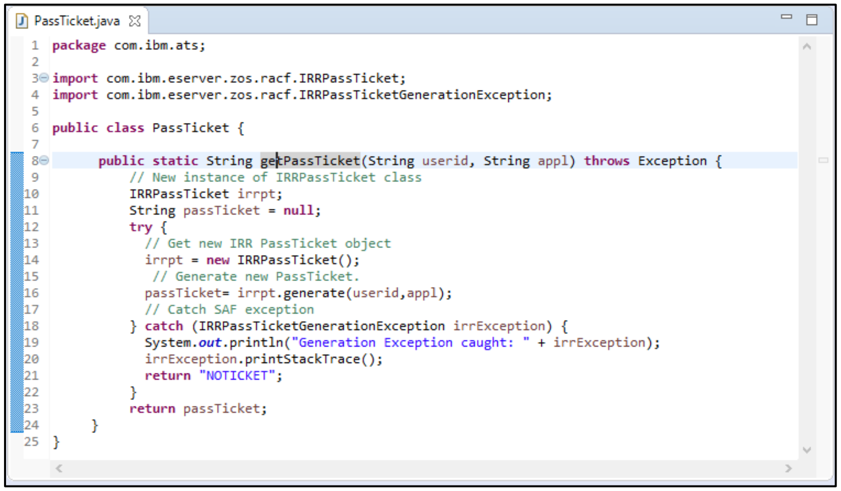
```
<safRegistry id="saf" />
   <safAuthorization racRouteLog="ASIS" />
   <safCredentials unauthenticatedUser="WSGUEST"
      profilePrefix="BBGZDFLT" />
```

```
RDEFINE PTKTDATA BBGZDFLT SSIGNON(0123456789ABCDEF))
           APPLDATA('NO REPLAY PROTECTION') UACC(NONE)
RDEFINE PTKTDATA IRRPTAUTH.BBGZDFLT.* UACC(NONE)
PERMIT IRRPTAUTH.BBGZDFLT.* ID(LIBSERV) CLASS(PTKTDATA) ACCESS(UPDATE)
PERMIT IRRPTAUTH.BBGZDFLT.USER1 ID(USER1) CLASS(PTKTDATA) ACCESS(UPDATE)
```

Generating PassTickets on z/OS

❑ On z/OS, a COBOL user applications can generate PassTickets by

- invoking the IBM provided class *com.ibm.eserver.zos.racf.IRRPassTicket*:



```java
package com.ibm.ats;

import com.ibm.eserver.zos.racf.IRRPassTicket;
import com.ibm.eserver.zos.racf.IRRPassTicketGenerationException;

public class PassTicket {

      public static String getPassTicket(String userid, String appl) throws Exception {
          // New instance of IRRPassTicket class
          IRRPassTicket irrpt;
          String passTicket = null;
          try {
            // Get new IRR PassTicket object
            irrpt = new IRRPassTicket();
             // Generate new PassTicket.
            passTicket= irrpt.generate(userid,appl);
            // Catch SAF exception
          } catch (IRRPassTicketGenerationException irrException) {
            System.out.println("Generation Exception caught: " + irrException);
            irrException.printStackTrace();
            return "NOTICKET";
          }
          return passTicket;
      }
}
```

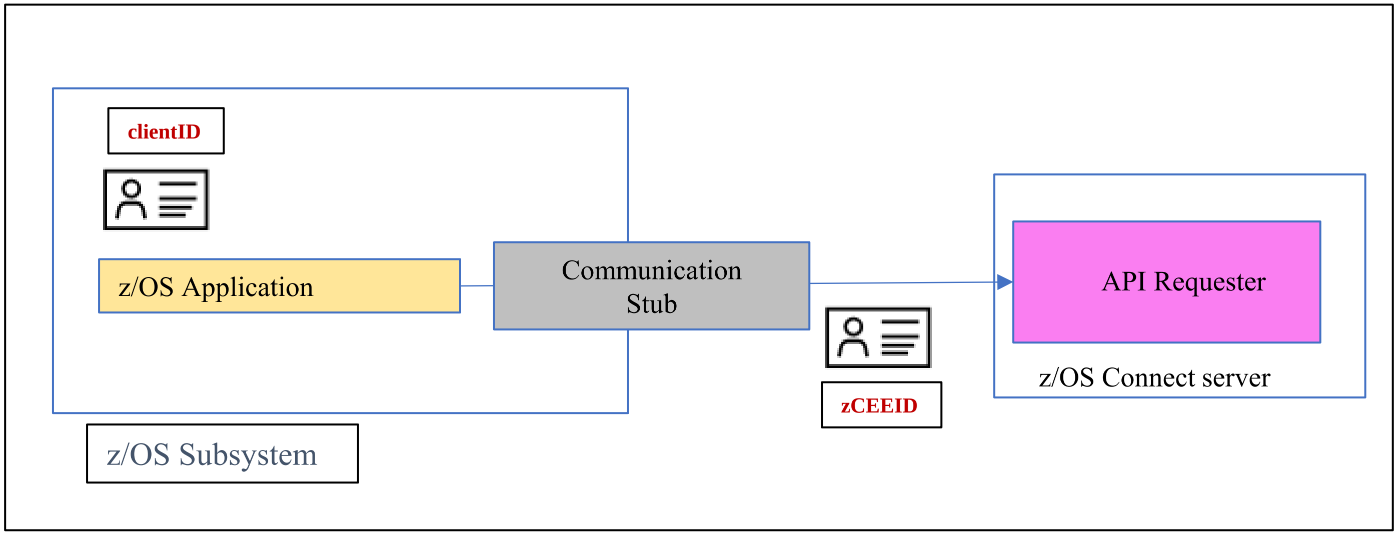
```
*-----------------------------------------------------------------*
* Invoke method GetPassTicket in Java class                       *
*      com.ibm.ats.GetPassTicket                                  *
*-----------------------------------------------------------------*
     Invoke PassTicketClass "getPassTicket"
            using by value jIdentity jAppl returning jResponse.
     Perform JavaExceptionCheck
```

- calling RACF service IRRSPK00:

```
CALL W-IRRSPK00 USING irr-workarea,
                IRR-ALET, irr-safrc,
                IRR-ALET, irr-racfrc,
                IRR-ALET, irr-racfrsn,
                IRR-ALET, irr-functionCode,
                irr-optionWord,
                IRR-PASSTICKET,
                irr-ticketOptions-ptr,
                IRR-IDENTITY,
                IRR-APPLID.
```

Examples of both are available on request

API Requester – identity assertion



zCEEID – The identity that is used for authenticating connectivity the z/OS subsystem to the zCEE server. It is configured using basic authentication or for CICS, TLS client authentication.

clientID – the identity under which the z/OS application is executing.

- For CICS, the task owner
- For IMS, the transaction owner
- For batch, the job owner

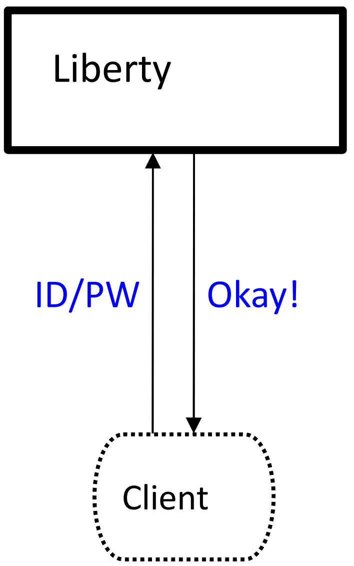
requireAuth	idAssertion	Actions performed by z/OS Connect
true	OFF	Identity assertion is disabled. The zCEE server authenticates *zCEEID* and checks whether *zCEEID* has the authority to invoke an API requester.
	ASSERT_SURROGATE	Identity assertion is enabled. The zCEE server authenticates *zCEEID* and checks whether *zCEEID* is a surrogate of *clientID*. If *zCEEID* is a surrogate of *clientID*, the server further checks whether *clientID* has the authority to invoke an API requester; otherwise, a BAQR7114E message occurs.
	ASSERT_ONLY	Identity assertion is enabled. The zCEE server authenticates *zCEEID* and directly checks whether *clientID* has the authority to invoke an API requester
false	OFF	Identity assertion is disabled. A BAQR0407W message occurs.
	ASSERT_SURROGATE	Identity assertion is enabled. The zCEE server checks whether *clientID* has the authority to invoke an API requester, and a warning message occurs to indicate that the ASSERT_ONLY value is used instead of the ASSERT_SURROGATE value.
	ASSERT_ONLY	Identity assertion is enabled. The zCEE server checks whether *clientID* has the authority to invoke an API requester

```
<zosconnect_apiRequesters idAssertion="ASSERT_ONLY">
</zosconnect_apiRequesters>
```

Authentication - TLS Mutual Authentication

Several different ways this can be accomplished:

Basic Authentication



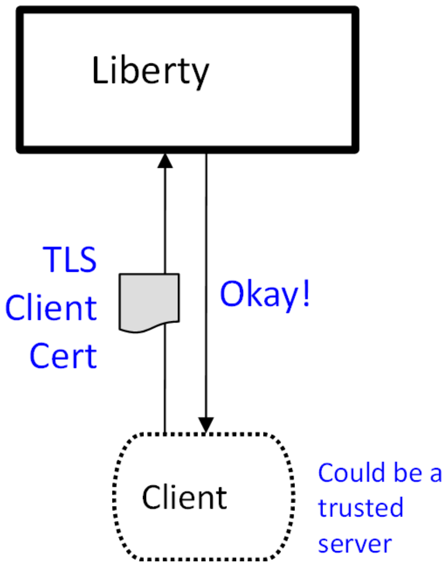
Server prompts for ID/PW

Client supplies ID/PW or ID/PassTicket

Server checks registry:

- **Basic (server.xml)**
- **SAF**

Client Certificate



Server prompts for cert.

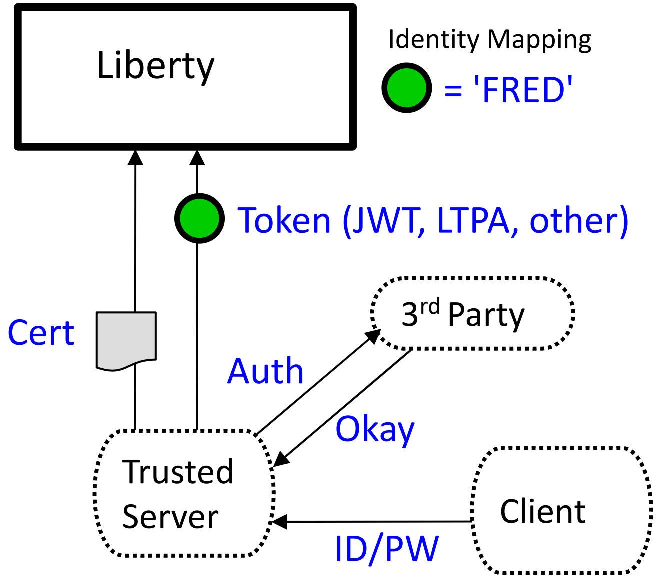
Client supplies certificate

Server validates cert and maps to an identity

Registry options:

- **SAF**

Third Party Authentication



Client authenticates to 3rd party sever

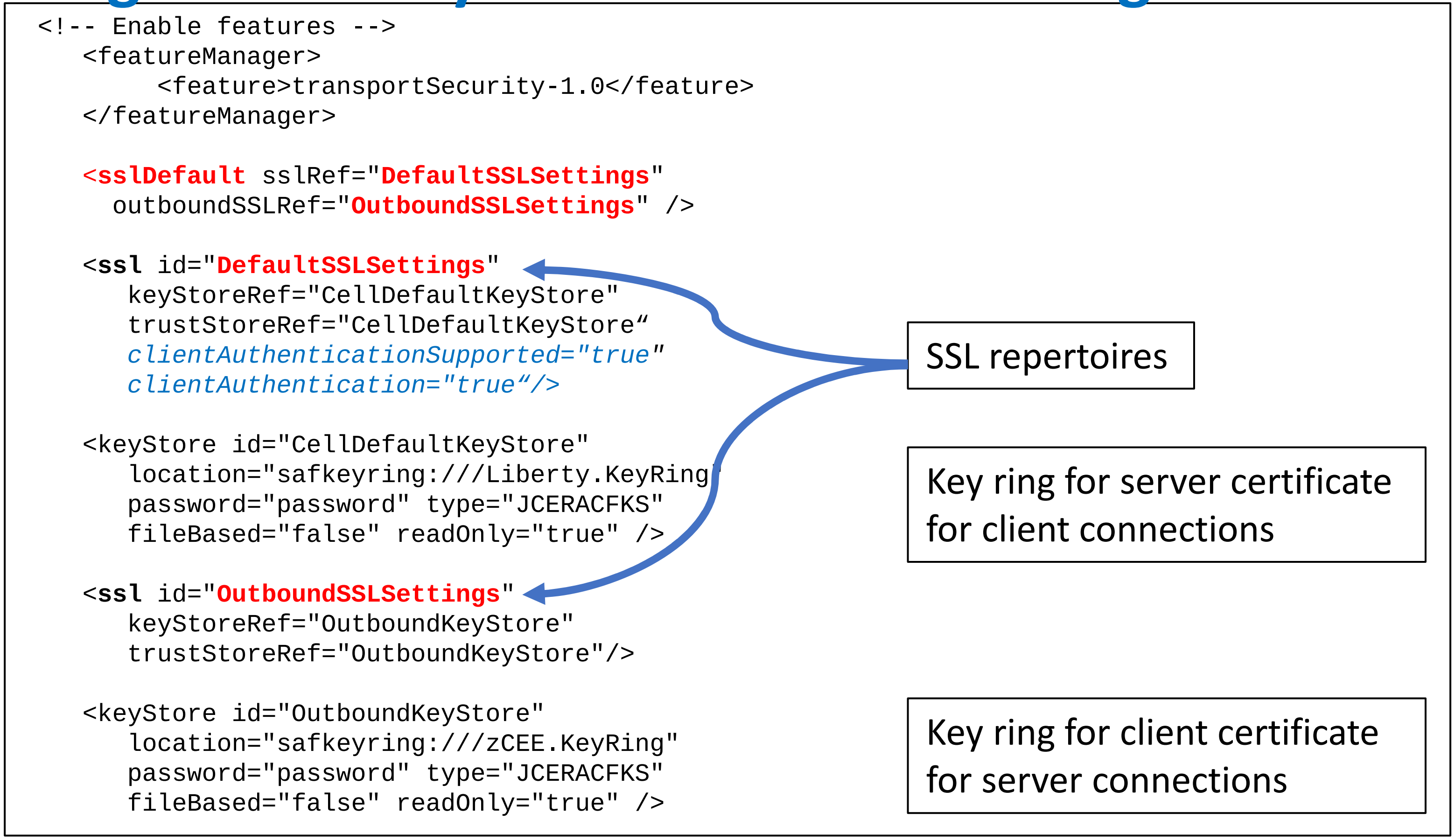
Client receives a trusted 3rd party token

Token flows to Liberty z/OS and is mapped to an identity

Registry options:

- **We may know these detail.**

Using this Liberty JSSE server XML configuration



```
<!-- Enable features -->
   <featureManager>
        <feature>transportSecurity-1.0</feature>
   </featureManager>

   <sslDefault sslRef="DefaultSSLSettings"
     outboundSSLRef="OutboundSSLSettings" />

   <ssl id="DefaultSSLSettings"
      keyStoreRef="CellDefaultKeyStore"
      trustStoreRef="CellDefaultKeyStore“
      clientAuthenticationSupported="true"
      clientAuthentication="true“/>

   <keyStore id="CellDefaultKeyStore"
      location="safkeyring:///Liberty.KeyRing"
      password="password" type="JCERACFKS"
      fileBased="false" readOnly="true" />

   <ssl id="OutboundSSLSettings"
      keyStoreRef="OutboundKeyStore"
      trustStoreRef="OutboundKeyStore"/>

   <keyStore id="OutboundKeyStore"
      location="safkeyring:///zCEE.KeyRing"
      password="password" type="JCERACFKS"
      fileBased="false" readOnly="true" />
```

SSL repertoires

Key ring for server certificate for client connections

Key ring for client certificate for server connections

Tech/Tip: Regarding *clientAuthentication* and *clientAuthenticationSupported*. Understand the implications of the interactions between these attributes. There may instances where you want to use HTTPS, but not always with mutual authentication Consider setting *clientAuthentication* to false when setting *clientAuthenticationSupported* to true.

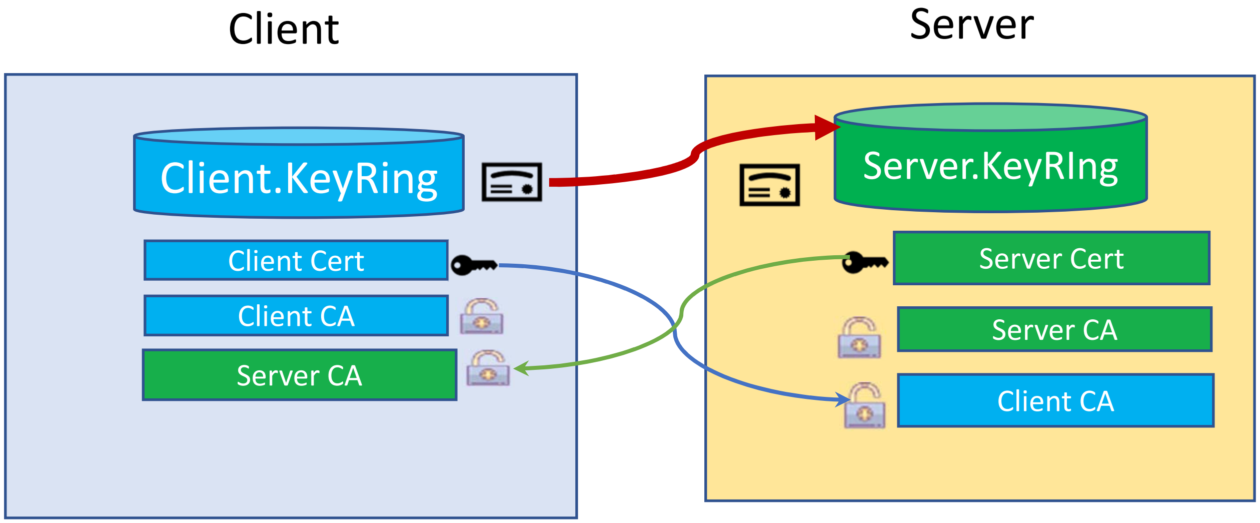
F BAQSTRT,REFRESH,KEYSTORE

Lets explore the basic TLS Handshake Flow

TLS handshake –

Server Authentication

Mutual Authentication (optional)



safkeyring:///**KeyRing** v safkeyring:**//owner/KeyRing**

RACF FACILITY resources

- IRR.DIGTCERT.LISTRING
  - READ to list your own key ring
  - UPDATE to list another user’s key ring
- IRR.DIGTCERT.LIST
  - READ to list your own certificate
  - UPDATE to list another user’s certificate
  - CONTROL to list SITE of CERTAUTH certificates

🔑 Certificate with a private key*

🔒 Certificate Authority (CA) certificate chain#

*For server and/or mutual authentication to work, the endpoint sending its server or client certificate must use a personal certificate with a private key. The private key is required to decrypt (or encrypt) a message digest that is sent from the other endpoint during the handshake flow. Generation of a message digest also requires access to the CA certificate used to sign the certificate.

#Refers to the set or of certificates used to issue the server or client personal certificate including any intermediate certificates all the way to the root CA.

Tech/Tip: cURL trace showing mutual authentication

```
* successfully set certificate verify locations:
* TLSv1.3 (OUT), TLS handshake, Client hello (01):
* TLSv1.3 (IN), TLS handshake, Server hello (02):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Request CERT (13):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS handshake, CERT verify (15):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (01):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384
* Server certificate:
*  subject: O=IBM; OU=LIBERTY; CN=wg31.washington.ibm.com
*  start date: Jan  4 04:00:00 2021 GMT
*  expire date: Jan  1 03:59:59 2023 GMT
*  common name: wg31.washington.ibm.com (matched)
*  issuer: OU=LIBERTY; CN=CA for Liberty
*  SSL certificate verify ok.
```

TLS 1.2 https://tools.ietf.org/html/rfc5246
TLS 1.3 https://tools.ietf.org/html/rfc8446

Tech/Tip: RACF digital certificate (RACDCERT) command review

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('Liberty CA’) +
  OU('LIBERTY')) WITHLABEL('Liberty CA') TRUST SIZE(2048) NOTAFTER(DATE(2022/12/31))
RACDCERT ID(LIBSERV) GENCERT SUBJECTSDN(CN('wg31.washington.ibm.com’) +
  O('IBM') OU('LIBERTY')) WITHLABEL('Liberty Client Cert') +
  ALTNAME(DOMAIN('wg31z.washington.ibm.com’)) SIGNWITH(CERTAUTH LABEL('Liberty CA')) SIZE(2048) +
  NOTAFTER(DATE(2022/12/31))
```

```
RACDCERT ID(LIBSERV) GENCERT SUBJECTSDN(CN('wg31.washington.ibm.com’) +
 O('IBM') OU('LIBERTY')) WITHLABEL('Liberty Server Cert')
RACDCERT ID(LIBSERV) GENREQ(LABEL('Liberty Server Cert')) DSN(CERT.REQ)

 Send the certificate to your Certificate Authority to be signed

racdcert CERTAUTH withlabel('Liberty CA')  add('USER1.LIBCA.PEM') TRUST
racdcert id(LIBSERV) withlabel('Liberty Server Cert') add('LIBSERV.P12) password('secret') TRUST
```

```
 /* Create Liberty key ring and connect CA and personal certificates */
racdcert id(libserv) addring(Liberty.KeyRing)
racdcert id(libserv) connect(ring(Liberty.KeyRing) label('CICS CA') certauth usage(certauth))
racdcert id(libserv) connect(ring(Liberty.KeyRing) label('Liberty CA') certauth usage(certauth)
 /* Connect default personal certificate */
racdcert id(libserv) connect(ring(Liberty.KeyRing) label('Liberty Client Cert') default
```

```
setropts raclist(digtcert) refresh
```

<u>Broadcom Support web pages</u>

Site of *What ACF2 security setup is needed for IBM’s z/OS Connect Enterprise Edition V3.0?*
https://knowledge.broadcom.com/external/article/128597/what-acf2-security-setup-is-needed-for-i.html
Site of *ACF2 setup for z/OS Connect Enterprise Edition V3.0*
https://knowledge.broadcom.com/external/article/142172/acf2-setup-for-zos-connect-enterprise-ed.html
Site of *Setting up Liberty Server for z/OS with Top Secret*
https://knowledge.broadcom.com/external/article/37272/setting-up-liberty-server-for-zos-with-t.html

Tech/Tip: RACF Certificate Filtering and Mapping

Filters for mapping certificates can be created with a RACDCERT command.

- Enter command RACDCERT ID MAP to create a filter that assigns RACF identity ATSUSER to any digital certificate signed with the ATS client signer certificate and where the subject is organizational unit ATS in organization IBM.

```
racdcert id(atsuser) map sdnfilter('OU=ATS.O=IBM’)
idnfilter('CN=ATS Client CA.OU=ATS.O=IBM’) withlabel('ATS USERS')
```

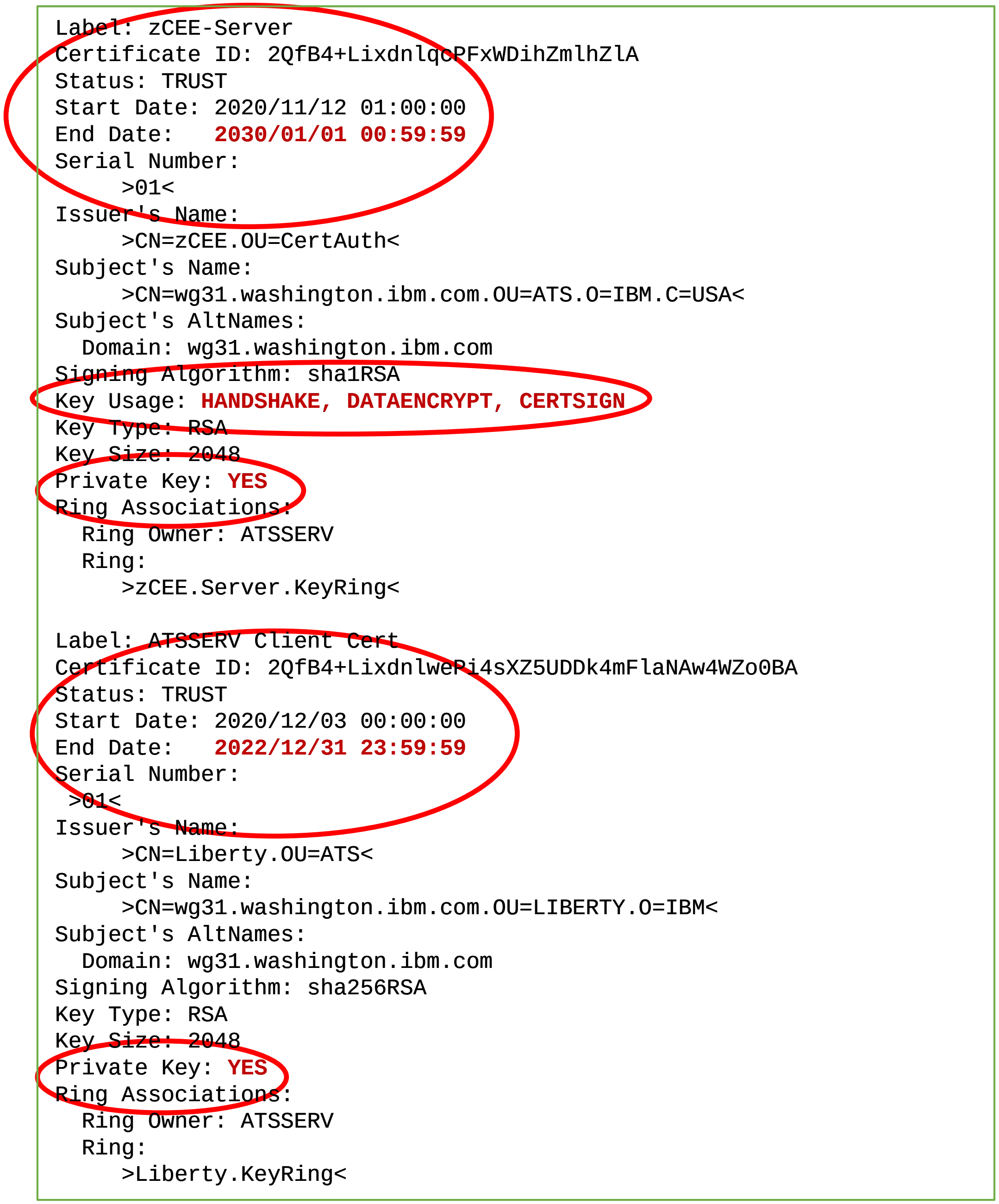
- Enter command RACDCERT ID MAP to create a filter that assigns RACF identity OTHUSER to any digital certificate signed by the ATS client signer certificate and where the subject is in organization IBM.

```
racdcert id(othuser) map sdnfilter('O=IBM’)
idnfilter('O=IBM') withlabel('IBM USERS')
```

- Refresh the in-storage profiles for digital certificate maps.

```
SETRPTS RACLIST(DIGTNMAP) REFRESH
```

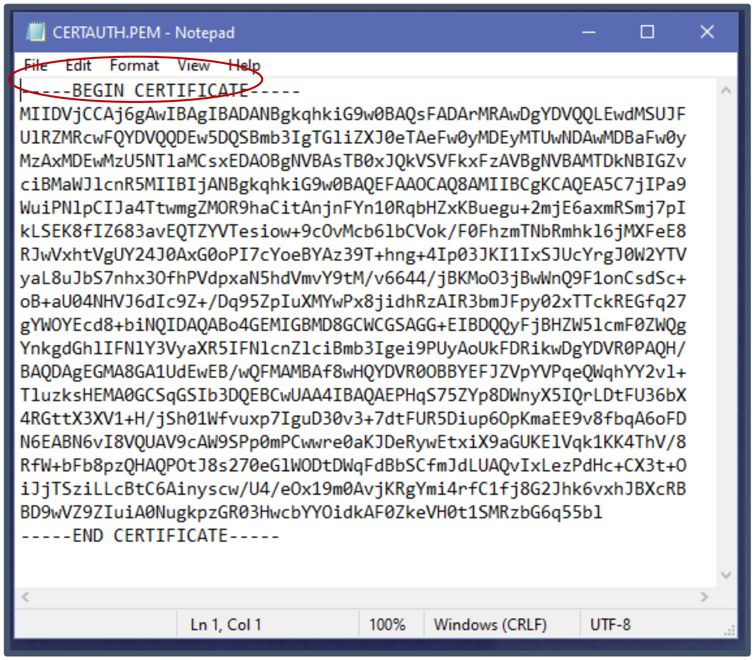
Tech/Tip: RACF Digital Certificates



```
Label: zCEE-Server
Certificate ID: 2QfB4+LixdnlqcPFxWDihZmlhZlA
Status: TRUST
Start Date: 2020/11/12 01:00:00
End Date:   2030/01/01 00:59:59
Serial Number:
     >01<
Issuer's Name:
     >CN=zCEE.OU=CertAuth<
Subject's Name:
     >CN=wg31.washington.ibm.com.OU=ATS.O=IBM.C=USA<
Subject's AltNames:
  Domain: wg31.washington.ibm.com
Signing Algorithm: sha1RSA
Key Usage: HANDSHAKE, DATAENCRYPT, CERTSIGN
Key Type: RSA
Key Size: 2048
Private Key: YES
Ring Associations:
  Ring Owner: ATSSERV
  Ring:
     >zCEE.Server.KeyRing<

Label: ATSSERV Client Cert
Certificate ID: 2QfB4+LixdnlwePi4sXZ5UDDk4mFlaNAw4WZo0BA
Status: TRUST
Start Date: 2020/12/03 00:00:00
End Date:   2022/12/31 23:59:59
Serial Number:
 >01<
Issuer's Name:
     >CN=Liberty.OU=ATS<
Subject's Name:
     >CN=wg31.washington.ibm.com.OU=LIBERTY.O=IBM<
Subject's AltNames:
  Domain: wg31.washington.ibm.com
Signing Algorithm: sha256RSA
Key Type: RSA
Key Size: 2048
Private Key: YES
Ring Associations:
  Ring Owner: ATSSERV
  Ring:
     >Liberty.KeyRing<
```

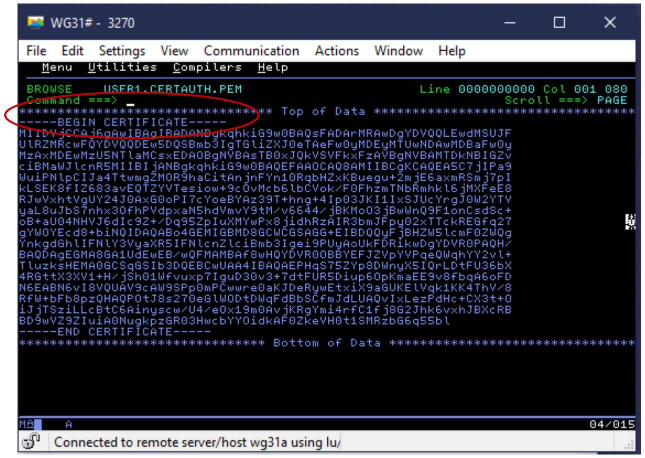
Tech/Tip: How to tell the difference?

Public certificate (site or certificate authority or certificate request)



```
-----BEGIN CERTIFICATE-----
MIIDVjCCAj6gAwIBAgIBADANBgkqhkiG9w0BAQsFADArMRAwDgYDVQQLEwdMSUJF
UlRZMRcwFQYDVQQDEw5DQSBmb3IgTGliZXJ0eTAeFw0yMDEyMTUwNDAwMDBaFw0y
MzAxMDEwMzU5NTlaMCsxEDAOBgNVBAsTB0xJQkVSVFkxFzAVBgNVBAMTDkNBIGZv
ciBMaWJlcnR5MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA5C7jIPa9
WuiPNlpCIJa4TtwmgZMOR9haCitAnjnFYn10RqbHZxKBuegu+2mjE6axmRSmj7pI
kLSEK8fIZ683avEQTZYVTesiow+9cOvMcb6lbCVok/F0FhzmTNbRmhkl6jMXFeE8
RJwVxhtVgUY24J0AxG0oPI7cYoeBYAz39T+hng+4Ip03JKI1IxSJUcYrgJ0W2YTV
yaL8uJbS7nhx3OfhPVdpxaN5hdVmvY9tM/v6644/jBKMoO3jBwWnQ9F1onCsdSc+
oB+aU04NHVJ6dIc9Z+/Dq95ZpIuXMYwPx8jidhRzAIR3bmJFpy02xTTckREGfq27
gYWOYEcd8+biNQIDAQABo4GEMIGBMD8GCWCGSAGG+EIBDQQyFjBHZW5lcmF0ZWQg
YnkgdGhlIFNlY3VyaXR5IFNlcnZlciBmb3Igei9PUyAoUkFDRikwDgYDVR0PAQH/
BAQDAgEGMA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0OBBYEFJZVpYVPqeQWqhYY2vl+
TluzksHEMA0GCSqGSIb3DQEBCwUAA4IBAQAEPHqS75ZYp8DWnyX5IQrLDtFU36bX
4RGttX3XV1+H/jSh01Wfvuxp7IguD30v3+7dtFUR5Diup6OpKmaEE9v8fbqA6oFD
N6EABN6vI8VQUAV9cAW9SPp0mPCwwre0aKJDeRywEtxiX9aGUKElVqk1KK4ThV/8
RfW+bFb8pzQHAQPOtJ8s270eGlWODtDWqFdBbSCfmJdLUAQvIxLezPdHc+CX3t+O
iJjTSziLLcBtC6Ainyscw/U4/eOx19m0AvjKRgYmi4rfC1fj8G2Jhk6vxhJBXcRB
BD9wVZ9ZIuiA0NugkpzGR03HwcbYYOidkAF0ZkeVH0t1SMRzbG6q55bl
-----END CERTIFICATE-----
```

Ln 1, Col 1
100%
Windows (CRLF)
UTF-8



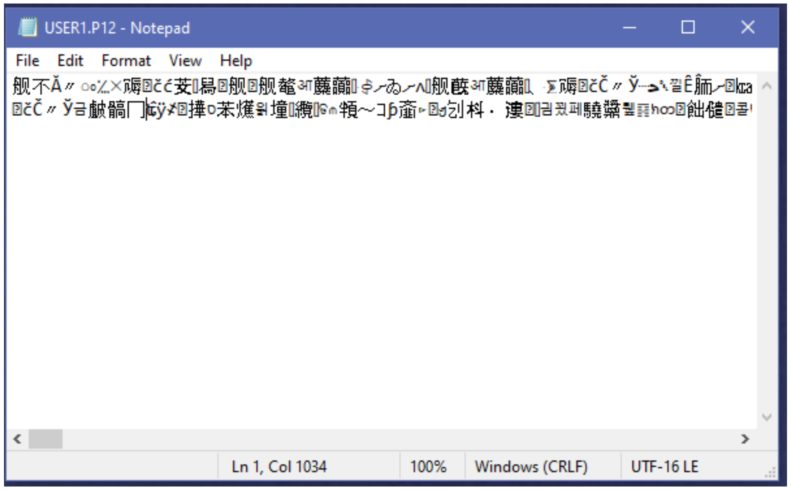
WG31# - 3270
File Edit Settings View Communication Actions Window Help
Menu Utilities Compilers Help

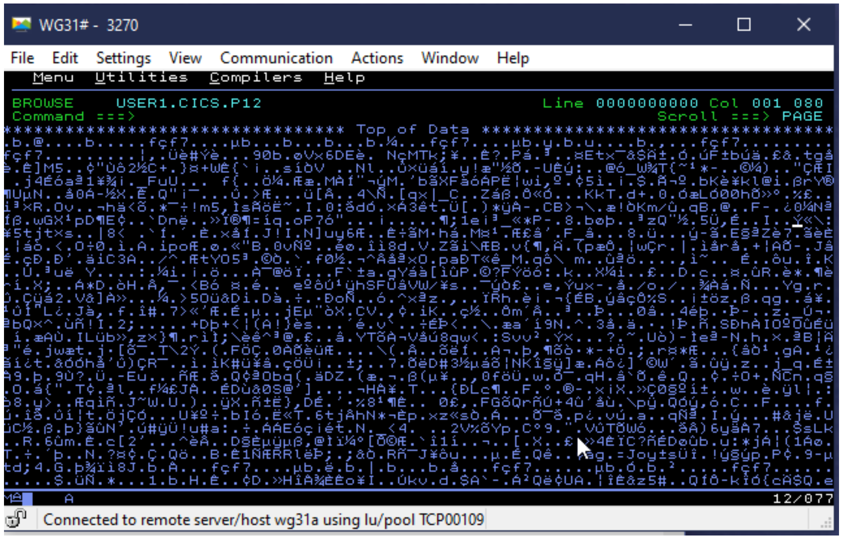
```
BROWSE    USER1.CERTAUTH.PEM                     Line 0000000000 Col 001 080
Command ===> _                                                Scroll ===> PAGE
******************************** Top of Data ********************************
-----BEGIN CERTIFICATE-----
```

MIIDVjCCAj6gAwIBAgIBADANBgkqhkiG9w0BAQsFADArMRAwDgYDVQQLEwdMSUJF
UlRZMRcwFQYDVQQDEw5DQSBmb3IgTGliZXJ0eTAeFw0yMDEyMTUwNDAwMDBaFw0y
MzAxMDEwMzU5NTlaMCsxEDAOBgNVBAsTB0xJQkVSVFkxFzAVBgNVBAMTDkNBIGZv
ciBMaWJlcnR5MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA5C7jIPa9
WuiPNlpCIJa4TtwmgZMOR9haCitAnjnFYn10RqbHZxKBuegu+2mjE6axmRSmj7pI
kLSEK8fIZ683avEQTZYVTesiow+9cOvMcb6lbCVok/F0FhzmTNbRmhkl6jMXFeE8
RJwVxhtVgUY24J0AxG0oPI7cYoeBYAz39T+hng+4Ip03JKI1IxSJUcYrgJ0W2YTV
yaL8uJbS7nhx3OfhPVdpxaN5hdVmvY9tM/v6644/jBKMoO3jBwWnQ9F1onCsdSc+
oB+aU04NHVJ6dIc9Z+/Dq95ZpIuXMYwPx8jidhRzAIR3bmJFpy02xTTckREGfq27
gYWOYEcd8+biNQIDAQABo4GEMIGBMD8GCWCGSAGG+EIBDQQyFjBHZW5lcmF0ZWQg
YnkgdGhlIFNlY3VyaXR5IFNlcnZlciBmb3Igei9PUyAoUkFDRikwDgYDVR0PAQH/
BAQDAgEGMA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0OBBYEFJZVpYVPqeQWqhYY2vl+
TluzksHEMA0GCSqGSIb3DQEBCwUAA4IBAQAEPHqS75ZYp8DWnyX5IQrLDtFU36bX
4RGttX3XV1+H/jSh01Wfvuxp7IguD30v3+7dtFUR5Diup6OpKmaEE9v8fbqA6oFD
N6EABN6vI8VQUAV9cAW9SPp0mPCwwre0aKJDeRywEtxiX9aGUKElVqk1KK4ThV/8
RfW+bFb8pzQHAQPOtJ8s270eGlWODtDWqFdBbSCfmJdLUAQvIxLezPdHc+CX3t+O
iJjTSziLLcBtC6Ainyscw/U4/eOx19m0AvjKRgYmi4rfC1fj8G2Jhk6vxhJBXcRB
BD9wVZ9ZIuiA0NugkpzGR03HwcbYYOidkAF0ZkeVH0t1SMRzbG6q55bl
-----END CERTIFICATE-----

```
******************************* Bottom of Data ******************************
```

Personal certificate with private key





WG31# - 3270
File Edit Settings View Communication Actions Window Help
Menu Utilities Compilers Help

```
BROWSE    USER1.CICS.P12                         Line 0000000000 Col 001 080
Command ===>                                                  Scroll ===> PAGE
******************************** Top of Data ********************************
```

12/077
Connected to remote server/host wg31a using lu/pool TCP00109

Tech/Tip: Combining TLS and basic authentication

```
//****************************************************************
//*  SET SYMBOLS
//****************************************************************
//EXPORT EXPORT SYMLIST=(*)
// SET CURL= '/usr/lpp/rocket/curl'
//****************************************************************
//*  CURL Procedure
//****************************************************************
//CURL PROC
//CURL EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSERR   DD SYSOUT=*
//STDOUT   DD SYSOUT=*
// PEND
//****************************************************************
//*  STEP CURL - use cURL to deploy API cscvinc
//****************************************************************
//DEPLOY EXEC CURL
BPXBATCH SH export CURL=&CURL; +
$CURL/bin/curl -X PUT -s +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
https://wg31.washington.ibm.com:9445/zosConnect/apis/cscvinc?status=sto+
pped > null; +
$CURL/bin/curl -X DELETE -s +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
https://wg31.washington.ibm.com:9445/zosConnect/apis/cscvinc > null; +
$CURL/bin/curl -X POST -s +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
--data-binary @/u/johnson/cscvinc.aar +
--header "Content-Type: application/zip" +
https://wg31.washington.ibm.com:9445/zosConnect/apis
//****************************************************************
//*  STEP CURL - use cURL to invoke the API cscvinc
//****************************************************************
//INVOKE EXEC CURL
//SYSTSIN  DD *,SYMBOLS=EXECSYS
BPXBATCH SH export CURL=&CURL; $CURL/bin/curl -X GET -s +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
https://wg31.washington.ibm.com:9444/cscvinc/employee/000100
```

```
<httpEndpoint id="defaultHttpEndpoint"
    host="*"
    httpPort="9080"
    httpsPort="9443" />

<sslDefault sslRef="DefaultSSLSettings"
   outboundSSLRef="DefaultSSLSettings" />

<ssl id="DefaultSSLSettings"
    keyStoreRef="CellDefaultKeyStore"
    trustStoreRef="CellDefaultKeyStore"
    clientAuthenticationSupported="true"
    clientAuthentication=“true"/>

<keyStore id="CellDefaultKeyStore"
    location="safkeyring:///Liberty.KeyRing"
    password="password" type="JCERACFKS"
    fileBased="false" readOnly="true" />
```

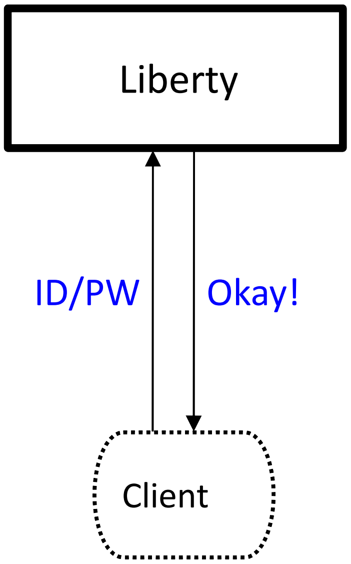
```
<httpEndpoint id="AdminHttpEndpoint"
    host="*"
    httpPort="-1"
    httpsPort="9445"
    sslOptionsRef="mySSLOptions"/>

<sslOptions id="mySSLOptions“
    sslRef="BatchSSLSettings“/>

<ssl id="BatchSSLSettings"
   keyStoreRef="CellDefaultKeyStore"
   trustStoreRef="CellDefaultKeyStore"
   clientAuthenticationSupported="true"
   clientAuthentication="false"/>
```

https://www.rocketsoftware.com/platforms/ibm-z/curl-for-zos

Authentication - Third Party Authentication

Several different ways this can be accomplished:

Basic Authentication



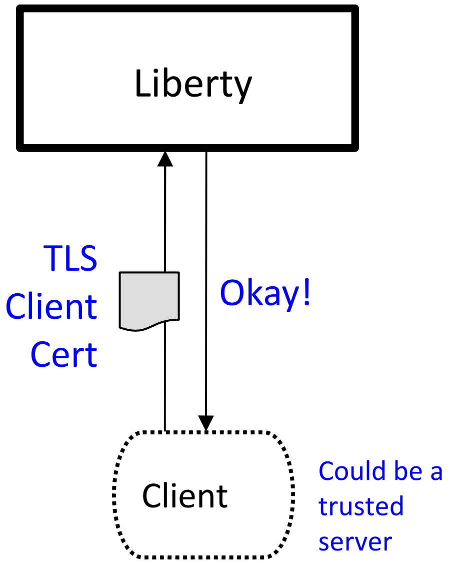
Server prompts for ID/PW

Client supplies ID/PW or ID/PassTicket

Server checks registry:

- **Basic (server.xml)**
- **SAF**

Client Certificate



Server prompts for cert.

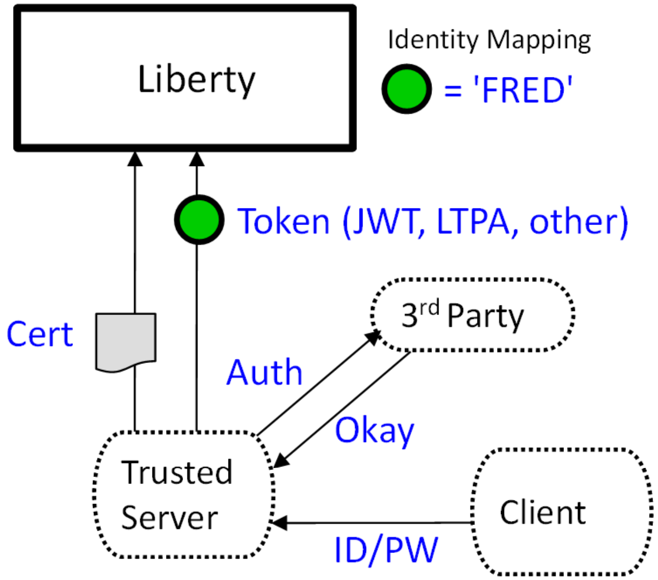
Client supplies certificate

Server validates cert and maps to an identity

Registry options:

- **SAF**

Third Party Authentication



Client authenticates to 3rd party sever

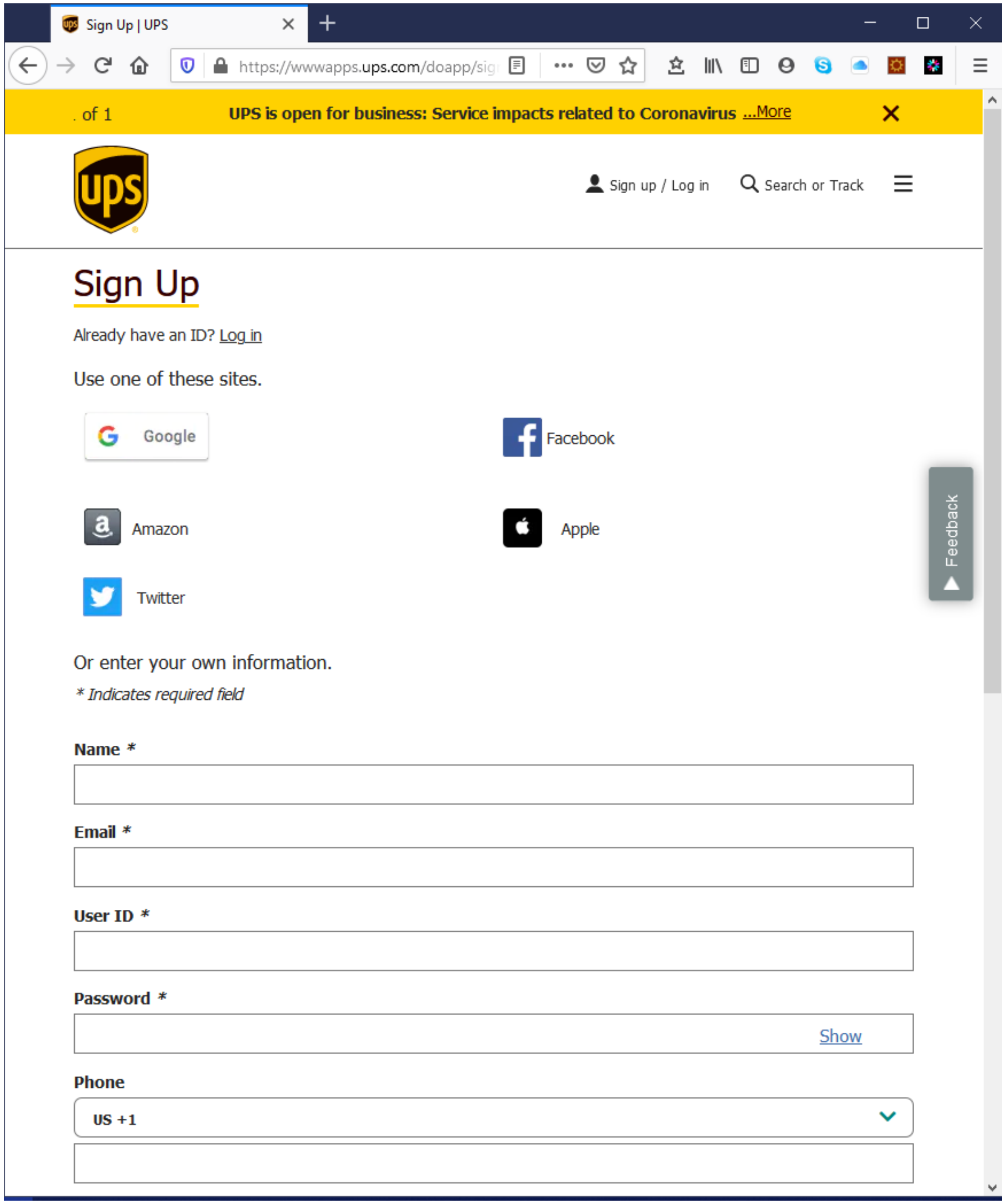
Client receives a trusted 3rd party token

Token flows to Liberty z/OS and is mapped to an identity

Registry options:

- **We may know these detail.**

UPS Authentication



UPS is open for business: Service impacts related to Coronavirus ...More

Sign Up

Already have an ID? Log in

Use one of these sites.

Google

Facebook

Amazon

Apple

Twitter

Or enter your own information.

** Indicates required field*

Name *

Email *

User ID *

Password * Show

Phone

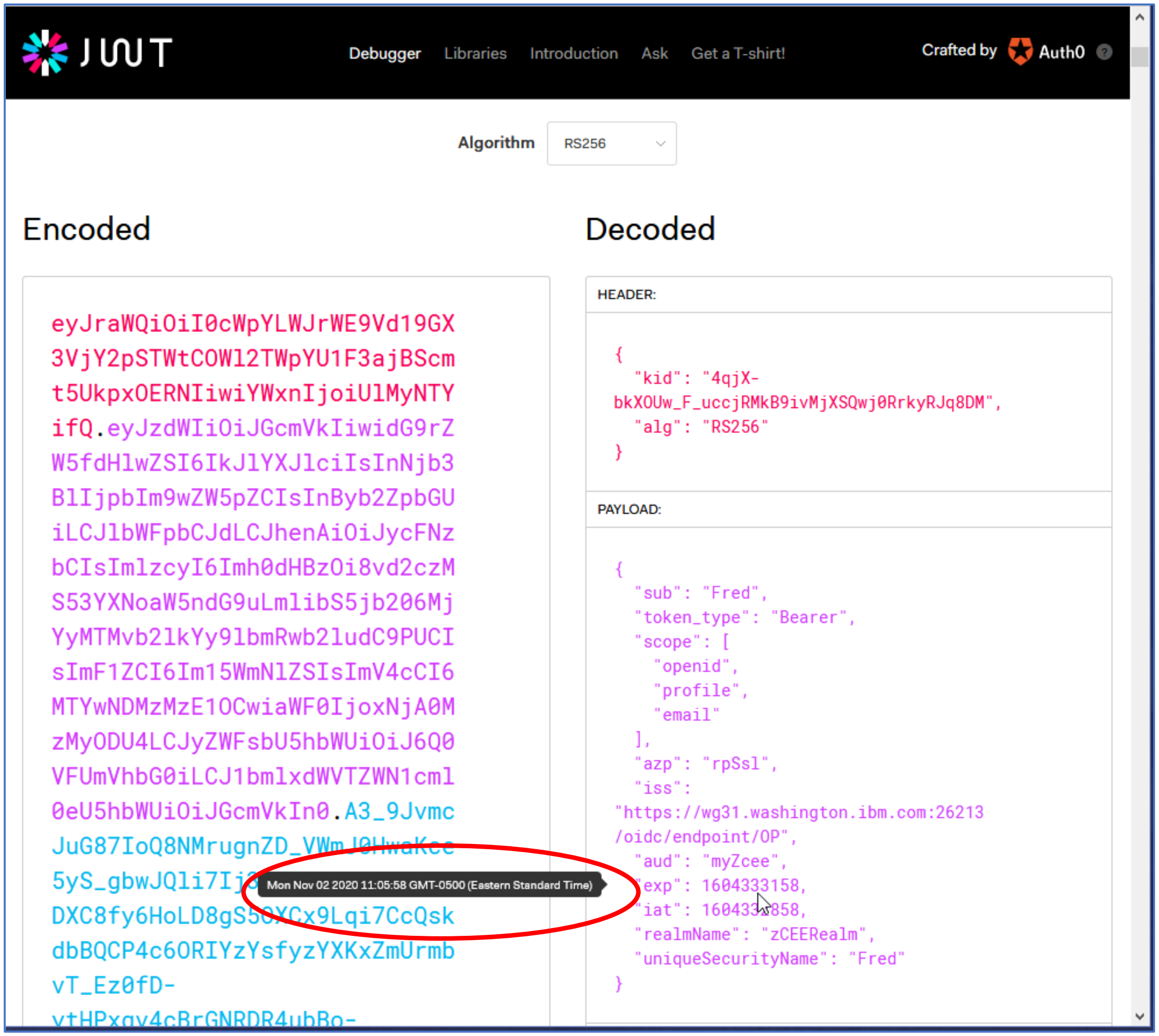
US +1

Security token types by Liberty

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 social 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	

What is a 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



Algorithm RS256

Encoded

```
eyJraWQiOiI0cWpYLWJrWE9Vd19GX
3VjY2pSTWtCOW12TWpYU1F3ajBScm
t5UkpxOERNIiwiYWxnIjoiUlMyNTY
ifQ.eyJzdWIiOiJGcmVkIiwidG9rZ
W5fdHlwZSI6IkJlYXJlciIsInNjb3
BlIjpbIm9wZW5pZCIsInByb2ZpbGU
iLCJlbWFpbCJdLCJhenAiOiJycFNz
bCIsImlzcyI6Imh0dHBzOi8vd2czM
S53YXNoaW5ndG9uLmlibS5jb206Mj
YyMTMvb2lkYy9lbmRwb2ludC9PUCI
sImF1ZCI6Im15WmN1ZSIsImV4cCI6
MTYwNDMzMzE1OCwiaWF0IjoxNjA0M
zMyODU4LCJyZWFsbU5hbWUiOiJ6Q0
VFUmVhbG0iLCJ1bmlxdWVTZWN1cml
0eU5hbWUiOiJGcmVkIn0.A3_9Jvmc
JuG87IoQ8NMrugnZD_VWmJ0HwaKce
5yS_gbwJQli7Ij
DXC8fy6HoLD8gS50XCx9Lqi7CcQsk
dbBQCP4c60RIYzYsfyzYXKxZmUrmb
vT_Ez0fD-
vtHPxqv4cBrGNRDR4ubBo-
```

Mon Nov 02 2020 11:05:58 GMT-0500 (Eastern Standard Time)

Decoded

HEADER:

```
{
  "kid": "4qjX-
bkXOUw_F_uccjRMkB9ivMjXSQwj0RrkyRJq8DM",
  "alg": "RS256"
}
```

PAYLOAD:

```
{
  "sub": "Fred",
  "token_type": "Bearer",
  "scope": [
    "openid",
    "profile",
    "email"
  ],
  "azp": "rpSsl",
  "iss":
"https://wg31.washington.ibm.com:26213
/oidc/endpoint/OP",
  "aud": "myZcee",
  "exp": 1604333158,
  "iat": 1604332858,
  "realmName": "zCEERealm",
  "uniqueSecurityName": "Fred"
}
```

Values derived from the OAUTH configuration:

- signatureAlgorithm="**RS256**“
- accessTokenLifetime="**300**"
- resourceIds="**myZcee**"

https://jwt.io

Open security standards

- **OAuth** is an open standard for access delegation, used as a way to grant websites or applications access to their information without requiring a password.
- **OpenID Connect** is an authentication layer on top of Oauth. It allows the verification of the identity of an end-user based on authentication performed by an authorization server.
- **JWT** (JSON Web token) defines a compact and self-contained way for securely transmitting information between parties as a JSON object

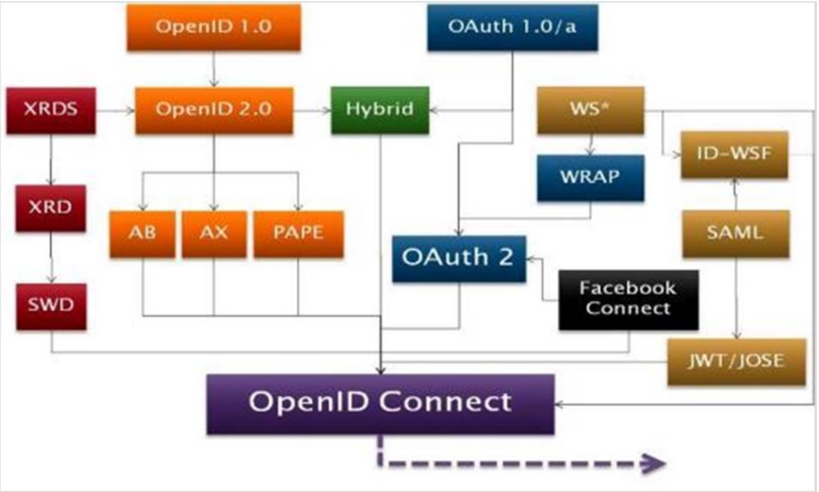
See the YouTube videos:

OAuth 2.0 and OpenID Connect (in plain English)

https://www.youtube.com/watch?v=996OiexHze0

OpenID Connect on Liberty

https://www.youtube.com/watch?v=fuajCS5bG4c



OpenID Connect/OAuth and z/OS Connect

- ***From the z/OS Connect Knowledge Center:*** *z/OS Connect EE security can operate with traditional z/OS security, for example, System Authorization Facility (SAF) and also with open standards such as Transport Layer Security (TLS), JSON Web Token (JWT), and* ***OpenID Connect.***

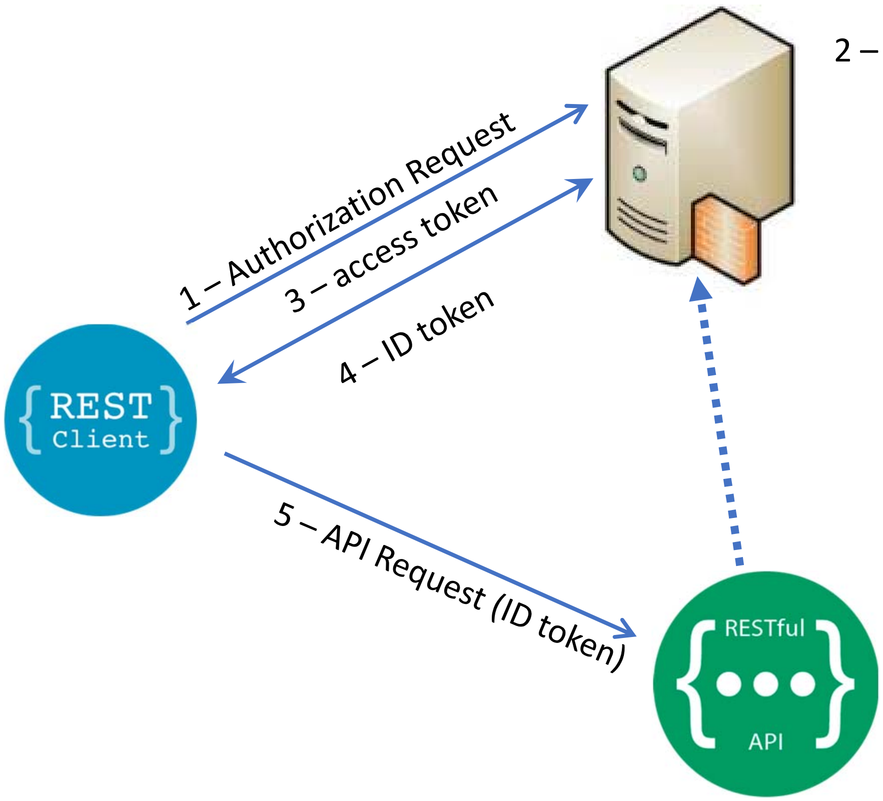
- ***From the OpenID Core specification:*** *OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol. It enables Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner.*

- ***OAuth 2.0 Core (RFC 6749) Specifications:*** https://tools.ietf.org/html/rfc6749

- ***OpenID Connect Core Specifications***: https://openid.net/specs/openid-connect-core-1_0.html

- ***Again for a very good explanation of this topic see YouTube video*** *OAuth 2.0 and OpenID Connect (in plain English)* https://www.youtube.com/watch?v=996OiexHze0

Typical OAuth Authorization Flow



2 – Authenticate User

Authorization Server

1 – Authorization Request

3 – access token

4 – ID token

OAuth 2.0
Authorization
Server

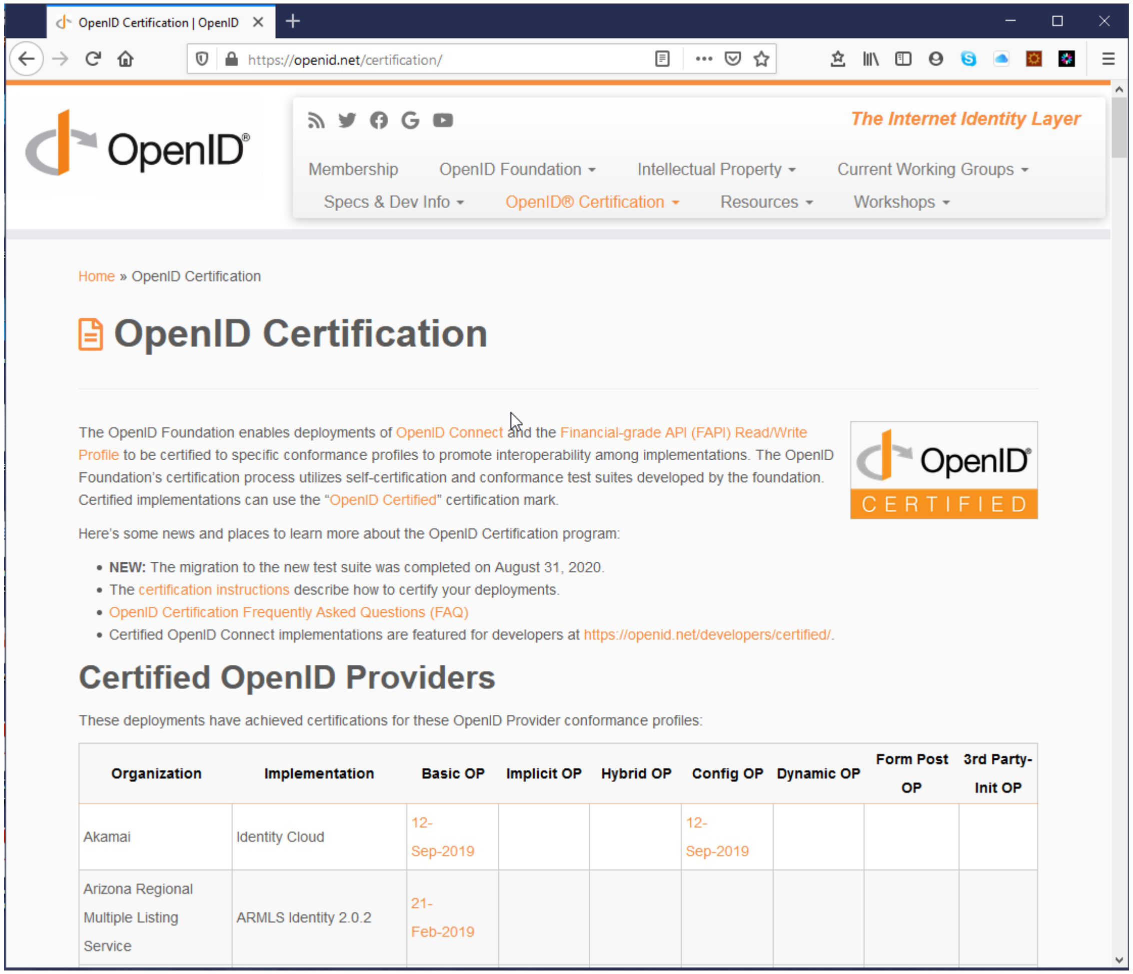
Client or
Relying Party (RP)

REST Client

5 – API Request (ID token)

RESTful API

There are a multitude of OpenID Certified Providers



Home » OpenID Certification

OpenID Certification

The OpenID Foundation enables deployments of OpenID Connect and the Financial-grade API (FAPI) Read/Write Profile to be certified to specific conformance profiles to promote interoperability among implementations. The OpenID Foundation's certification process utilizes self-certification and conformance test suites developed by the foundation. Certified implementations can use the "OpenID Certified" certification mark.

Here's some news and places to learn more about the OpenID Certification program:

- **NEW:** The migration to the new test suite was completed on August 31, 2020.
- The certification instructions describe how to certify your deployments.
- OpenID Certification Frequently Asked Questions (FAQ)
- Certified OpenID Connect implementations are featured for developers at https://openid.net/developers/certified/.

Certified OpenID Providers

These deployments have achieved certifications for these OpenID Provider conformance profiles:

Organization	Implementation	Basic OP	Implicit OP	Hybrid OP	Config OP	Dynamic OP	Form Post OP	3rd Party-Init OP
Akamai	Identity Cloud	12-Sep-2019			12-Sep-2019			
Arizona Regional Multiple Listing Service	ARMLS Identity 2.0.2	21-Feb-2019						

https://openid.net/certification/

Some basic OAuth/OpenID Connect terms

- ***Authorization server* -** The server that issues access tokens to the client after authenticating the resource owner and obtaining authorization. *In a z/OS Connect EE API requester scenario, the authorization server is called by the z/OS Connect EE server to retrieve an access token.*

- ***Authorization Endpoint*** - A service or endpoint on an OAuth authorization server that accepts an authorization request from a client to perform authentication and authorization of a user. The authorization endpoint returns an authorization grant, or code, to the client in the Authorization Code Flow. In the Implicit Flow, the authorization endpoint returns an access token to the client.

- **Token Endpoint** – A service or endpoint on an OP that accepts an authorization grant, or code, from a client in exchange for an access token, ID token, and refresh token

- ***Access Token*** – A credential that is used to access protected resources. An access token is a string that represents an authorization that is issued to the client. The access token is usually opaque to the client (it does not have to be opaque) and can be JSON Web Token (JTW). See URL https://tools.ietf.org/html/rfc6749 Section 1.4 for more information.

- ***OAuth token* -** With OAuth 2.0, access tokens are used to access protected resources. An access token is normally a string that represents an authorization that is issued to the client. The string is usually opaque to the client. Opaque tokens may require that the token recipient call back to the server that issued the token. *However, an access token can also be in the form of a JSON Web Token (JWT) which does not require a call back (introspection).*

- **Scope -** Privilege or permission that allows access to a set of resources of a third party.

Some basic OAuth/OpenID Connect terms

- **Relying Party (RP) –** An entity that relies on an OP to authenticate a user and obtain an authorization to access a user’s resource. *For z/OS Connect API Requester, it is the Liberty server configured as an OpenID Connect Client, e.g., using <openidConnectClient/> XML configuration elements.*

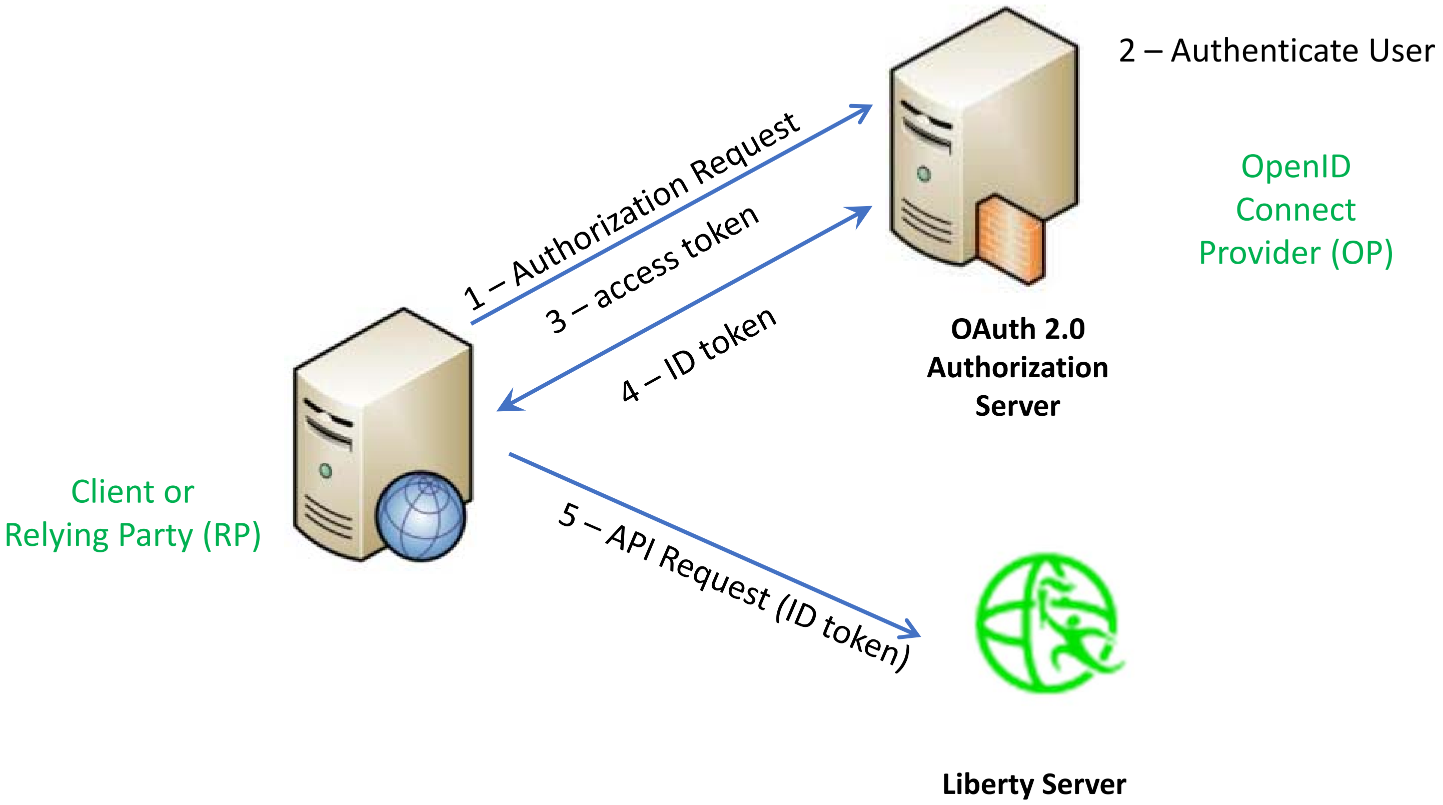
- **OpenID Connect Provider (OP)** - An OAuth 2.0 authorization server that is capable of providing claims to a client or Relying Party (RP) , *an OpenID component.*

- ***Resource owner*** - An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end user. *In a z/OS Connect EE API requester scenario, the resource owner might be the user of the CICS, IMS, or z/OS application.*

- ***Resource server*** - The server that hosts the protected resources and accepts and responds to protected resource requests by using access tokens. *In a z/OS Connect API provider, the resource server is the z/OS Connect server. In a z/OS Connect EE API requester scenario, the resource server is the request endpoint for the remote RESTful API*

- ***ID Token*** - is an OpenID Connect token that is an extension to OAuth 2.0 specification access tokens. This token is a JSON Web Token (JWT). See URL https://openid.net/specs/openid-connect-core-1_0.html#IDToken for more information about the extensions.

Typical OAuth Authorization Flow for a z/OS Connect API Provider


2 – Authenticate User
OpenID Connect Provider (OP)
1 – Authorization Request
3 – access token
4 – ID token
OAuth 2.0 Authorization Server
Client or Relying Party (RP)
5 – API Request (ID token)
Liberty Server

Tech/Tip: Let’s explore an OpendID Provider flow using Liberty

```
<httpEndpoint host="*" httpPort="26212" httpsPort="26213" id="defaultHttpEndpoint"/>

<openidConnectProvider id="OP“
   signatureAlgorithm="RS256“
   keyStoreRef="jwtStore"
  oauthProviderRef="OIDCssl" >
</openidConnectProvider>

<oauthProvider id="OIDCssl"
  httpsRequired="true"
  jwtAccessToken="true"
  autoAuthorize ="true"
  accessTokenLifetime="300">

<!-- Define OIDC Client for zCEE Authentication  -->
 <autoAuthorizeClient>zCEEClient</autoAuthorizeClient>
   <localStore>
     <client name="zCEEClient"
     secret="secret"
     displayname="zCEEClient"
     scope="openid"
     enabled="true"
     resourceIds="myZcee"/>
   </localStore>
```

Key Points:

- **keyStoreRef -** A keystore containing the private key necessary for signing with an asymmetric algorithm.
- **jwtAccessToken -** generate a Json Web Token, serialize it as a string and put in the place of the access token.

Tech/Tip: Generating a JWT using Liberty

```
  <autoAuthorizeClient>rpSsl</autoAuthorizeClient>
    <localStore>
      <client name="rpSsl"
      displayname="rpSsl"
      grantType="implicit"
      redirect="https://wg31.washington.ibm.com:26223/oidcclient/redirect/RPssl"
      scope="openid profile scope1 email phone address"
      enabled="true"
      resourceIds="myZcee"/>
     </localStore>
 </oauthProvider>

<!--Key store that contains certificate used to sign JWT-->
<keyStore fileBased="false" id="jwtStore"
  location="safkeyring:///JWT.KeyRing"
  password="password" readOnly="true" type="JCERACFKS"/>

<!-- Define a basic user registry -->
<basicRegistry id="basicRegistry"
  realm="zCEERealm">
  <user name="auser" password="pwd"/>
  <user name="Fred" password="fredpwd"/>
  <user name="distuser1" password="pwd"/>
  <user name="distuser2" password="pwd"/>
</basicRegistry>
```

```
RACMAP ID(FRED)  MAP USERDIDFILTER(NAME('Fred'))
     REGISTRY(NAME('*')) WITHLABEL('zCEE JWT FRED')
RACMAP ID(USER1)  MAP USERDIDFILTER(NAME('distuser1'))
     REGISTRY(NAME('*')) WITHLABEL('zCEE JWT distuser1')
RACMAP ID(USER2)  MAP USERDIDFILTER(NAME('distuser2'))
     REGISTRY(NAME('*')) WITHLABEL('zCEE JWT distuser2')
```

Tech/Tip: RACMAP Commands

```
RACMAP ID(USER1) MAP USERDIDFILTER(NAME('distuser1'))
     REGISTRY(NAME('*')) WITHLABEL('zCEE token user1')
RACMAP ID(USER1) MAP USERDIDFILTER(NAME('distributeUser1'))
     REGISTRY(NAME('zCEERealm')) WITHLABEL('zCEE user1')
RACMAP ID(USER1) MAP USERDIDFILTER(NAME('UID=user1,CN=User Name,OU=IBM ATG,O=IBM,C=US'))
     registry(name('*')) withlabel('USER X500 DN’)
RACMAP ID(ATSUSER) MAP USERDIDFILTER(NAME('OU=IBM ATS,O=IBM,C=US'))
     registry(name('*')) withlabel('ATS USER’)
RACMAP ID(IBMUSER) MAP USERDIDFILTER(NAME('O=IBM,C=US'))
     registry(name('*')) withlabel('IBM USER')
```

```
RACMAP ID(USER1) LISTMAP
Label: zCEE token user1
Distributed Identity User Name Filter:
  >distuser1<
Registry Name:
  >*<

Label: zCEE user1
Distributed Identity User Name Filter:
  >distributedUuser1<
Registry Name:
  >zCEERealm<

Label: USER X500 DN
Distributed Identity User Name Filter:
  >UID=user1,CN=User Name,OU=IBM ATG,O=IBM,C=US<
Registry Name:
  >*<
```

```
RACMAP ID(USER1) LISTMAP(LABEL('USER X500 DN’)

RACMAP ID(USER1) DELMAP (LABEL('zCEE distuser1’)

RACMAP QUERY USERDIDFILTER(NAME('USER1')) REGISTRY(NAME('*'))
```

Liberty/zCEE OpenID Client identity mapping configuration attributes

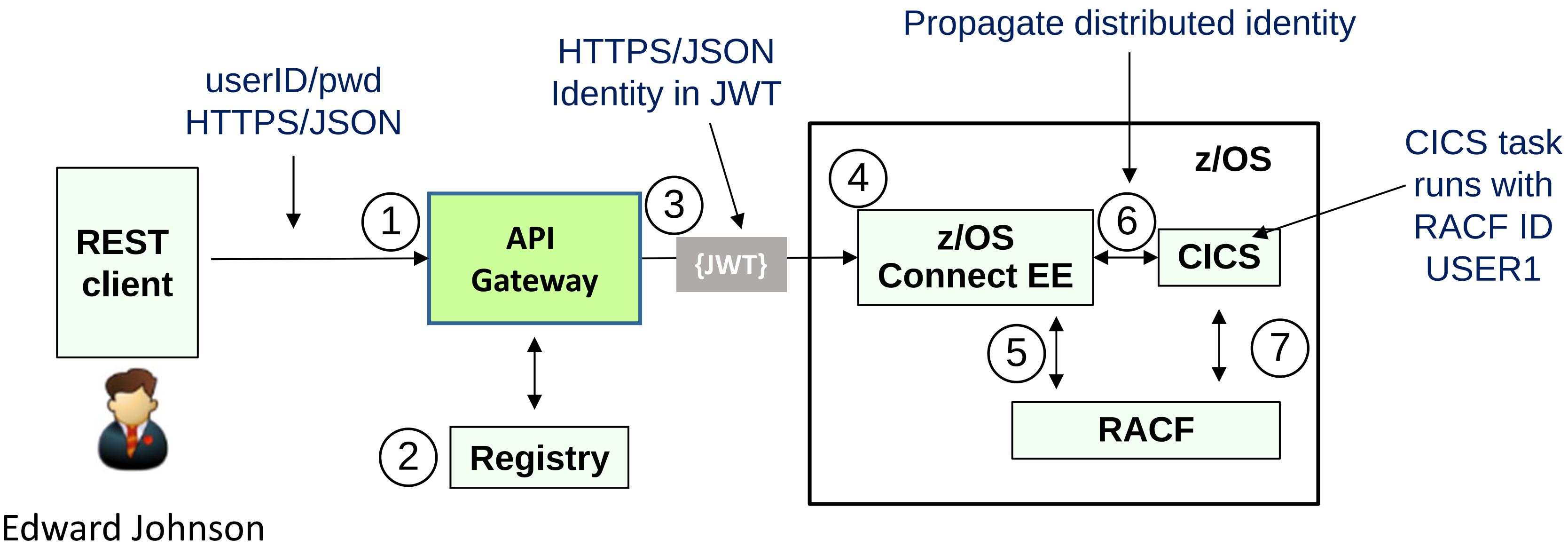
```
<safRegistry id="saf" />
<safAuthorization racRouteLog="ASIS" />
<safCredentials unauthenticatedUser="WSGUEST"
       mapDistributedIdentities="true"
       profilePrefix="BBGZDFLT" />
```

Use distributed identity filters to map the distributed identities to SAF user IDs, e.g., IDIDMAP, e.g., RACMAP.

```
<authFilter id="ATSAuthFilter">
   <requestUrl  id="ATSDemoUrl“
    name="ATSRefererUri"
    matchType="contains"
    urlPattern="/cscvinc/employee|/db2/employee|/mqapi/loan"/>
</authFilter>
<openidConnectClient id="ATS"
      httpsRequired="true"
      authFilterRef="ATSAuthFilter"
      inboundPropagation="required"
      scope="openid profile email"
      audiences="myZcee"
      issuerIdentifier=
https://wg31.washington.ibm.com:26213/oidc/endpoint/OP
      mapIdentityToRegistryUser=“false"
      signatureAlgorithm="RS256"
      userIdentityToCreateSubject="sub"
      trustAliasName="JWT-Signer-Certificate"
      trustStoreRef="jwtTrustStore"
      authnSessionDisabled="true"
      disableLtpaCookie="true">
  </openidConnectClient>
<keyStore fileBased="false" id="jwtTrustStore"
     location="safkeyring:///JWT.KeyRing"
     password="password" readOnly="true" type="JCERACFKS"/>
```

Specifies whether to map the identity to a registry user. If this is set to false, then the user registry is not used to create the user subject.

Example scenario – security flow



```
RACMAP ID(USER1) MAP USERDIDFILTER(NAME(‘Edward Johnson‘))
REGISTRY(NAME('*'))
```

1. User authenticates with the managed API using a "distributed" identity and a password
2. An external registry is used as the user registry for distributed users and groups
3. API Gateway 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 calls RACF to map distributed ID to RACF user ID and authorizes access to API
6. z/OS Connect EE CICS service provider propagates distributed ID to CICS
7. CICS calls RACF to map distributed ID to RACF user ID and performs resource authorization checks

JWT used in scenario

```
{
 "alg": "RS256"
}
{
 "sub": "Edward Johnson",
 "token_type": "Bearer",
 "azp": "rpSsl",
 "iss": "https://wg31.washington.ibm.com:26213/oidc/endpoint/OPssl",
 "aud": "myZcee",,
 "realmName": "zCEERealm",
 "uniqueSecurityName": "Edward Johnson"
}
RSASHA256(base64UrlEncode(header)+ base64UrlEncode(payload)
```

- 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 **distuser** identifies the principal that is the subject of the JWT
- The **aud** (audience) claim **myZcee** identifies the recipients for which the JWT is intended

Configuring authentication with JWT

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="RPssl" inboundPropagation="required"
    signatureAlgorithm="RS256" trustAliasName="JWT-Signer"
    trustStoreRef="jwtTrustStore"
     userIdentityToCreateSubject="sub" mapIdentityToRegistryUser=“true"
    issuerIdentifier=“https://wg31.washington.ibm.com:26213/oidc/endpoint/OPssl"
    authnSessionDisabled="true" audiences="myZcee"/>
```

- ***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

Using authorization filters with 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="RPssl" inboundPropagation="required"
     signatureAlgorithm="RS256" trustAliasName="JWT-Signer"
     trustStoreRef="jwtTrustStore"
      userIdentityToCreateSubject="sub" mapIdentityToRegistryUser=“true"
     issuerIdentifier=“https://wg31.washington.ibm.com:26213/oidc/endpoint/OPssl"
     authnSessionDisabled="true" audiences="myZcee"
    authFilterRef=“JwtAuthFilter”/>
<authFilter id=“API Gateway">
    <remoteAddress id=“ApiAddress" ip="10.7.1.*" matchType="equals"/>
</authFilter>
<authFilter id=“URLFilter">
     <requestUrl id=“URL"  urlPattern=“/cscvinc/employee/|/db2/employee|/mqapi/loan"/>“
          matchType="equals"/> </authFilter>
<authFilter id="JwtAuthFilter" >
  <requestHeader id="authHeader" name="Authorization" value="Bearer" matchType="contains"/>
</authFilter>
```

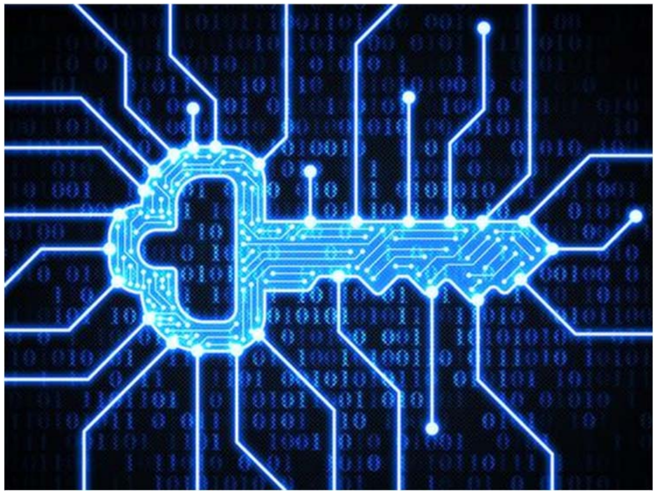
Some alternative filter types

- 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.

General security terms or considerations

- Identifying who or what is requesting access (**Authentication)**
  - Basic Authentication
  - Mutual Authentication using TLS
  - Third Party Tokens

- Ensuring that the message has not been altered in transit (**Data Integrity)** and ensuring the confidentiality of the message in transit (**Encryption)**
  - TLS (encrypting messages and generating/sending a digital signature)



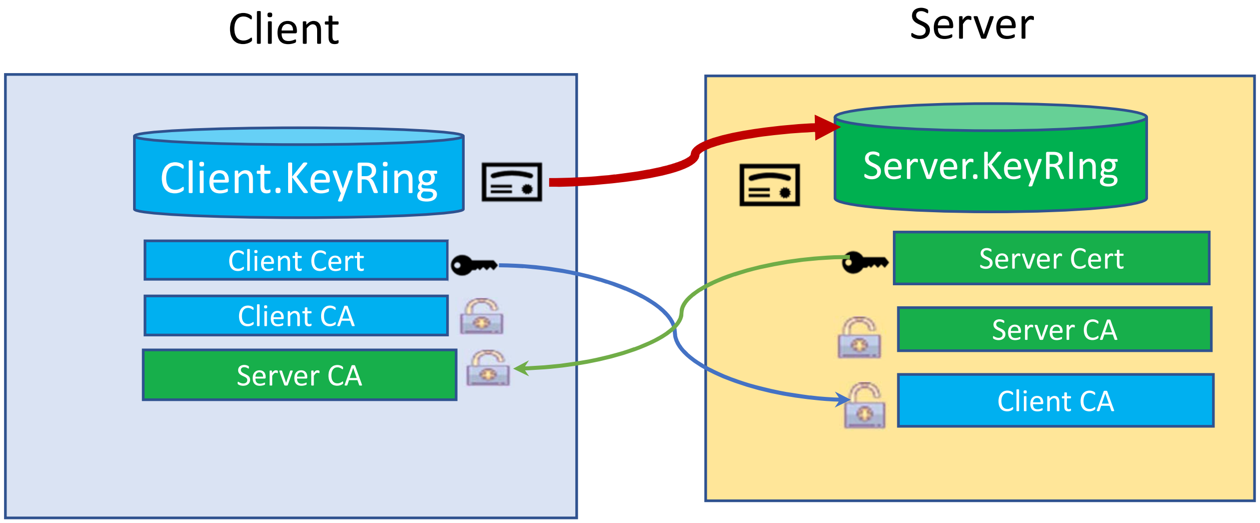
- Controlling access (**Authorization)**
  - Is the authenticated identity authorized to access to z/OS Connect
  - Is the authenticated identity authorized to access a specific API, Services, etc.

-

Basic TLS Handshake Flow

TLS handshake –

Encryption/Message Integrity



safkeyring:///**KeyRing** v safkeyring:**//owner/KeyRing**

RACF FACILITY resources

- IRR.DIGTCERT.LISTRING
  - READ to list your own key ring
  - UPDATE to list another user's key ring
- IRR.DIGTCERT.LIST
  - READ to list your own certificate
  - UPDATE to list another user's certificate
  - CONTROL to list SITE of CERTAUTH certificates

🔑 Certificate with a private key*

🔒 Certificate Authority (CA) certificate chain#

*For server and/or mutual authentication to work, the endpoint sending its server or client certificate must use a personal certificate with a private key. The private key is required to decrypt (or encrypt) a message digest that is sent from the other endpoint during the handshake flow. Generation of a message digest also requires access to the CA certificate used to sign the certificate.

#Refers to the set or of certificates used to issue the server or client personal certificate including any intermediate certificates all the way to the root CA.

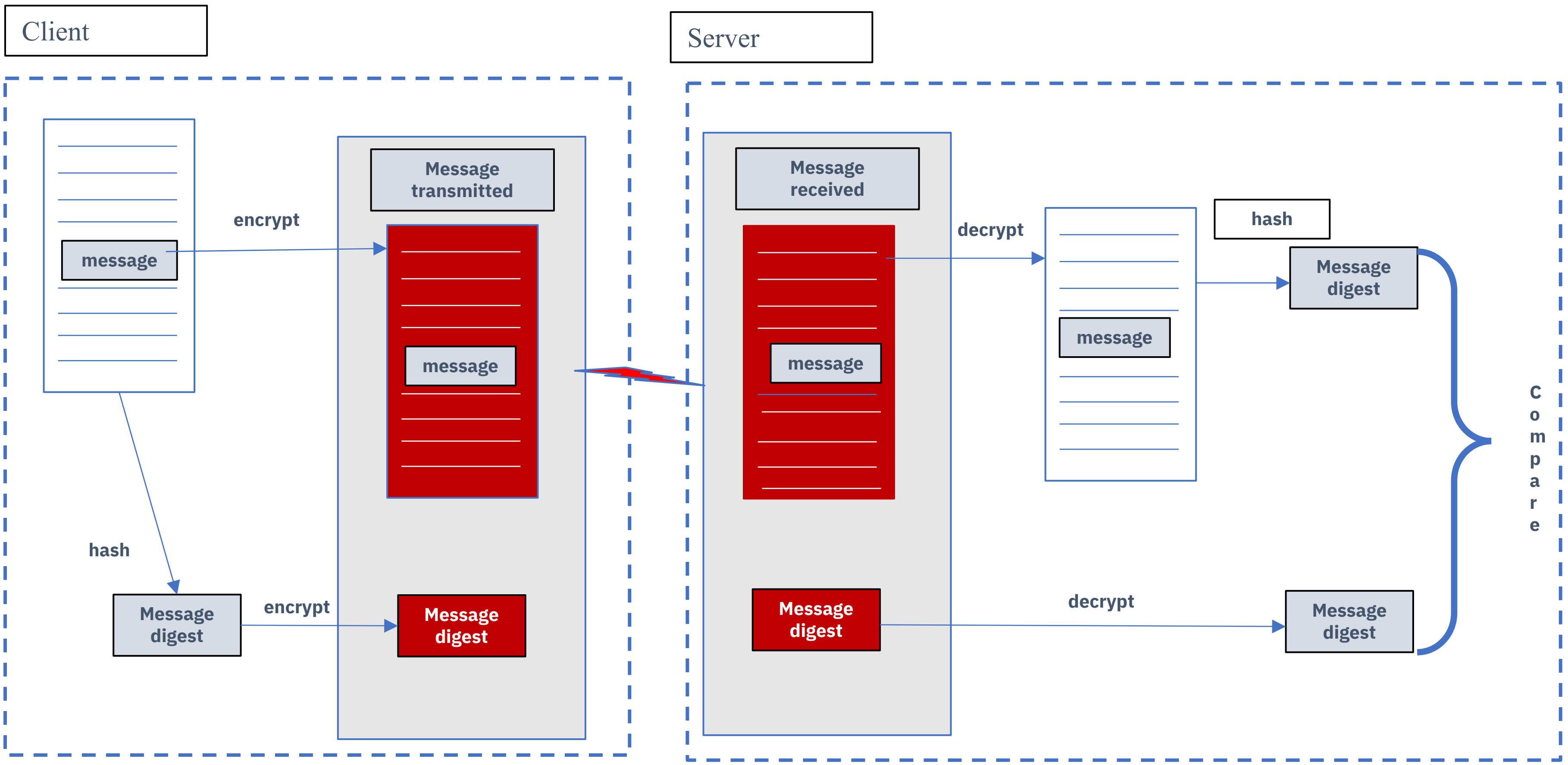
Tech/Tip: cURL trace of a TLS Handshake

```
* successfully set certificate verify locations:
*   CAfile: certauth.pem
  CApath: none
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384
* Server certificate:
*  subject: O=IBM; OU=LIBERTY; CN=wg31.washington.ibm.com
*  start date: Dec 23 04:00:00 2020 GMT
*  expire date: Jan  1 03:59:59 2023 GMT
*  common name: wg31.washington.ibm.com (matched)
*  issuer: OU=LIBERTY; CN=CA for Liberty
*  SSL certificate verify ok.
```

* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
~~* TLSv1.2 (IN), TLS handshake, Request CERT (13):~~
* TLSv1.2 (IN), TLS handshake, Server finished (14):
~~* TLSv1.2 (OUT), TLS handshake, Certificate (11):~~
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
~~* TLSv1.2 (OUT), TLS handshake, CERT verify (15):~~
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (01):

TLS 1.2 https://tools.ietf.org/html/rfc5246
TLS 1.3 https://tools.ietf.org/html/rfc8446

Message Integrity and Encryption



mitchj@us.ibm.com

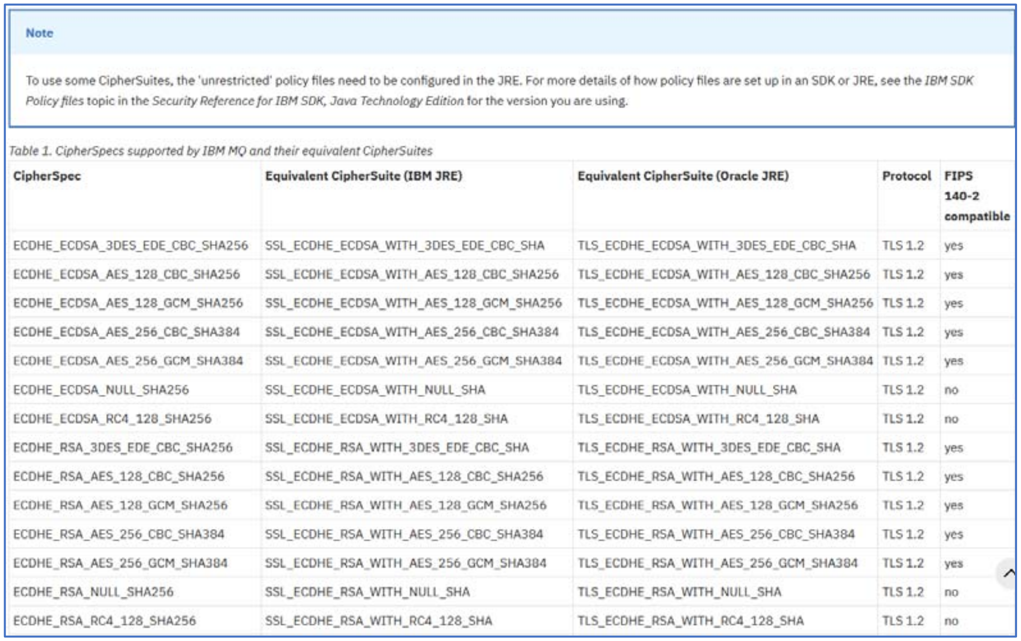
Tech/Tip: A note on cipher suite names

A CipherSuite is a suite of cryptographic algorithms used by a TLS connection. A suite comprises three distinct algorithms:

- The key exchange and authentication algorithm, used during the handshake
- The encryption algorithm, used to encipher the data
- The MAC (Message Authentication Code) algorithm, used to generate the message digest

There are several options for each component of the suite, but only certain combinations are valid when specified for a TLS connection. The name of a valid CipherSuite defines the combination of algorithms used. For example, the CipherSuite ***TLS_RSA_WITH_AES_128_CBC_SHA*** specifies:

- The RSA key exchange and authentication algorithm
- The AES encryption algorithm, using a 128-bit key and cipher block chaining (CBC) mode
- The SHA-1 Message Authentication Code (MAC)



Note

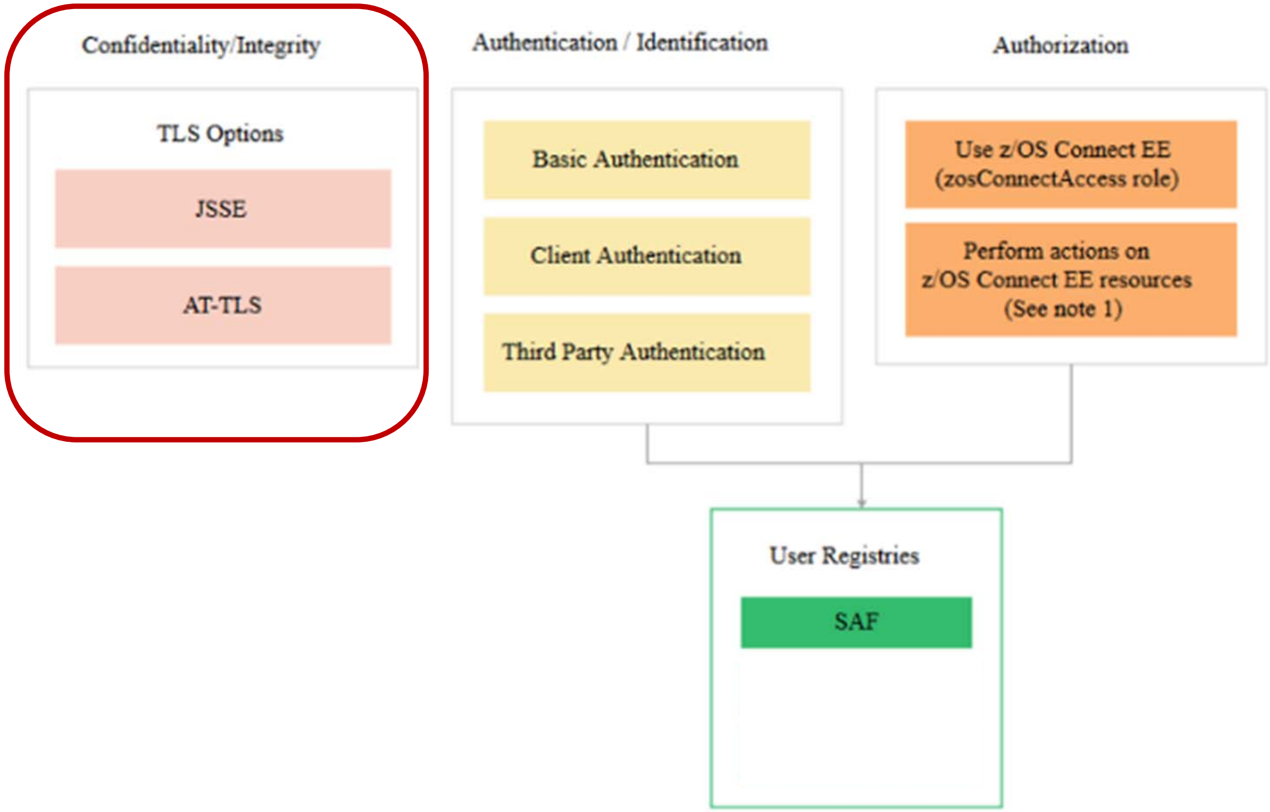
To use some CipherSuites, the 'unrestricted' policy files need to be configured in the JRE. For more details of how policy files are set up in an SDK or JRE, see the *IBM SDK Policy files* topic in the *Security Reference for IBM SDK, Java Technology Edition* for the version you are using.

Table 1. CipherSpecs supported by IBM MQ and their equivalent CipherSuites

CipherSpec	Equivalent CipherSuite (IBM JRE)	Equivalent CipherSuite (Oracle JRE)	Protocol	FIPS 140-2 compatible
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	no
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	no
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	no
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	no

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.dev.doc/q113210_.htm

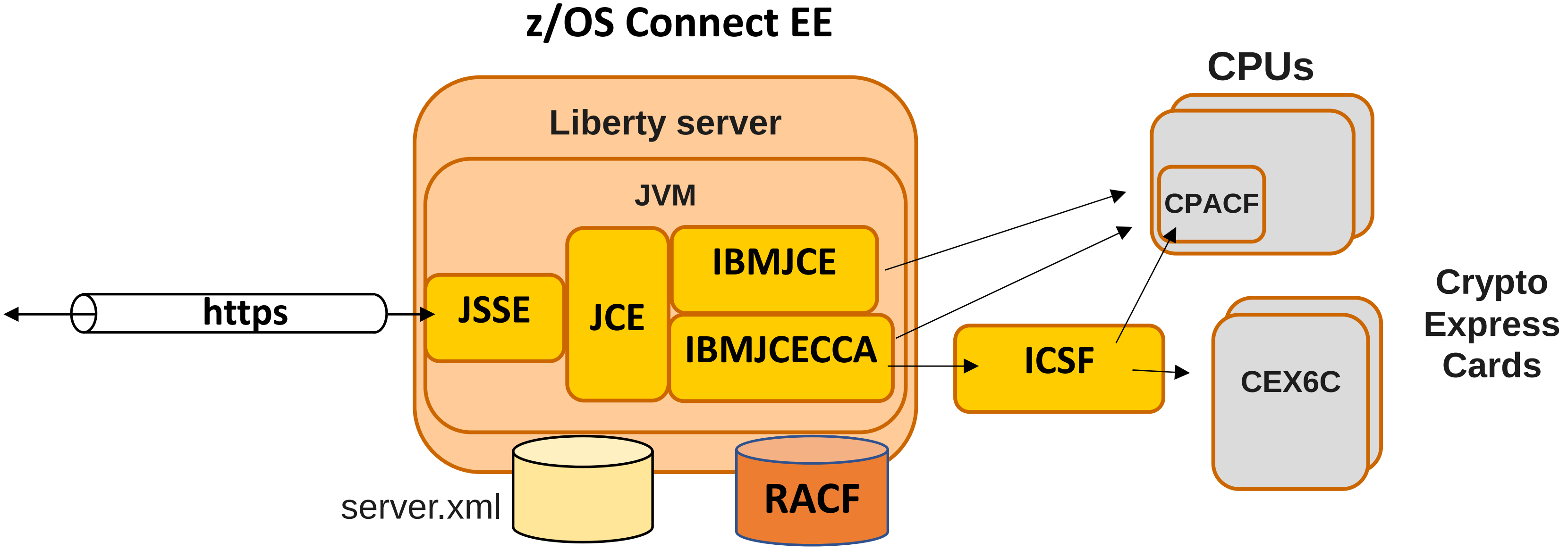
Liberty and z/OS Connect EE security options


Confidentiality/Integrity
TLS Options
JSSE
AT-TLS
Authentication / Identification
Basic Authentication
Client Authentication
Third Party Authentication
Authorization
Use z/OS Connect EE (zosConnectAccess role)
Perform actions on z/OS Connect EE resources (See note 1)
User Registries
SAF

The actions which can be controlled by authorization are deploying, querying, updating, starting, stopping and deleting of APIs, services and API requesters.

Using JSSE with z/OS Connect EE

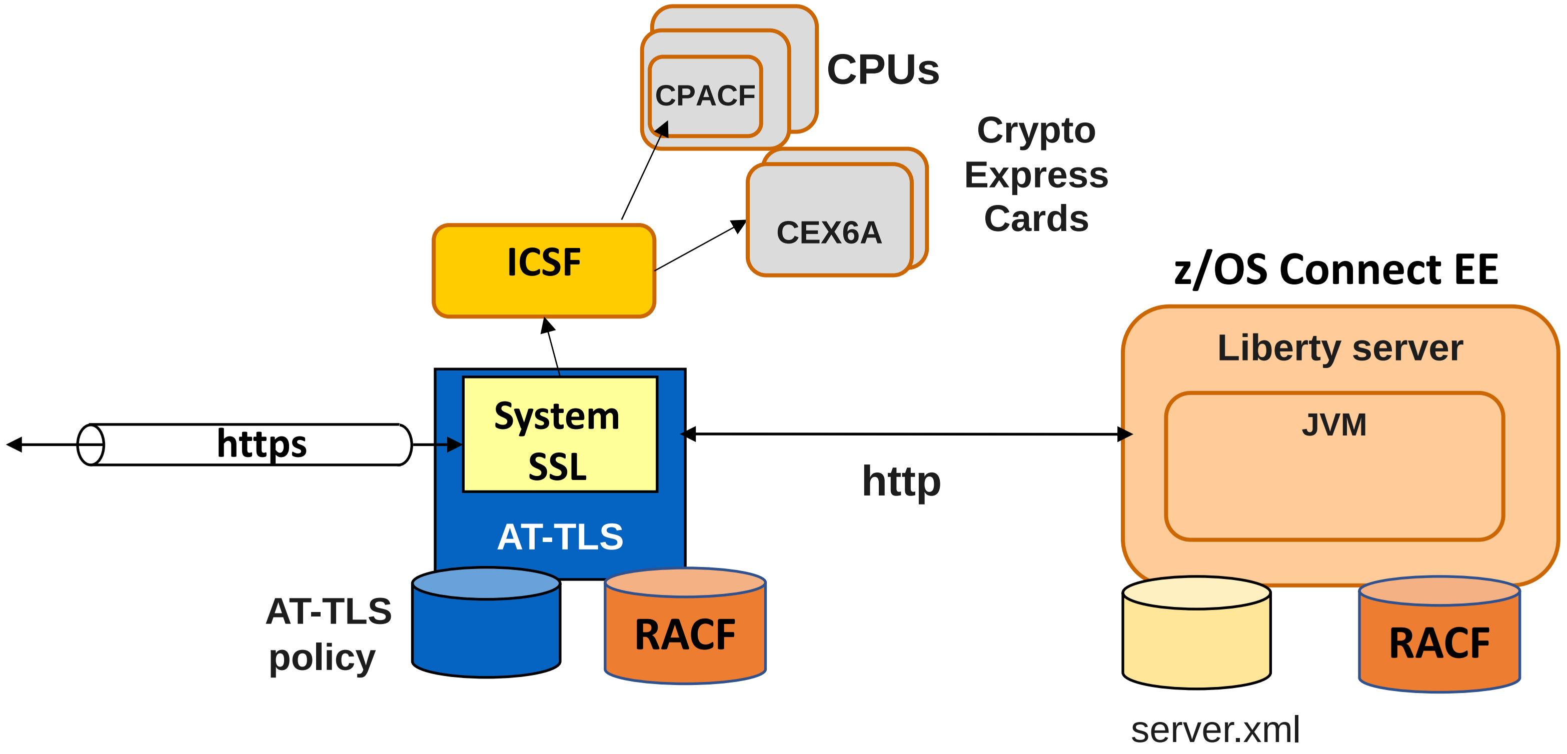
The server XML configuration defines the HTTPS ports, key rings, JSSE attributes, etc.



- z/OS Connect EE support for TLS is based on **Liberty** server support
- **Java Secure Socket Extension** (JSSE) API provides framework and Java implementation of 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**
- The JCE providers access CPACF (**CP Assist for Cryptographic Functions)** directly, therefore keep your Java service levels current.

Using AT-TLS with z/OS Connect EE

The server XML configuration defines no HTTPS ports, key rings, JSSE attributes

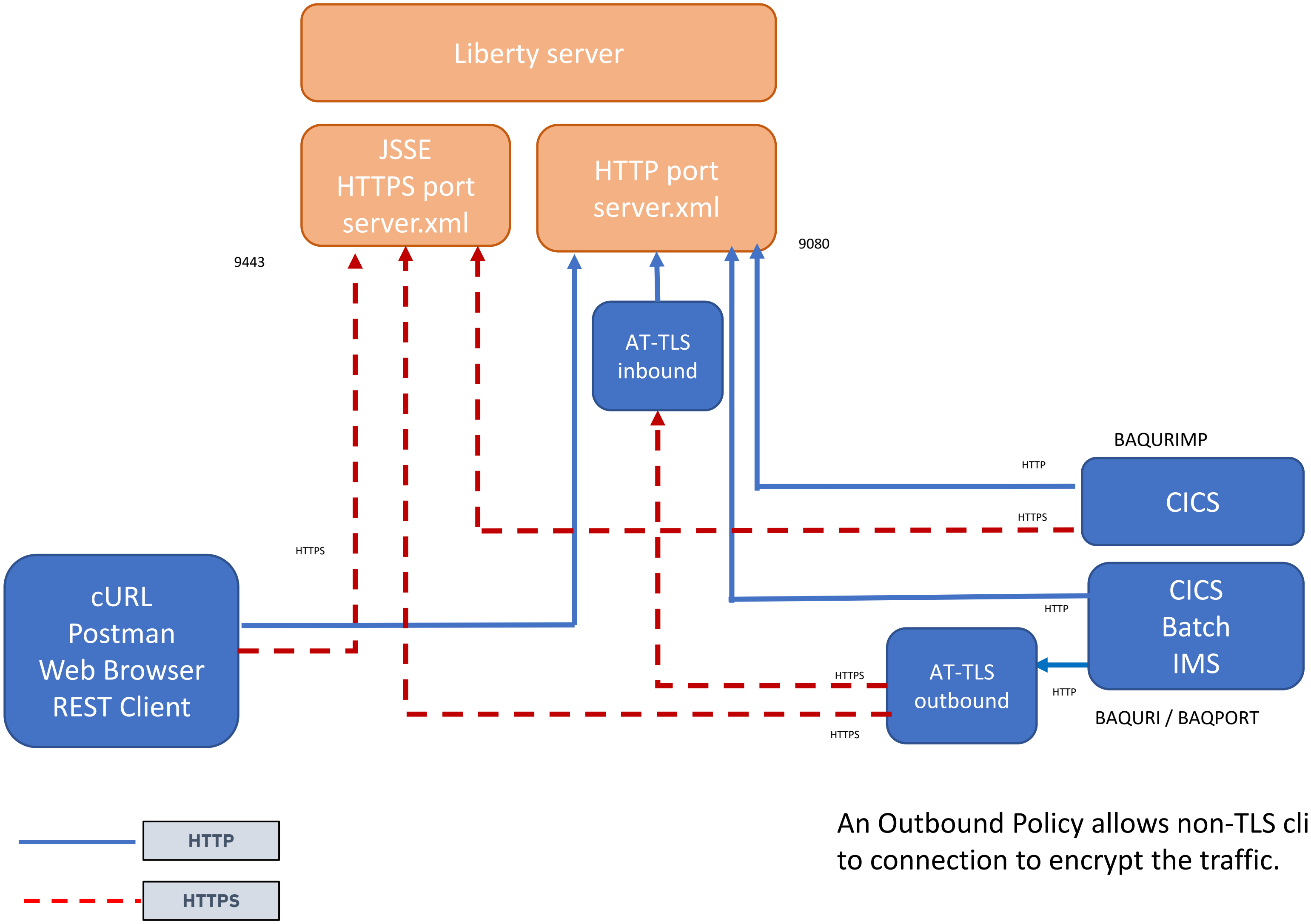


- **Application Transparent TLS** (AT-TLS) creates a secure session on behalf of z/OS Connect
- Only define http ports in server.xml (z/OS Connect does not know that TLS session exists)
- Define TLS protection for all applications (including z/OS Connect) in **AT-TLS policy**
- AT-TLS uses **System SSL**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

JSSE and AT-TLS comparison

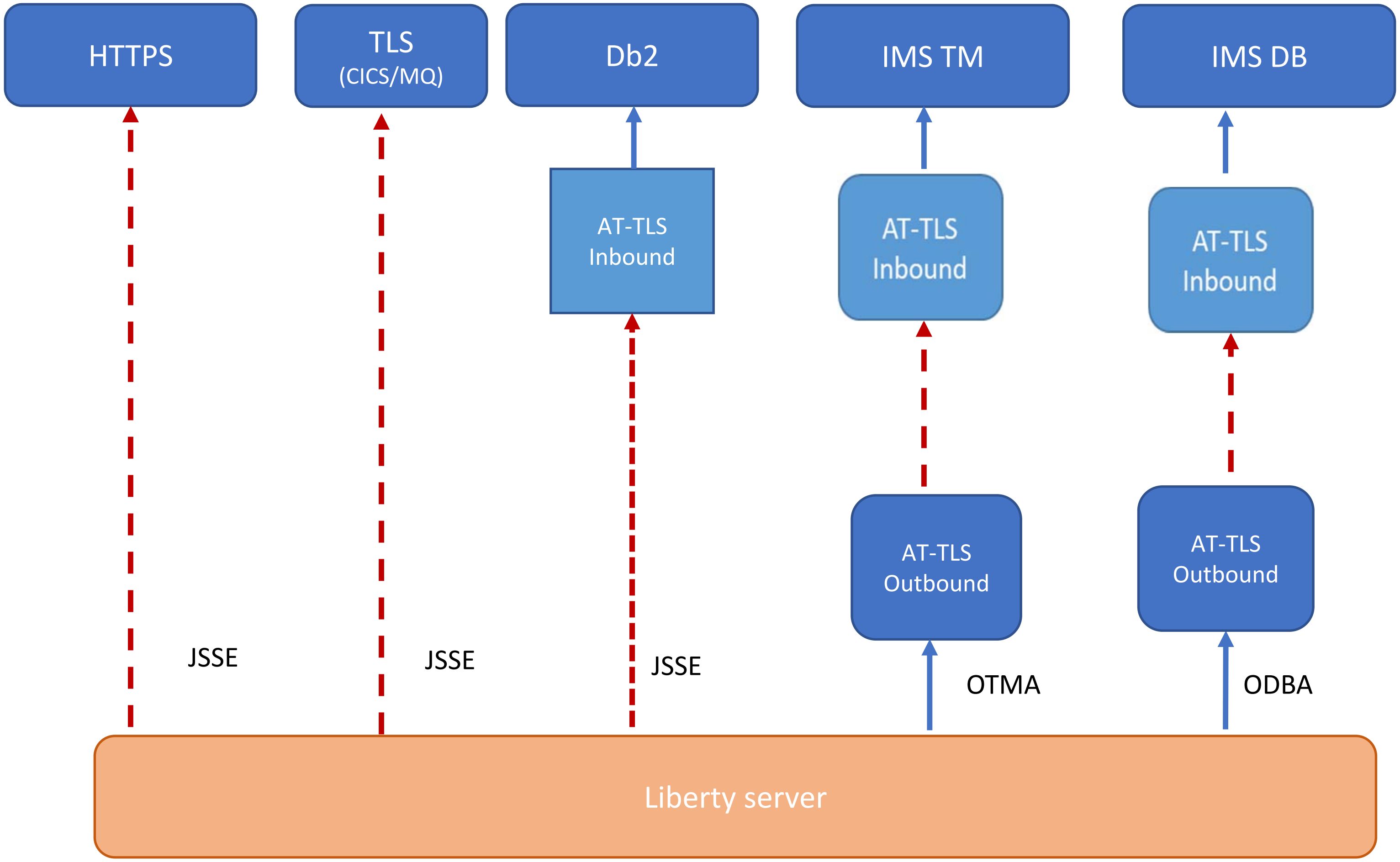
Capability	Description	JSSE	AT-TLS
Server authentication	Verification of z/OS Connect server certificate by client	Yes	Yes
Mutual authentication	Verification of client certificate by z/OS Connect	Yes	Yes
TLS client authentication	Use of client certificate for authentication	Yes	**No**
Support for requireSecure 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 TLS session	To reduce number of full handshakes	Yes	Yes
Shared TLS sessions	To share TLS 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

TLS client encryption to a Liberty server scenarios



An Outbound Policy allows non-TLS clients to connection to encrypt the traffic.

TLS encryptions from a zCEE server (HTTPS/native TLS/OTMA/ODBA) scenarios



Inbound Policies Provide TLS support for incoming requests to Db2 and IMS Connect

Outbound Policies provide TLS support for request outbound from a z/OS Connect server

Let’s explore using TLS for encryption and data integrity in various scenarios

Using the contents of these key rings

```
Digital ring information for user LIBSERV:
  Ring:
       >zCEE.KeyRing<
  Certificate Label Name                 Cert Owner     USAGE      DEFAULT
  --------------------------------       ------------   --------   -------
  zCEE CA                                CERTAUTH       CERTAUTH     NO
  Liberty CA                             CERTAUTH       CERTAUTH     NO
  zCEE Client Cert                       ID(LIBSERV)    PERSONAL     YES
  DB2 CA                                 CERTAUTH       CERTAUTH     NO
  MQ CA                                  CERTAUTH       CERTAUTH     NO
  CICS CA                                CERTAUTH       CERTAUTH     NO

  Ring:
       >Liberty.KeyRing<
  Certificate Label Name                 Cert Owner     USAGE      DEFAULT
  --------------------------------       ------------   --------   -------
  Liberty Server Cert                    ID(LIBSERV)    PERSONAL     YES
  Liberty CA                             CERTAUTH       CERTAUTH     NO
  zCEE CA                                CERTAUTH       CERTAUTH     NO
  CICS CA                                CERTAUTH       CERTAUTH     NO

Digital ring information for user CICSSTC:
  Ring:
       >CICS.KeyRing<
  Certificate Label Name                 Cert Owner     USAGE      DEFAULT
  --------------------------------       ------------   --------   -------
  CICS CA                                CERTAUTH       CERTAUTH     NO
  CICS Client Cert                       ID(CICSSTC)    PERSONAL     YES
  Liberty CA                             CERTAUTH       CERTAUTH     NO
  zCEE CA                                CERTAUTH       CERTAUTH     NO

Digital ring information for user DB2USER:
  Ring:
       >Db2.KeyRing<
  Certificate Label Name                 Cert Owner     USAGE      DEFAULT
  --------------------------------       ------------   --------   -------
  DB2 CA                                 CERTAUTH       CERTAUTH     NO
  zCEE CA                                CERTAUTH       CERTAUTH     NO
  DB2USER                                ID(DB2USER)    PERSONAL     YES
```

Outbound key ring

Inbound key ring

Note: when more than one personal certificate is connected to a key ring. Use the SSL repertoire *serverKeyAlias* or *clientKeyAlias* attributes to select the personal certificate to be used in a handshake.

Using this Liberty JSSE server XML configuration

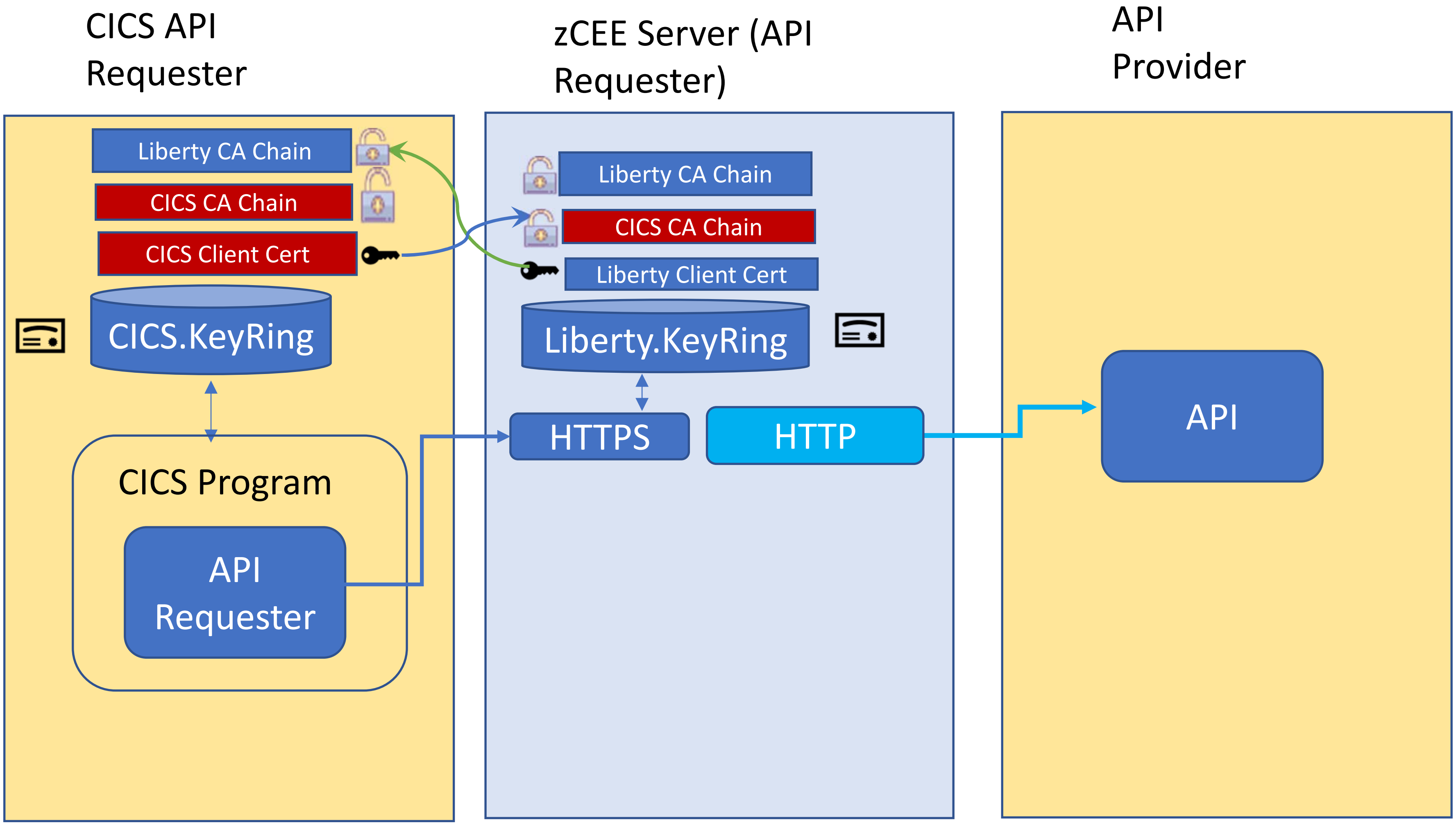
```
<!-- Enable features -->
   <featureManager>
        <feature>transportSecurity-1.0</feature>
   </featureManager>

   <sslDefault sslRef="DefaultSSLSettings"
     outboundSSLRef="OutboundSSLSettings" />

   <ssl id="DefaultSSLSettings"
      keyStoreRef="CellDefaultKeyStore"
      trustStoreRef="CellDefaultKeyStore“
      clientAuthenticationSupported="true"
      clientAuthentication="true“
      serverKeyAlias=“Liberty Server Cert"/>

   <keyStore id="CellDefaultKeyStore"
      location="safkeyring:///Liberty.KeyRing"
      password="password" type="JCERACFKS"
      fileBased="false" readOnly="true" />

   <ssl id="OutboundSSLSettings"
      keyStoreRef="OutboundKeyStore"
      trustStoreRef="OutboundKeyStore"/>

   <keyStore id="OutboundKeyStore"
      location="safkeyring:///zCEE.KeyRing"
      password="password" type="JCERACFKS"
      clientKeyAlias="Liberty Client Cert"
      fileBased="false" readOnly="true" />
```

SSL repertoires

```
<zosconnect_authorizationServer sslCertRef=“SSL repertoire”/>
<zosconnect_cicsIpicConnection sslCertRef=“SSL repertoire”/>
<zosconnect_endpointConnect sslCertRef=“SSL repertoire”/>
<zosconnect_zosConnectRestClient sslCertRef=“SSL repertoire”/>
<zosconnect_zosConnectServiceRestClientConnection sslCertRef=“SSL repertoire”/>
```

F BAQSTRT,REFRESH,KEYSTORE

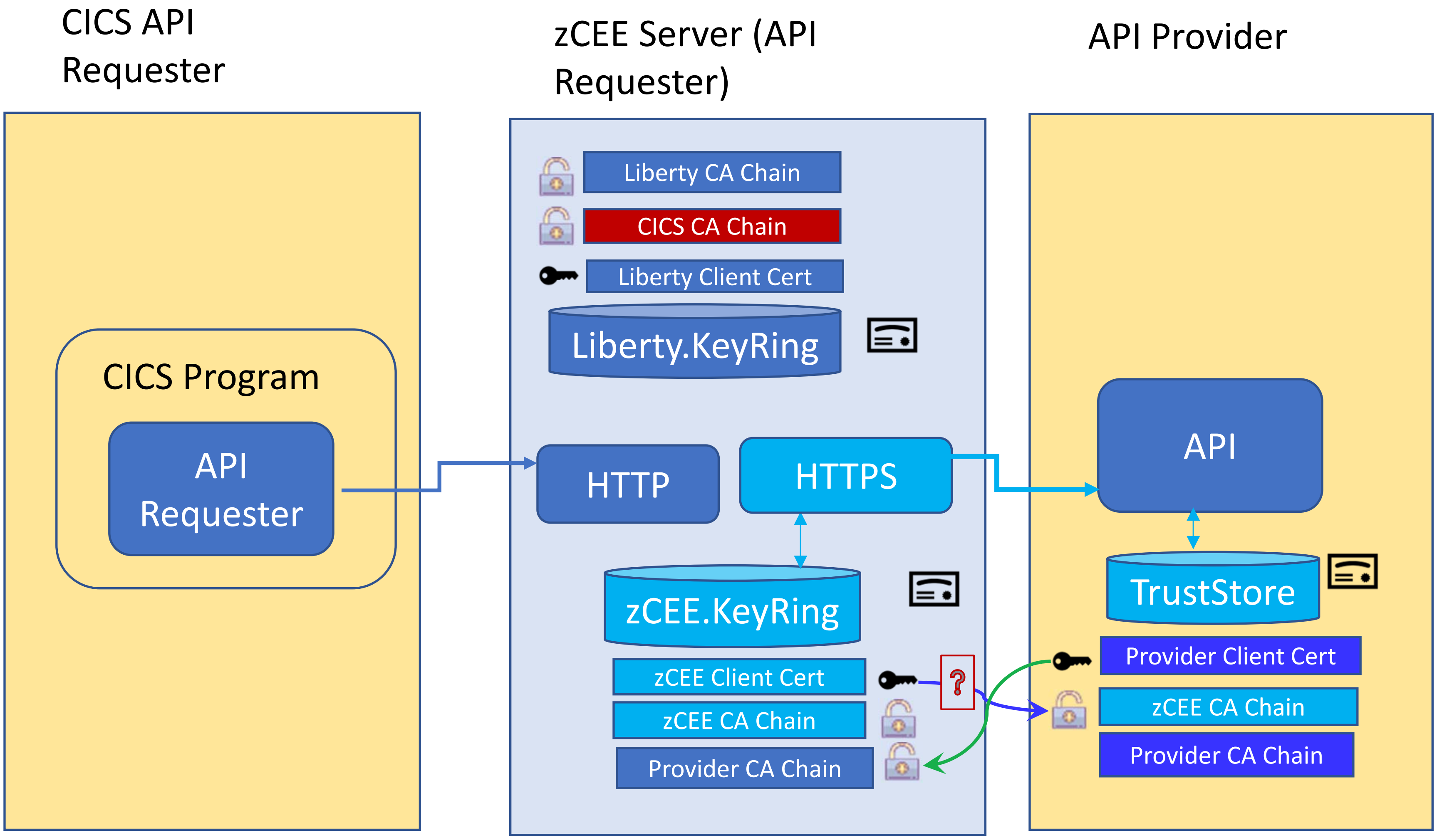
TLS handshake scenario (CICS inbound handshake)



```
KEYRING=CICS.KeyRing,
ENCRYPTION=STRONG,
MAXSSLTCBS=8,
SSLDELAY=600
```

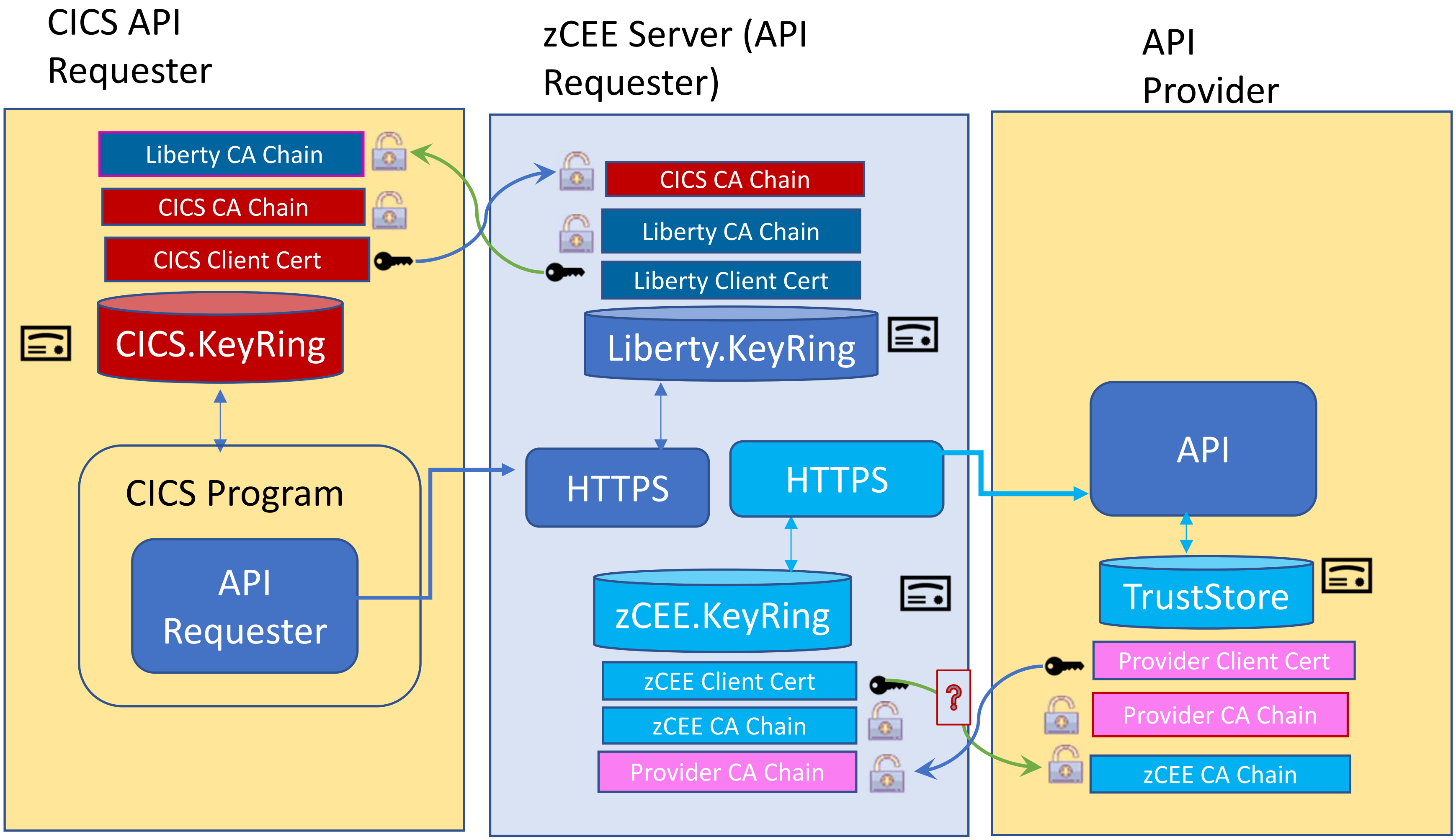
Tech/Tip: This is perhaps the best way to enable group member authorization from a CICS region to a z/OS Connect server.

TLS handshake scenario (outbound handshake)



? Question if this really needed, TLS encryption is independent of TLS authentication.

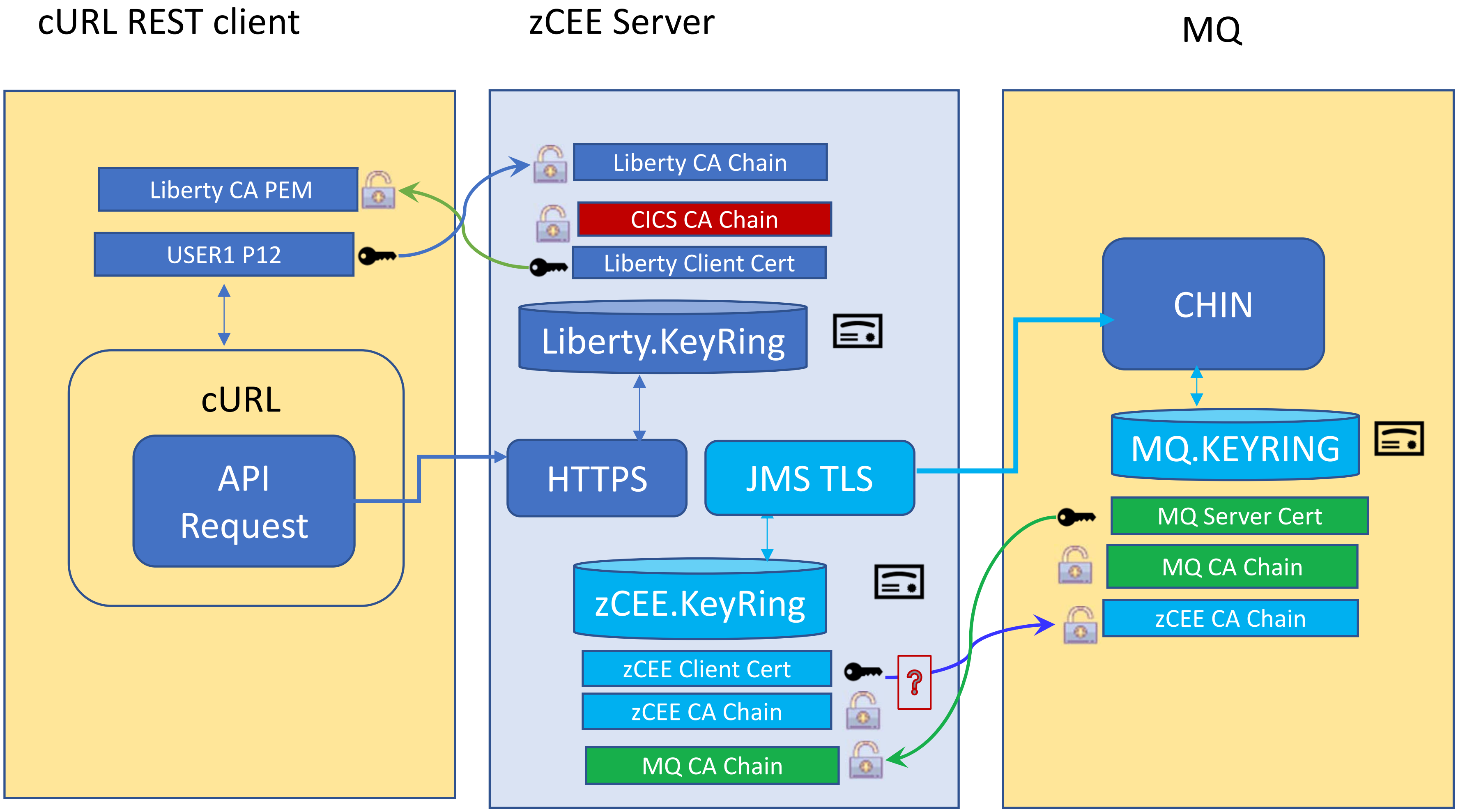
TLS handshake scenario (multiple handshakes)



```
KEYRING=CICS.KeyRing,
ENCRYPTION=STRONG,
MAXSSLTCBS=8,
SSLDELAY=600
```

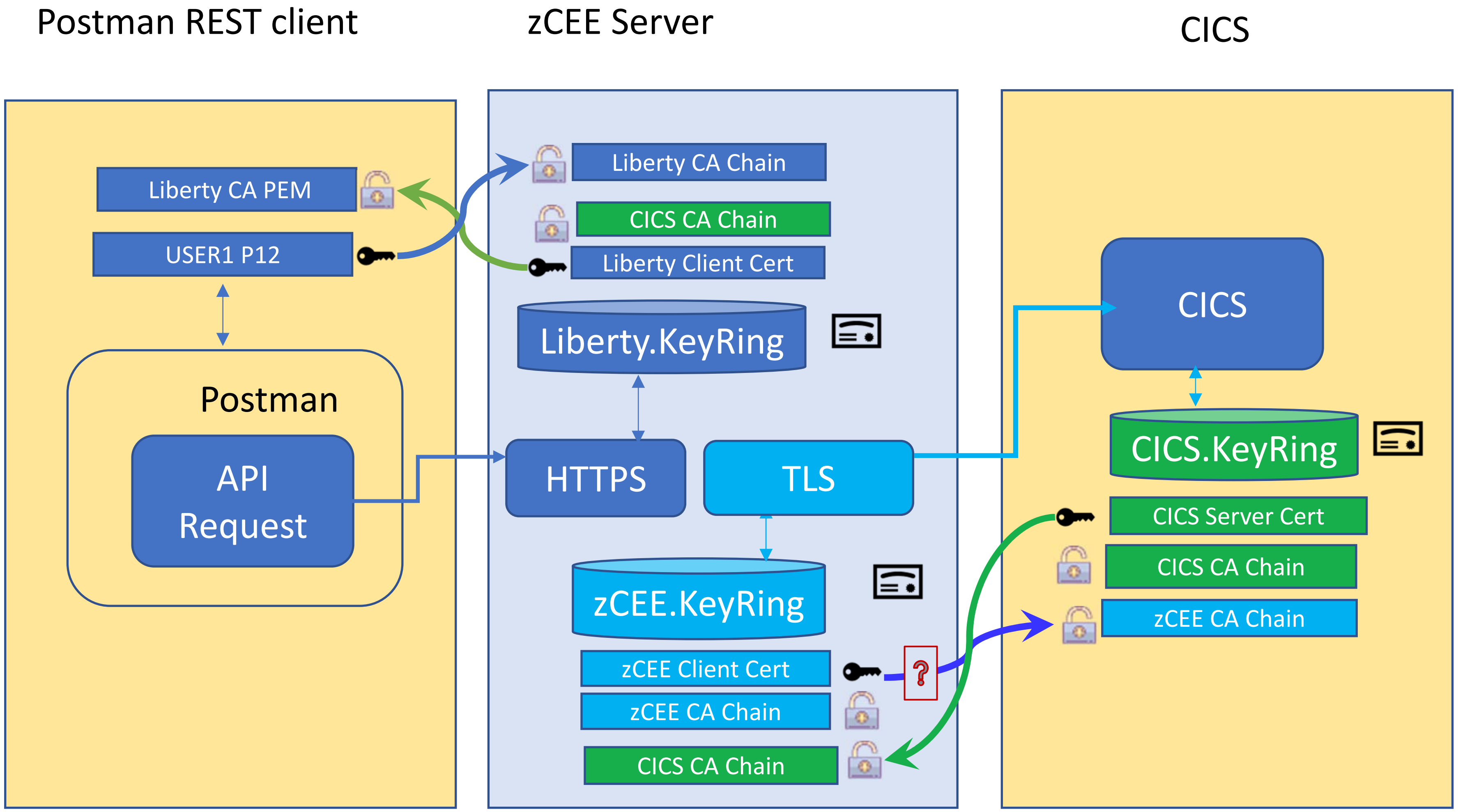
Remember, TLS encryption is independent of TLS authentication. So mutual authentication may not be needed.

TLS handshake scenario (cURL REST client)



```
RACDCERT ID(USER1) EXPORT(LABEL('USER1')) -
 DSN('USER1.USER1.P12') FORMAT(PKCS12DER) PASSWORD('secret')

RACDCERT CERTAUTH EXPORT(LABEL('Liberty CA')) -
 DSN('USER1.CERTAUTH.PEM')
```

TLS handshake scenario (Postman REST client)

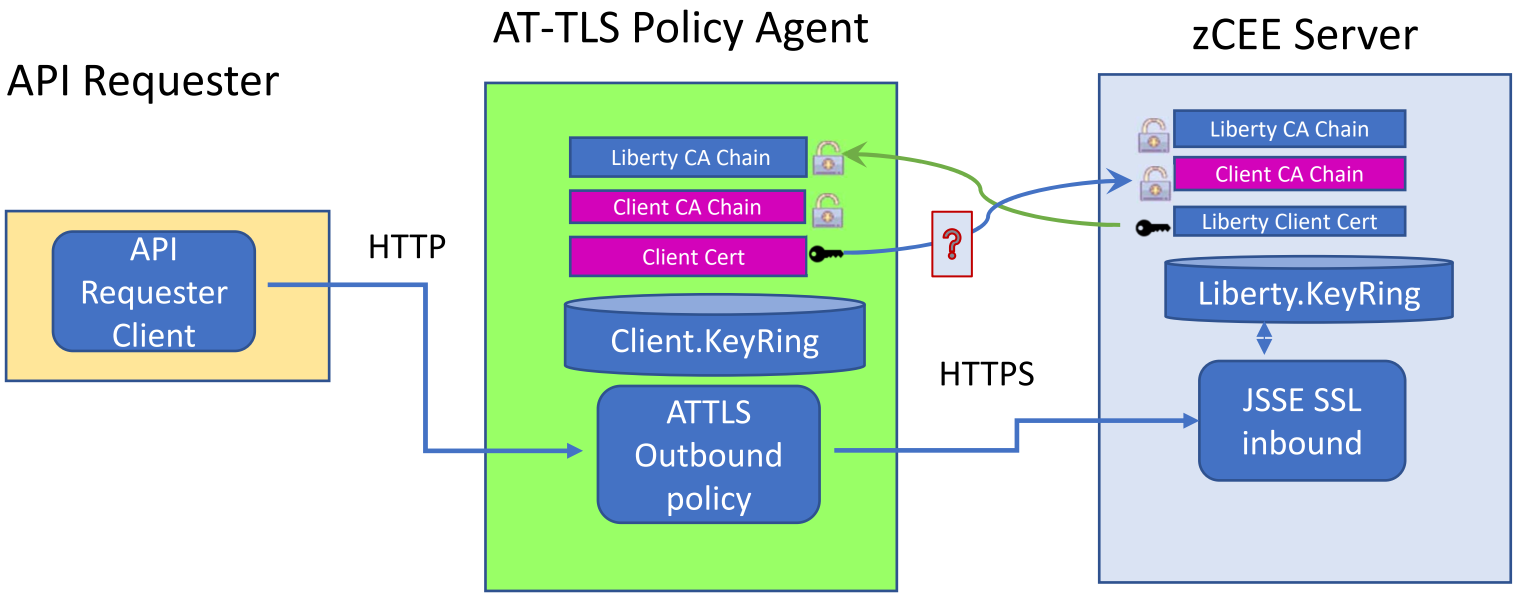


Note that CICS is AT-TLS aware, but we no longer have access to the personal certificate sent by the REST client

```
KEYRING=CICS.KeyRing,
ENCRYPTION=STRONG,
MAXSSLTCBS=8,
SSLDELAY=600
```

AT-TLS - outbound policy handshake scenarios

Policy Agent uses an outbound policy and acts a surrogate TLS client



```
<zosconnect_apiRequesters idAssertion="ASSERT_ONLY">
</zosconnect_apiRequesters>
```

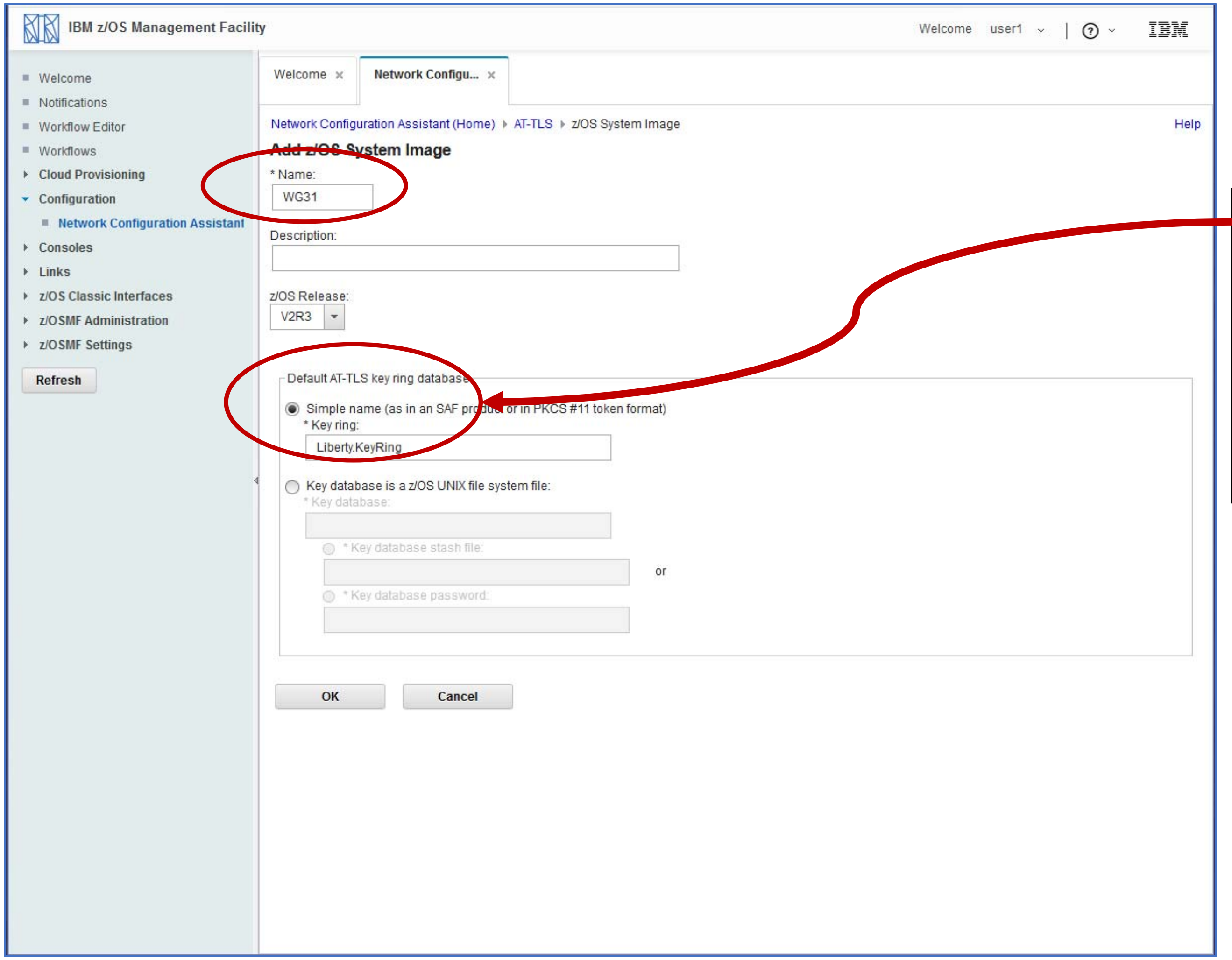
🔑 Certificate with a private key

🔒 Certificate Authority (CA) certificate

? Question if this really needed, remember TLS encryption is independent of TLS authentication.

AT-TLS – setting a default key ring

Using zOSMF Network Configuration Assistant to provide default Key Ring name



Add z/OS System Image

* Name: WG31

Description:

z/OS Release: V2R3

Default AT-TLS key ring database

● Simple name (as in an SAF product or in PKCS #11 token format)

* Key ring: Liberty.KeyRing

○ Key database is a z/OS UNIX file system file:

* Key database:

○ * Key database stash file:

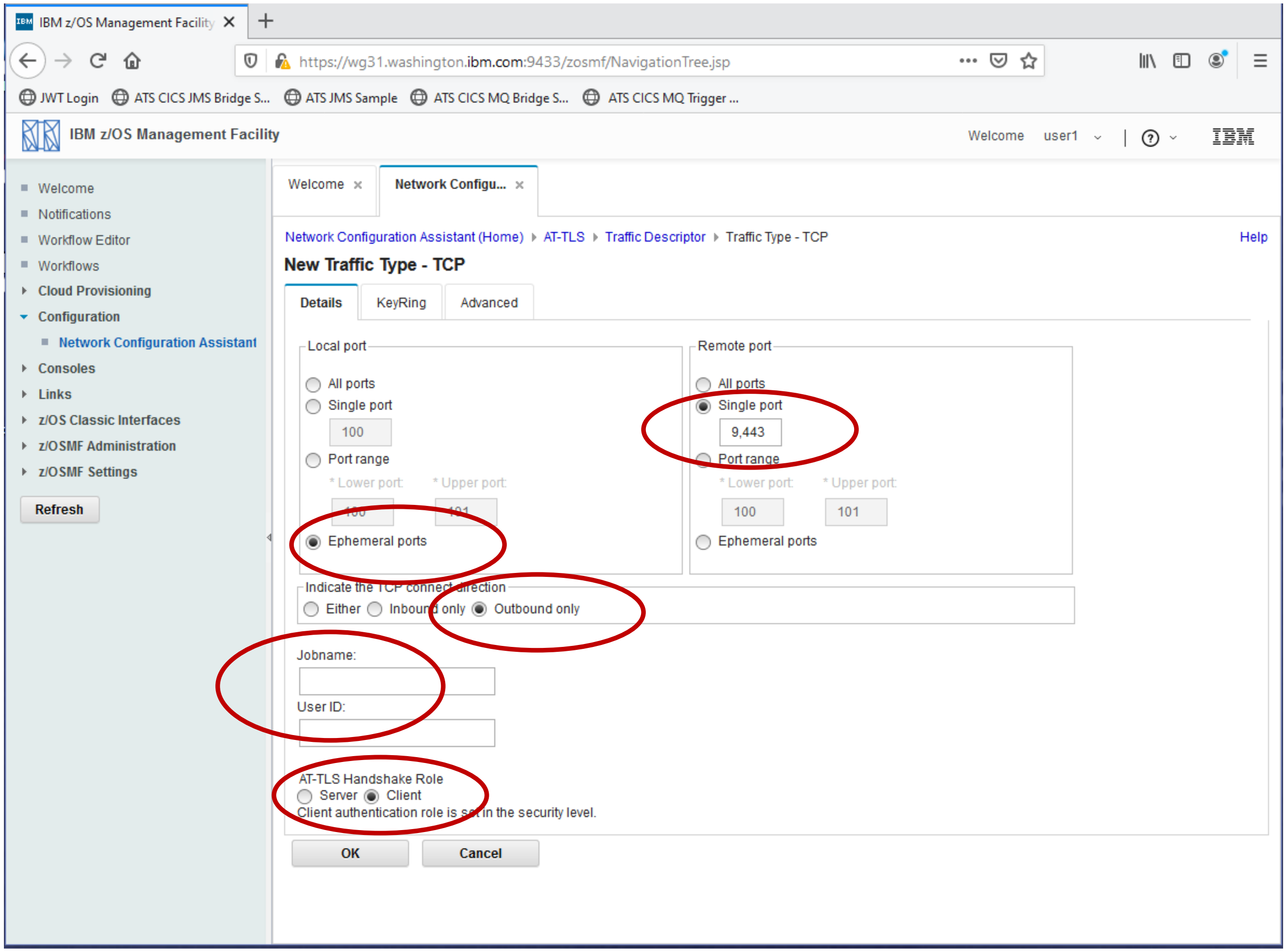
or

○ * Key database password:

An SAF key ring name is specified as "identity/keyring". The current identity is used if the identity field is omitted.

AT-TLS – configuring an outbound policy

Using zOSMF Network Configuration Assistant to configure traffic descriptor



Network Configuration Assistant (Home) ▸ AT-TLS ▸ Traffic Descriptor ▸ Traffic Type - TCP

New Traffic Type - TCP

Details | KeyRing | Advanced

Local port
- ○ All ports
- ○ Single port: 100
- ○ Port range: * Lower port: 100 * Upper port: 101
- ● Ephemeral ports

Remote port
- ○ All ports
- ● Single port: 9,443
- ○ Port range: * Lower port: 100 * Upper port: 101
- ○ Ephemeral ports

Indicate the TCP connect direction
○ Either ○ Inbound only ● Outbound only

Jobname:

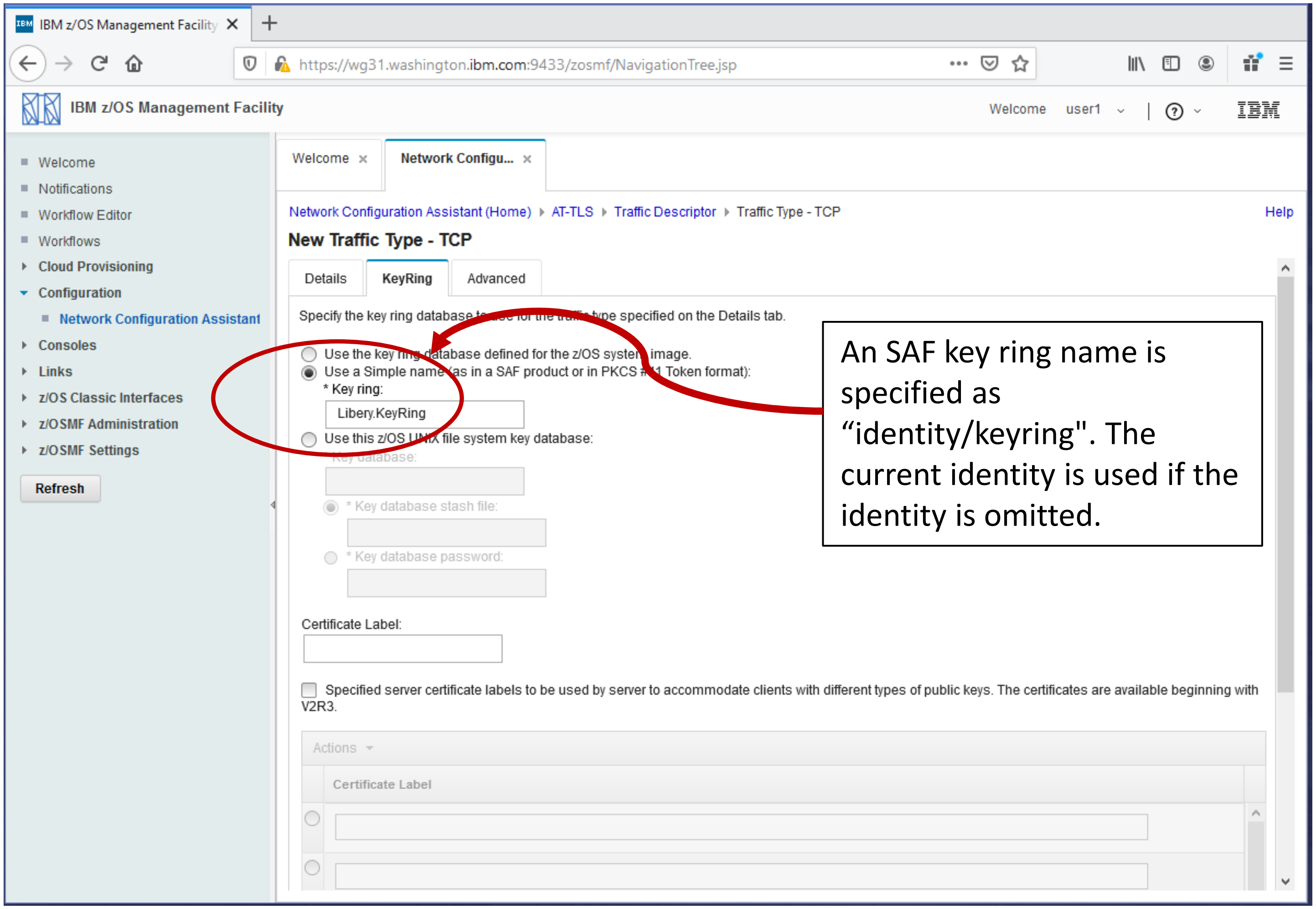
User ID:

AT-TLS Handshake Role
○ Server ● Client
Client authentication role is set in the security level.

OK Cancel

AT-TLS - setting a policy key ring

Using zOSMF Network Configuration Assistant to traffic descriptor Key Ring name



New Traffic Type - TCP

Specify the key ring database to use for the traffic type specified on the Details tab.

○ Use the key ring database defined for the z/OS system image.

● Use a Simple name (as in a SAF product or in PKCS #11 Token format):

* Key ring: Libery.KeyRing

○ Use this z/OS UNIX file system key database:

Key database:

● * Key database stash file:

○ * Key database password:

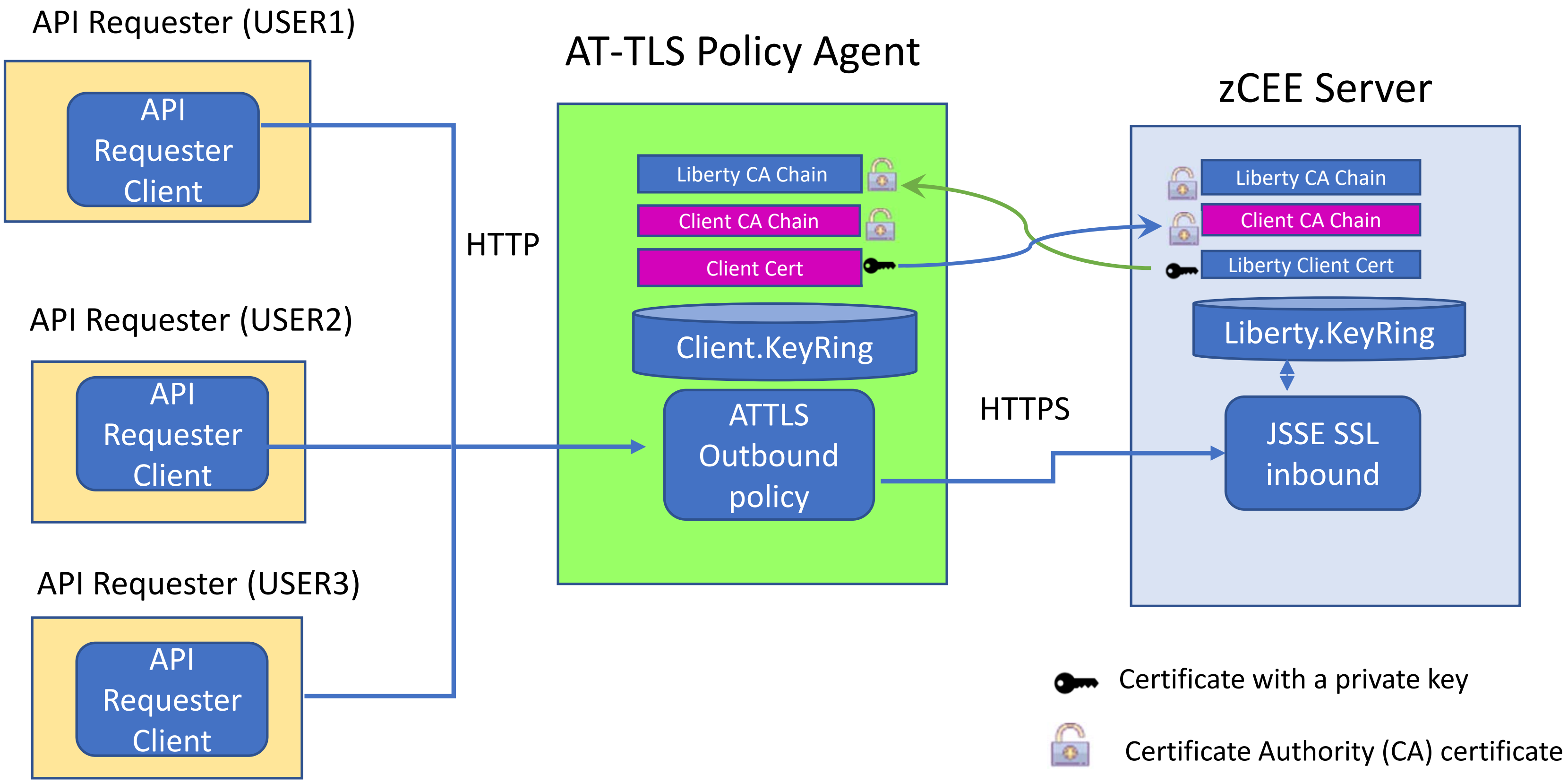
Certificate Label:

□ Specified server certificate labels to be used by server to accommodate clients with different types of public keys. The certificates are available beginning with V2R3.

An SAF key ring name is specified as "identity/keyring". The current identity is used if the identity is omitted.

AT-TLS - outbound policy handshake scenario

Use of a common key ring name with multiple client identities

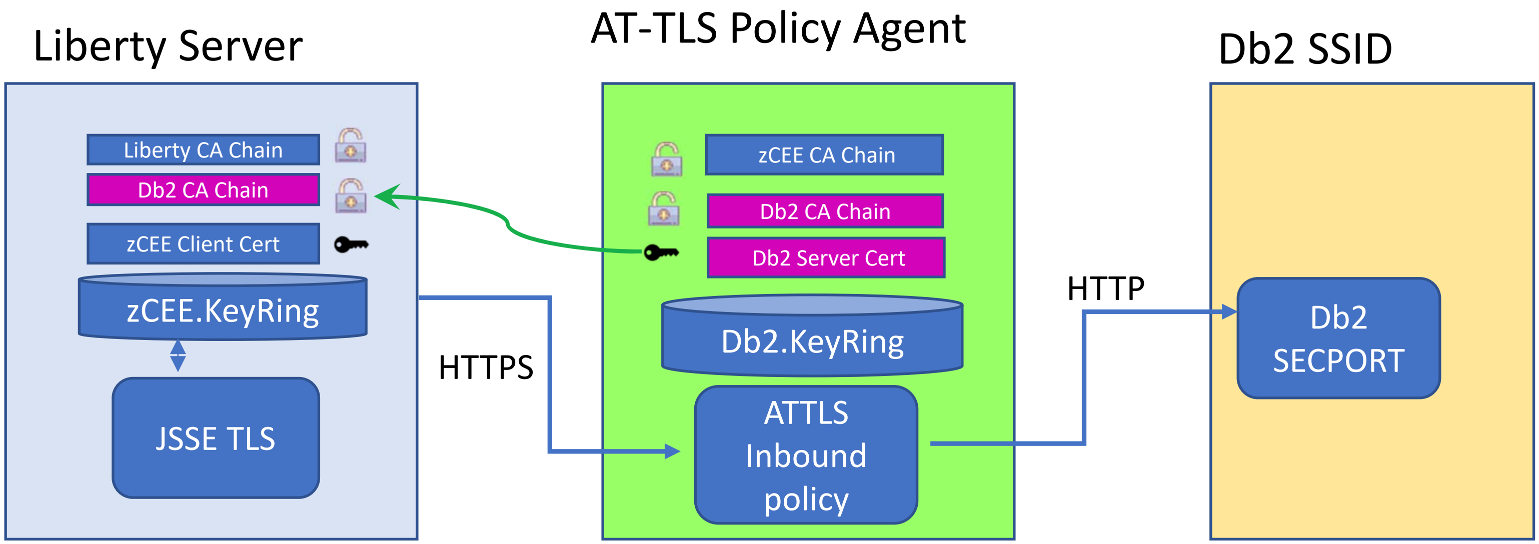


- Each user owns a keyring with the name Liberty.KeyRing.
- Each key ring has a different default client certificate for mutual authentication purposes.

This is a situation when AT-TLS mutual authentication has a benefit.

AT-TLS - inbound policy handshake scenario (Db2)

Policy Agent uses an inbound policy and acts a surrogate TLS server



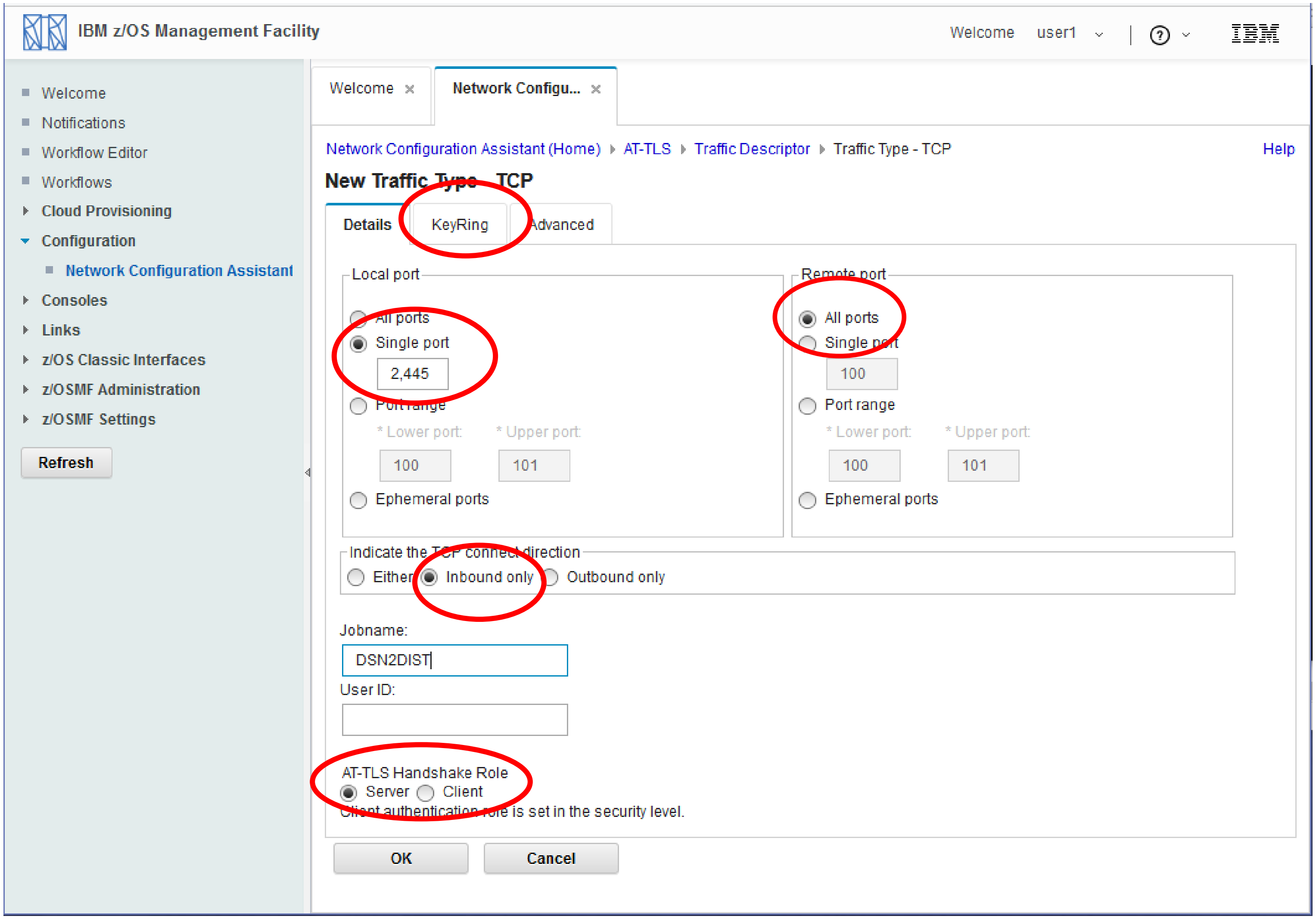
Note that DB2 is AT-TLS aware

🔑 Certificate with a private key

🔒 Certificate Authority (CA) certificate

AT-TLS – configuring an inbound policy

Using zOSMF Network Configuration Assistant



Network Configuration Assistant (Home) ▸ AT-TLS ▸ Traffic Descriptor ▸ Traffic Type - TCP

New Traffic Type - TCP

Details | KeyRing | Advanced

Local port
○ All ports
● Single port
2,445
○ Port range
* Lower port: 100 * Upper port: 101
○ Ephemeral ports

Remote port
● All ports
○ Single port
100
○ Port range
* Lower port: 100 * Upper port: 101
○ Ephemeral ports

Indicate the TCP connect direction
○ Either ● Inbound only ○ Outbound only

Jobname: DSN2DIST

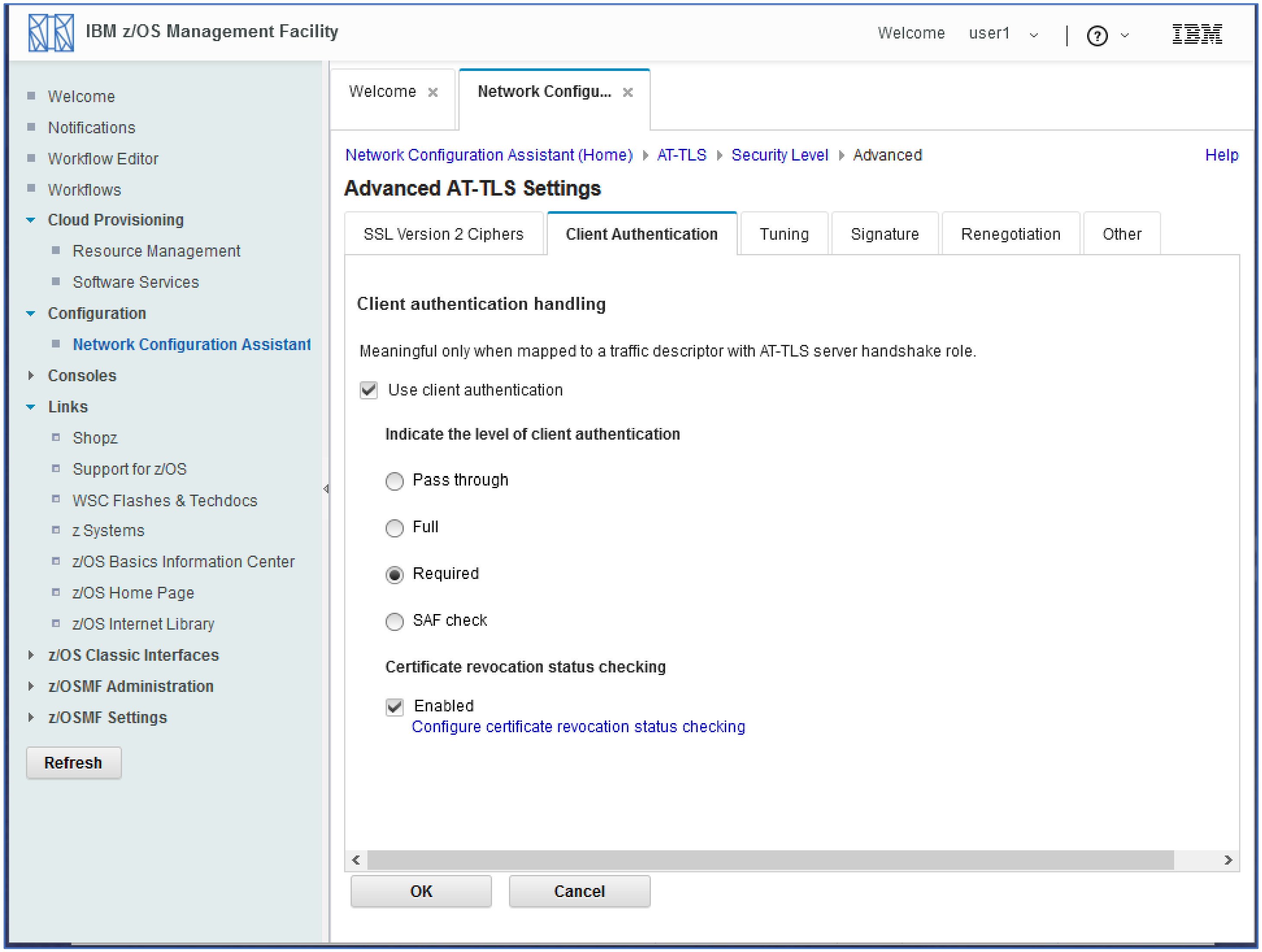
User ID:

AT-TLS Handshake Role
● Server ○ Client
Client authentication role is set in the security level.

OK | Cancel

AT-TLS – configuring client authentication

Using zOSMF Network Configuration Assistant (mutual authentication)



Advanced AT-TLS Settings

Client authentication handling

Meaningful only when mapped to a traffic descriptor with AT-TLS server handshake role.

☑ Use client authentication

Indicate the level of client authentication

○ Pass through

○ Full

● Required

○ SAF check

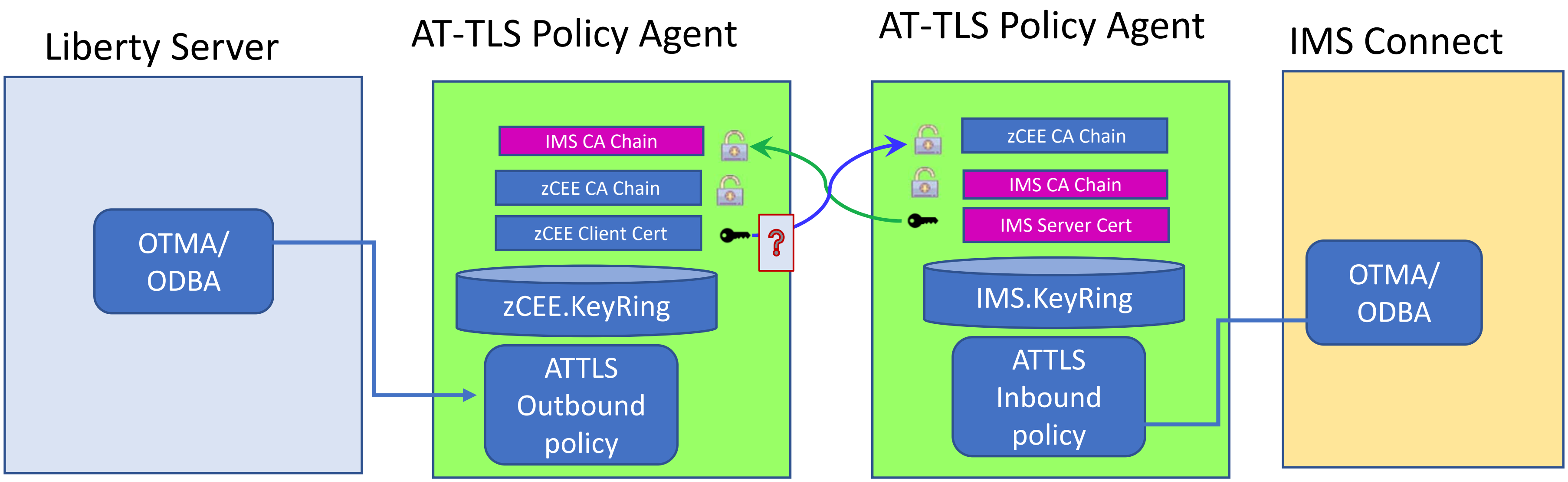
Certificate revocation status checking

☑ Enabled

Configure certificate revocation status checking

AT-TLS multiple policies handshake scenario (IMS)

Policy Agent uses both inbound and outbound policies and acts a surrogate TLS client with one and a TLS server with the other



Note that IMS is not AT-TLS aware

🔑 Certificate with a private key

🔒 Certificate Authority (CA) certificate

? Question if this really needed, TLS encryption is independent of TLS authentication.

Configuring TLS Encryption with JSSE

© 2017, 20

Ciphers

- 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 (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

For cipher details, see IBM SDK Java 8.0.0 Cipher Suites at URL
https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/ciphersuites.html

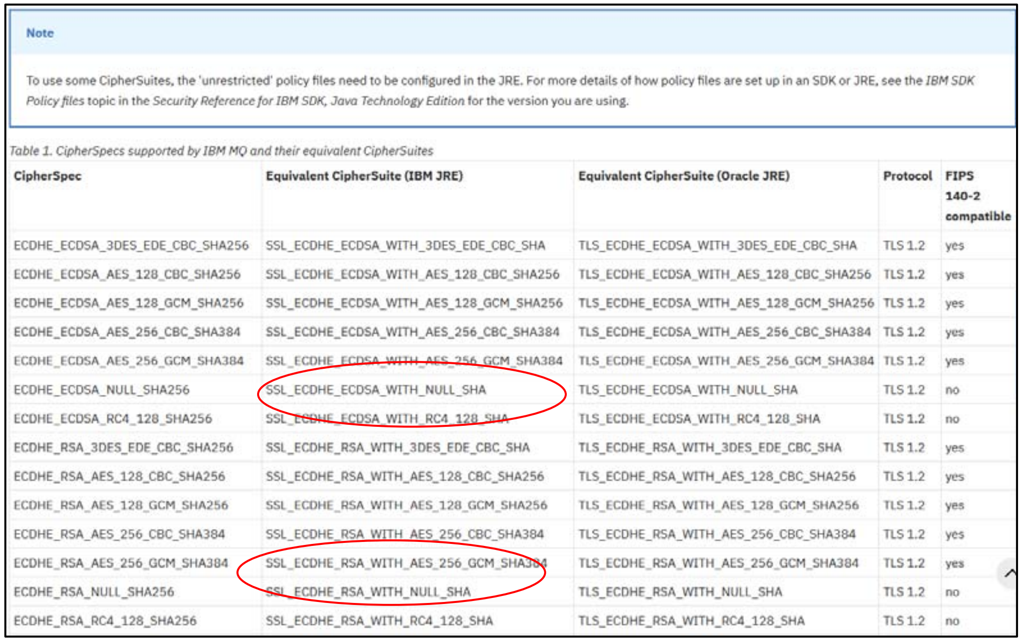
Tech/Tip: Cipher Suite names

A CipherSuite is a suite of cryptographic algorithms used by a TLS connection. A suite comprises three distinct algorithms:

- The key exchange and authentication algorithm, used during the handshake
- The encryption algorithm, used to encipher the data
- The MAC (Message Authentication Code) algorithm, used to generate the message digest

There are several options for each component of the suite, but only certain combinations are valid when specified for a TLS connection. The name of a valid CipherSuite defines the combination of algorithms used. For example, the CipherSuite ***TLS_RSA_WITH_AES_128_CBC_SHA*** specifies:

- The RSA key exchange and authentication algorithm
- The AES encryption algorithm, using a 128-bit key and cipher block chaining (CBC) mode
- The SHA-1 Message Authentication Code (MAC)



Note

To use some CipherSuites, the 'unrestricted' policy files need to be configured in the JRE. For more details of how policy files are set up in an SDK or JRE, see the *IBM SDK Policy files* topic in the *Security Reference for IBM SDK, Java Technology Edition* for the version you are using.

Table 1. CipherSpecs supported by IBM MQ and their equivalent CipherSuites

CipherSpec	Equivalent CipherSuite (IBM JRE)	Equivalent CipherSuite (Oracle JRE)	Protocol	FIPS 140-2 compatible
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	no
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	no
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	no
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	no

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.dev.doc/q113210_.htm

Tech/Tip: Cipher Suite numbers (CICS TCPIPService):

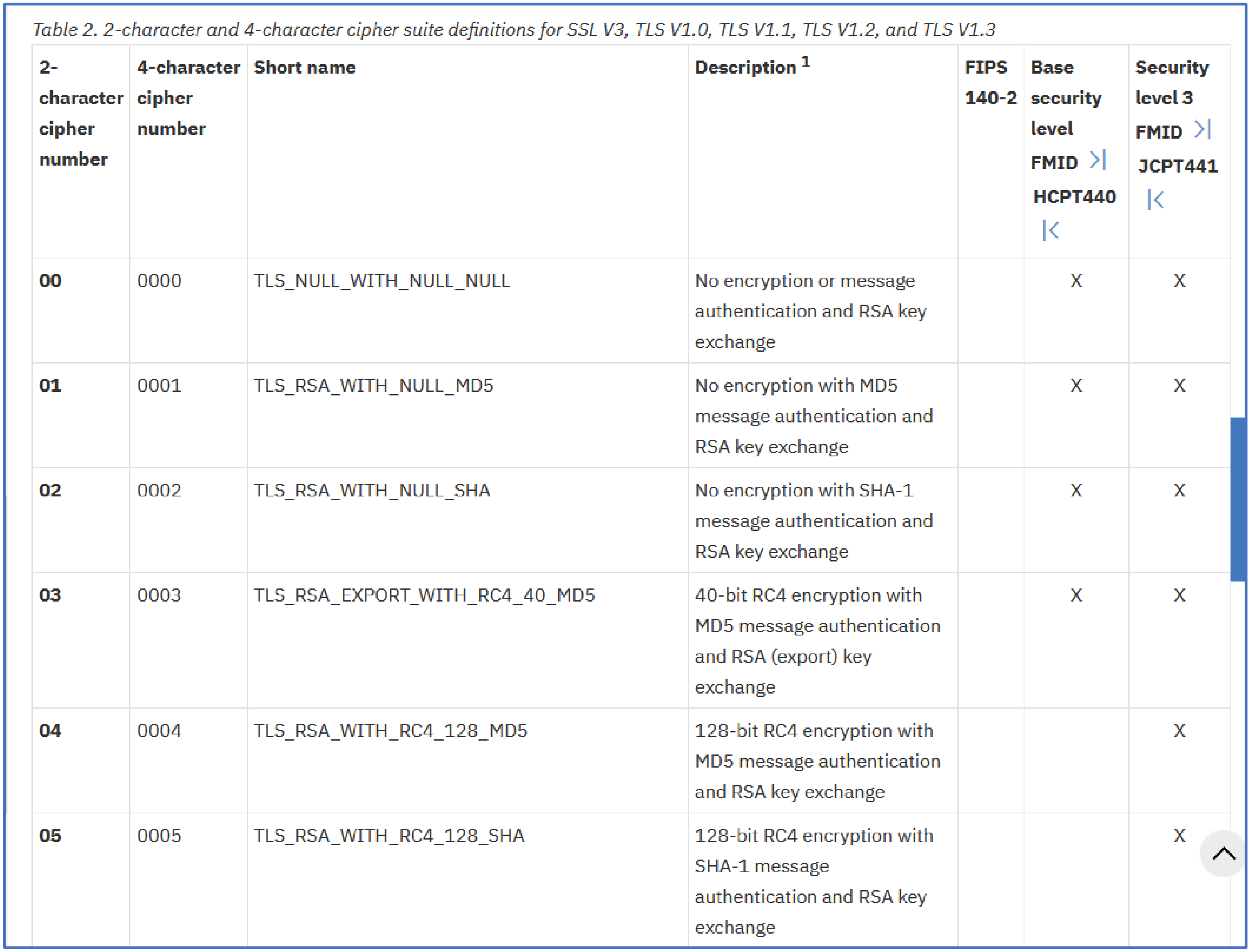


Table 2. 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3

2-character cipher number	4-character cipher number	Short name	Description [1]	FIPS 140-2	Base security level FMID >| HCPT440 |<	Security level 3 FMID >| JCPT441 |<
00	0000	TLS_NULL_WITH_NULL_NULL	No encryption or message authentication and RSA key exchange		X	X
01	0001	TLS_RSA_WITH_NULL_MD5	No encryption with MD5 message authentication and RSA key exchange		X	X
02	0002	TLS_RSA_WITH_NULL_SHA	No encryption with SHA-1 message authentication and RSA key exchange		X	X
03	0003	TLS_RSA_EXPORT_WITH_RC4_40_MD5	40-bit RC4 encryption with MD5 message authentication and RSA (export) key exchange		X	X
04	0004	TLS_RSA_WITH_RC4_128_MD5	128-bit RC4 encryption with MD5 message authentication and RSA key exchange			X
05	0005	TLS_RSA_WITH_RC4_128_SHA	128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange			X

https://www.ibm.com/support/knowledgecenter/SSLTBW_2.4.0/com.ibm.zos.v2r4.gska100/csdcwh.htm

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

TLS sessions

- When connections timeout, it is still possible to avoid the impact of full handshakes by reusing the TLS 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 TLS session to **10 minutes** (default is 8640ms)
- TLS session ids are not shared across z/OS Connect servers

Tech/Tip: SSL issues

Some SSL related messages will occur in the spool output of the server, i.e.,

- ***CWPKI0022E: SSL HANDSHAKE FAILURE: A signer with SubjectDN CN=USER3 D. Client, OU=LIBERTY, O=IBM was sent from the target host. The signer might need to be added to local trust store safkeyring:///Liberty.KeyRing, located in SSL configuration alias DefaultSSLSettings. The extended error message from the SSL handshake exception is: PKIX path building failed: com.ibm.security.cert.IBMCertPathBuilderException: unable to find valid certification path to requested target***

This message is indicating a personal certificate was presented in a TLS handshake and there was no corresponding certificate authority certificate connect to the local trust store (e.g., key ring). This can occur either for client connecting to the server or an API requester request going to an API provider.

Identify the certificate authority which signed this personal certificate and connect it to the keyring with usage CERTAUTH.

Tech/Tip: SSL issues

With a few exceptions, most of TLS errors may require a review of a trace.
I use the *traceSpecification* as shown below:

```
<logging traceSpecification="com.ibm.ws.security.*=all:SSLChannel=all:SSL=all:zosConnectSaf=all"/>
```

This will generate a *trace.out* file in the *logs* subdirectory. This trace will provide details about the key ring and certificates involved in the handshake. There is a wealth of information about the flow between the client and server endpoints Review this trace for exceptions. The following exceptions are some of the most commonly experienced.

- ***Error occurred during a read, exception:javax.net.ssl.SSLHandshakeException: null cert chain***

This exception occurs when the server configuration set to require client certificates (clientAuthenication="true") and the client had no certificate to provide and no alternative authentication method was available.

- ***Error occurred during a read, exception:javax.net.ssl.SSLException: Received fatal alert: bad_certificate error (handshake), vc=1083934466***
Caught exception during unwrap, javax.net.ssl.SSLException: Received fatal alert: bad_certificate

This is usually caused when the client certificate presented to the server did not have a valid CA certificate for the client's personal certificate in the server's trust store key ring.

Tech/Tip: SSL issues

- ***FFDC1015I: An FFDC Incident has been created: "java.io.IOException: Failed validating certificate paths com.ibm.ws.ssl.config.WSKeyStore$1 do_getKeyStore" at ffdc_19.12.04_20.51.47.0.log***

This is can occur when the CA certificate used to sign the server's personal certificate was not connected to the server's local trust store (key ring on z/OS).

- ***java.io.IOException: IOException invoking https://132.25.33.351:9443/employees/John?validated=true: HTTPS hostname wrong: should be <132.25.33.351>***

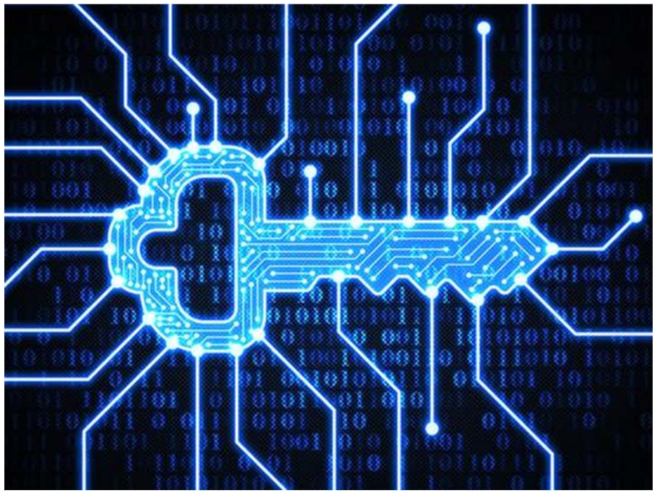
In this situation the endpoint for the outbound API request was configured to use an IP address rather than a hostname. This should not be an issue unless an exchange of digital certificates is required.
The trace showed that during the handshake process the outbound API provider server's certificate had a common name (CN) which specified the hostname of the TCPIP stack where the API resided. This hostname was not known (e.g., DNS-resolvable) on the TCPIP stack where the z/OS Connect server was executing. This meant that communications back to the API requester's TCPIP stack based on the hostname was not possible which caused the IO exception. The best solution would be to use the host name in the server.xml configuration rather than the IP address and either add an entry to the local TCPIP stack's hostname (e.g., hosts) file for the IP address and hostname or add an entry to the DNS servers used by this TCPIP stack.

General security terms or considerations

- Identifying who or what is requesting access (**Authentication)**
  - Basic Authentication
  - Mutual Authentication using TLS
  - Third Party Tokens

- Ensuring that the message has not been altered in transit (**Data Integrity)** and ensuring the confidentiality of the message in transit (**Encryption)**
  - TLS (encrypting messages and generating/sending a digital signature)

- Controlling access (**Authorization)**
  - Is the authenticated identity authorized to access to z/OS Connect
  - Is the authenticated identity authorized to access a specific API, Services, etc.

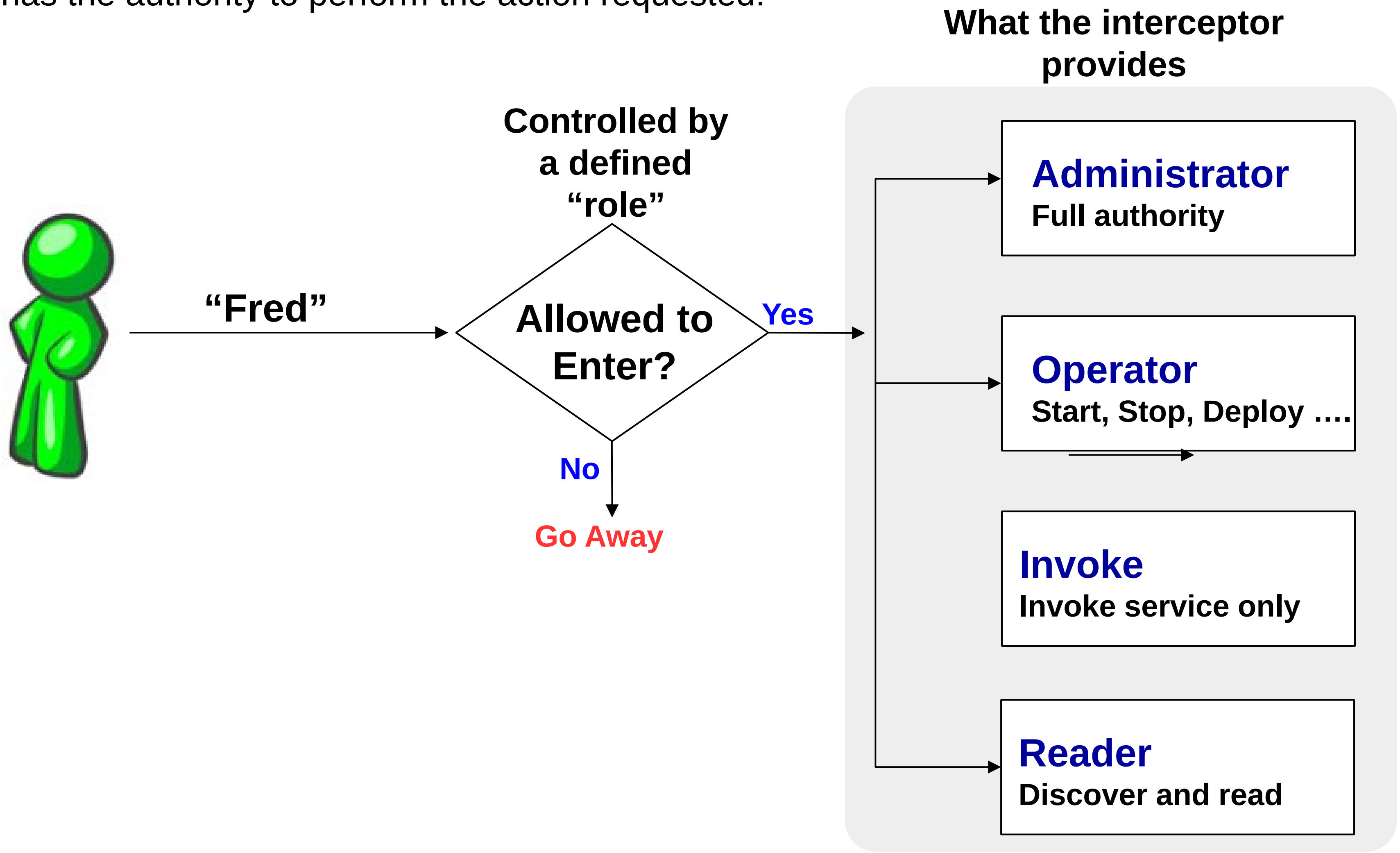


Authorization

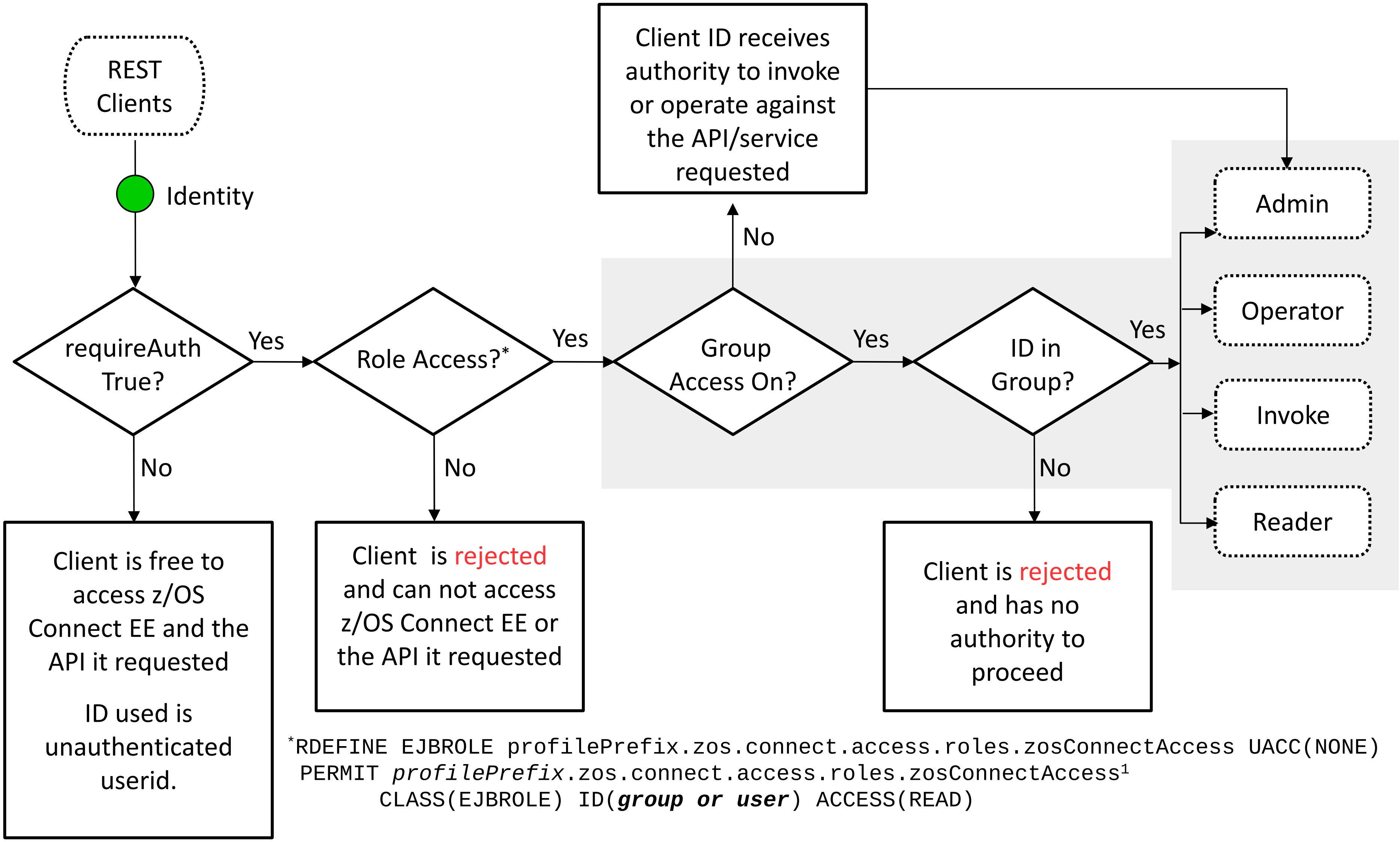
Once we have an identity, then what?

Authorization interceptor

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:



Security flow with z/OS Connect EE


REST Clients
Identity
requireAuth True?
Yes
No
Client is free to access z/OS Connect EE and the API it requested
ID used is unauthenticated userid.
Role Access?*
Yes
No
Client is rejected and can not access z/OS Connect EE or the API it requested
Group Access On?
No
Client ID receives authority to invoke or operate against the API/service requested
Yes
ID in Group?
Yes
No
Client is rejected and has no authority to proceed
Admin
Operator
Invoke
Reader

```
*RDEFINE EJBROLE profilePrefix.zos.connect.access.roles.zosConnectAccess UACC(NONE)
 PERMIT profilePrefix.zos.connect.access.roles.zosConnectAccess1
        CLASS(EJBROLE) ID(group or user) ACCESS(READ)
```

**where profilePrefix is value of the profilePrefix attribute in the safCredentials configuration element*
[1]as specified in the web.xml file

Interceptor - server XML example

```
<zosconnect_zosConnectManager
         globalInterceptorsRef="interceptorList_g"
         globalAdminGroup="GMADMIN"
         globalOperationsGroup="GMOPERS"
         globalInvokeGroup="GMINVOKE"
         globalReaderGroup="GMREADR"/>

<zosconnect_authorizationInterceptor id="auth"/>
<zosconnect_auditInterceptor id="audit"/>
<zosconnect_zosConnectInterceptors id="interceptorList_g"
       interceptorRef="auth"/>
<zosconnect_zosConnectInterceptors id="interceptorList_a"
       interceptorRef="auth,audit"/>

<zosconnect_zosConnectAPIs>
   <zosConnectAPI name="catalog"
       runGlobalInterceptorsRef="true"
       adminGroup="aapigrp1,aapigrp2"
       operationsGroup="oapigrp1,oapigrp2"
       invokeGroup="iapigrp1,oapigrp2"
       readerGroup="rapigrp1,rapigrp2"/>
</zosconnect_zosConnectAPIs>

<zosconnect_apiRequesters>
   <apiRequester name="cscvincapi_1.0.0"
       runGlobalInterceptorsRef="false"
       interceptorsRef="interceptorList_a"
       adminGroup="aaprgrp1,aaprgrp2"
       operationsGroup="oaprgrp1,oaprgrp2"
       invokeGroup="iaprgrp1,oaprgrp2"
       readerGroup="raprgrp1,raprgrp2"/>
</zosconnect_apiRequesters>

<zosconnect_services>
  <service id="selectByEmployee" name="selectEmployee"
       runGlobalInterceptorsRef="false"
       interceptorsRef="interceptorList_a"
       adminGroup="asrvgrp1,asrvgrp2"
       operationsGroup="osrvgrp1,osrvgrp2"
       invokeGroup="isrvgrp1,isrvgrp2"
       readerGroup="rsrvrgrp1,rsrvgrp2"/>
</zosconnect_services>
```

```
ADDGROUP GMADMIN OMVS(AUTOGID)
ADDGROUP GMINVOKE OMVS(AUTOGID)
CONNECT FRED GROUP(GMADMIN)
CONNECT USER1 GROUP(GMINVOKE)
```

Global interceptor list – authorization only

Alternative interceptor list – authorization and audit

Avoid duplication of interceptors

Note that these are z/OS Connect configuration elements. Documented in the z/OS Connect KC

```
RDEFINE FACILITY BPX.NEXT.USER APPLDATA('2001/201')
```

Server XML example TLS/AUTH interceptor Example

```
 <zosconnect_zosConnectManager
           requireAuth="true"
           requireSecure="true"
           globalInterceptorsRef="interceptorList_g"
           globalAdminGroup="GMADMIN"
           globalOperationsGroup="GMOPERS"
           globalInvokeGroup="GMINVOKE"
           globalReaderGroup="GMREADR"/>

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

  <zosconnect_apiRequesters>
     <apiRequester name="cscvincapi_1.0.0"
         requireSecure="false"
         adminGroup="aaprgrp1,aaprgrp2"
         operationsGroup="oaprgrp1,oaprgrp2"
         invokeGroup="iaprgrp1,oaprgrp2"
         readerGroup="raprgrp1,raprgrp2"/>
  </zosconnect_apiRequesters>
```

Global TLS security and authentication are enabled.

TLS security is disabled for this API requester archive artifact.

This configuration would allow a MVS batch job to authenticate to z/OS Connect and use HTTP for the protocol. Only authorization identities which are members of groups identified as administrators or invokers would be authorized to invoke this API requester.

```
F BAQSTRT,ZCON,CLEARSAFCACHE
```

Flowing identities to back-end z/OS systems

Basic authentication - Identity and Password

Server XML Configuration elements where basic authentication can be provided.

```
<connectionFactory id="imsTM"> containerAuthDataRef="credentials">
<authData id="credentials" user= "identity" password= "password"/>

<connectionFactory id="imsDB">
<properties.imsudbJLocal  databaseName="DFSIVPA"   user="identity" password="password"/>
</connectionFactory>

<zosconnect_cicsIpicConnection id="CICS"  authDataRef="credentials"/>
<zosconnect_authData id="credentials" user= "identity" password= "password"/>

<zosconnect_zosConnectServiceRestClientConnection id="Db2" basicAuthRef=“db2Auth"/>
<zosconnect_zosConnectServiceRestClientBasicAuth id="db2Auth"
    userName="identity"  password="password"/>

<jmsQueueConnectionFactory jndiName=“MQ“>
            <properties.wasJms userName=“identity" password=“password"   />
</jmsQueueConnectionFactory>
```

The value of the password can be encoded in the server XML configuration file. Using the ***securityUtility*** shipped with WebSphere Liberty Profile.

Using securityUtility to encrypt passwords

Best practice : use encryption for passwords instead of base64 encoding

- **securityUtility** – located in <wlp_install_dir>/wlp/bin
  - Usage: securityUtility {encode|createSSLCertificate|createLTPAKeys|help} [options]
  - For encryption, use encode --key=encryption_key
    - Specifies the key to be used when encoding using AES encryption. This string is hashed to produce an encryption key that is used to encrypt and decrypt the password. The key can be provided to the server by defining the variable **wlp.password.encryption.key** whose value is the key. If this option is not provided, a default key is used.

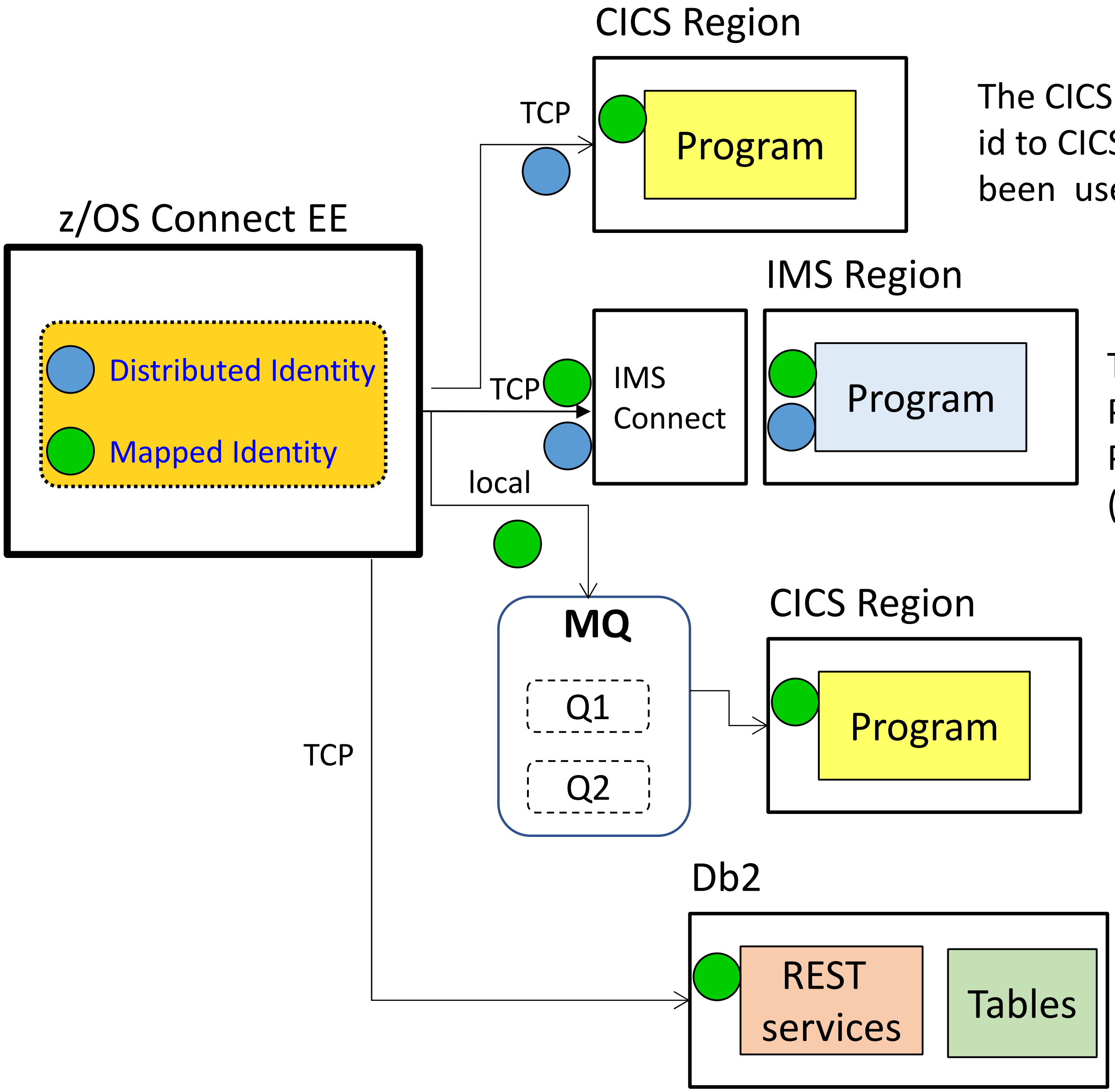
```
./securityUtility encode --encoding=aes --key=myKey passW0rd
{aes}AHO0aXdiVD96u4oMRhoKeYH3U7aDqtFXTuHFBsO98Wlb
```

  - Also supports 1-way hash encoding – for passwords in server.xml with basicRegistry
    - For hash, use encode –encoding=hash

```
./securityUtility encode –-encoding=hash XXXXXXXX
{hash}ATAAAAIcqTmHn5qZahAAAAIMjzy+hP8YFaIO6LiCreVe4etRLUS9a25eVuYtx6WKiv
```

https://www.ibm.com/support/knowledgecenter/en/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/rwlp_command_securityutil.html

Flowing an identity to a back end subsystem



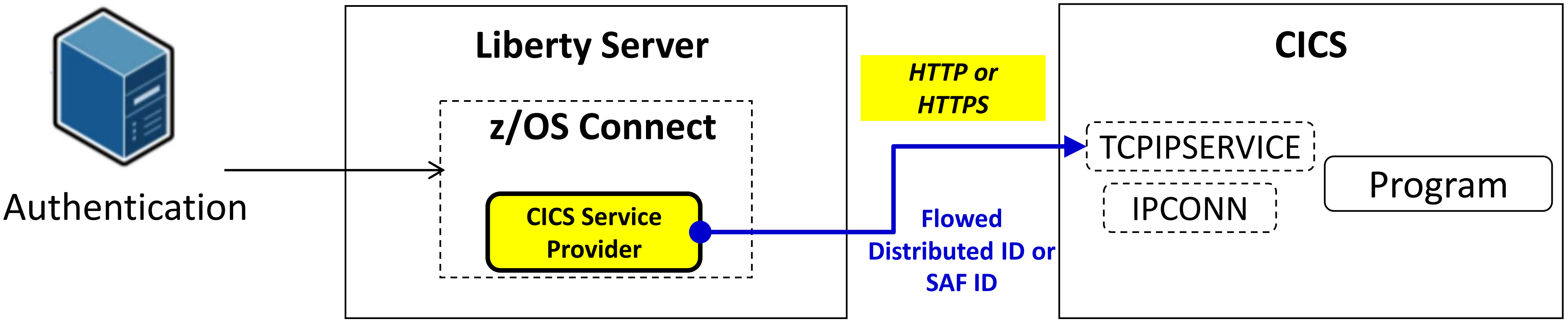
The CICS SP propagates the distributed id to CICS (or sends the SAF id if it has been used for authentication)

The IMS SP can request a PassTicket and send the PassTicket to IMS Connect (available in V3.0.33)

The MQ SP asserts the mapped id to MQ

The REST client SP can request a PassTicket and send with mapped ID to Db2 (available in V3.0.15)

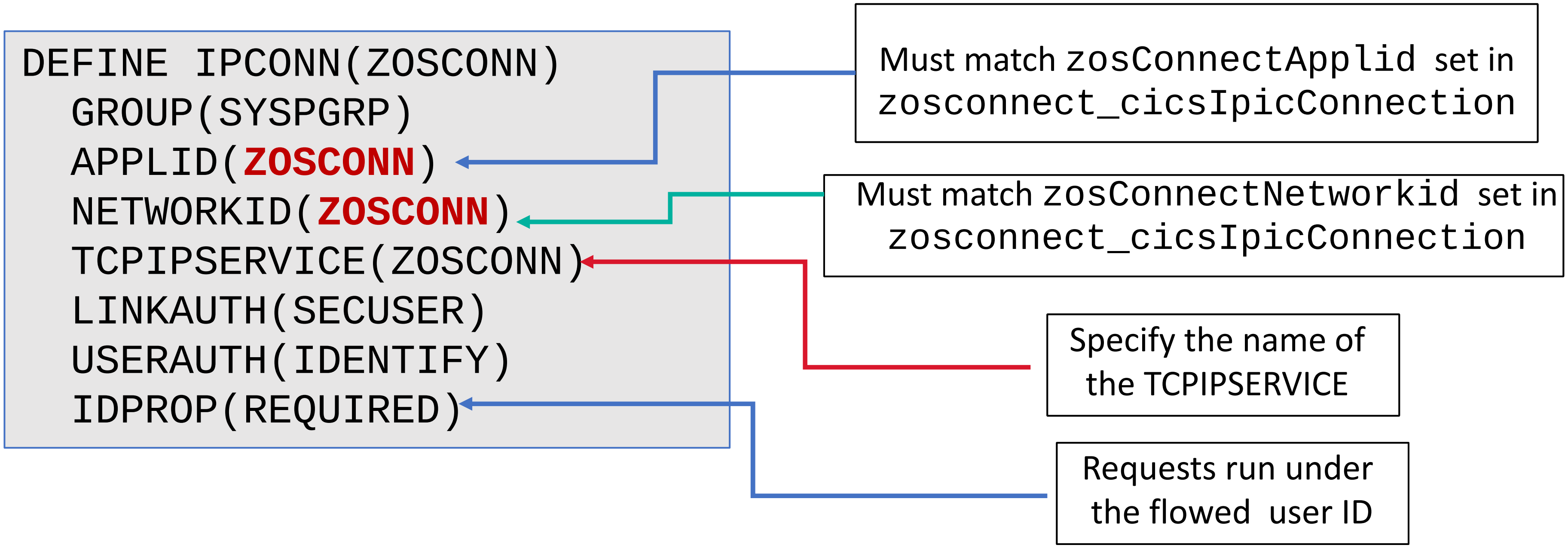
Flowing a user ID with CICS service provider



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 TLS (JSSE) connection that is configured with client authentication.

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

CICS IPCONN



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

Must match `zosConnectApplid` set in `zosconnect_cicsIpicConnection`

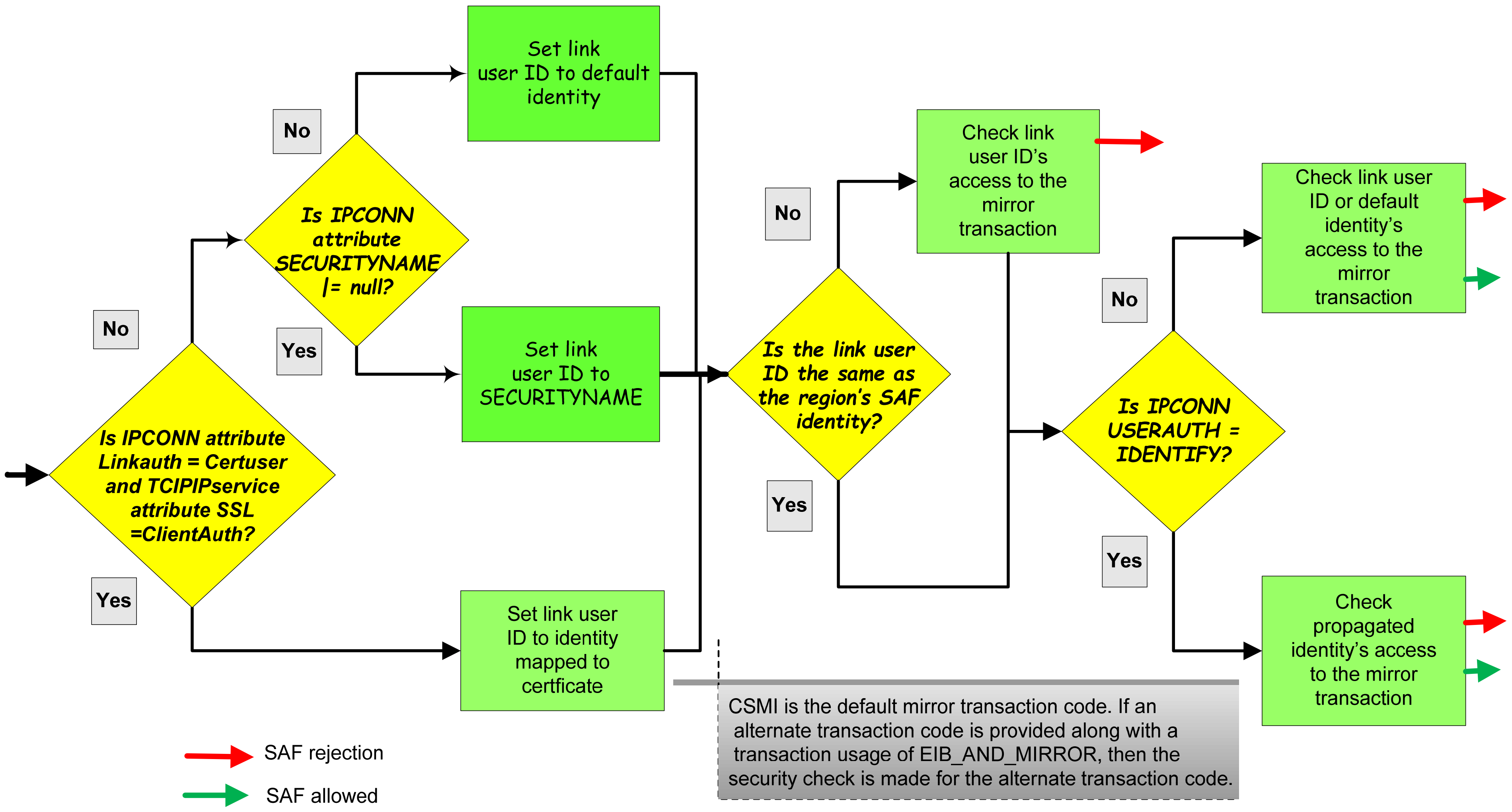
Must match `zosConnectNetworkid` set in `zosconnect_cicsIpicConnection`

Specify the name of the TCPIPSERVICE

Requests run under the flowed user ID

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

Tech/Tip: CICS IPIC Security



Is IPCONN attribute Linkauth = Certuser and TCIPIPservice attribute SSL =ClientAuth?

No

Yes

Is IPCONN attribute SECURITYNAME |= null?

No

Yes

Set link user ID to default identity

Set link user ID to SECURITYNAME

Set link user ID to identity mapped to certficate

Is the link user ID the same as the region's SAF identity?

No

Yes

Check link user ID's access to the mirror transaction

Is IPCONN USERAUTH = IDENTIFY?

No

Yes

Check link user ID or default identity's access to the mirror transaction

Check propagated identity's access to the mirror transaction

🟥 SAF rejection

🟩 SAF allowed

CSMI is the default mirror transaction code. If an alternate transaction code is provided along with a transaction usage of EIB_AND_MIRROR, then the security check is made for the alternate transaction code.

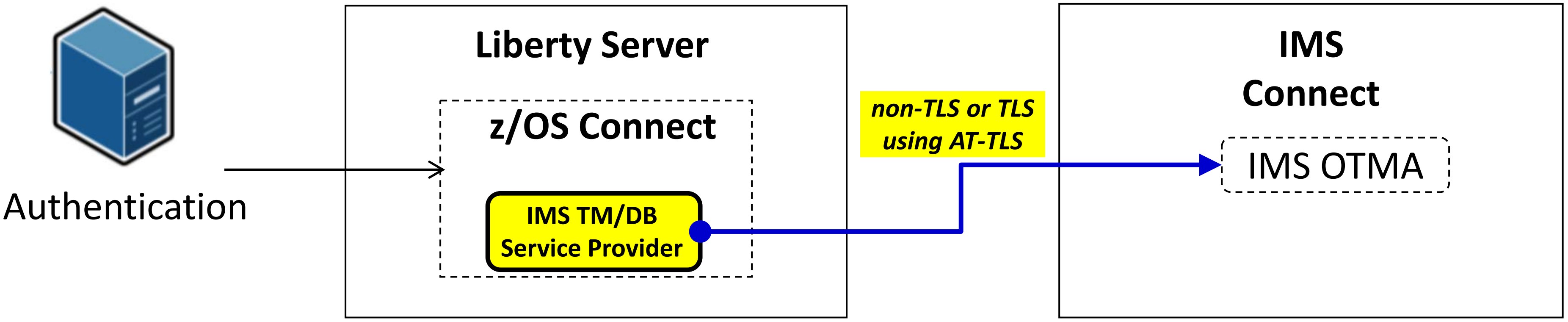
PassTickets and IMS

- ❑ Basic authentication to IMS Connect using a PassTicket depends on the APPL parameters configured in IMS Connect.

```
HWS=(ID=IMS15HWS,XIBAREA=100,RACF=Y,RRS=Y)
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,TIMEOUT=5000)
DATASTORE=(GROUP=OTMAGRP,ID=IVP1,MEMBER=HWSMEM,DRU=HWSYDRU0,
TMEMBER=OTMAMEM,APPL=IMSTMAPL)
ODACCESS=(ODBMAUTOCONN=Y,IMSPLEX=(MEMBER=IMS15HWS,TMEMBER=PLEX1),
     DRDAPORT=(ID=5555,PORTTMOT=6000),ODBMTMOT=6000,APPL=IMSDBAPL)
```

```
RDEFINE PTKTDATA IMSTMAPL SSIGNON(0123456789ABCDEF)) APPLDATA(‘NO REPLAY PROTECTION’) UACC(NONE)
RDEFINE PTKTDATA IRRPTAUTH.IMSTMAPL.* UACC(NONE)
PERMIT IRRPTAUTH.IMSTMAPL ID(LIBSERV) CLASS(PTKTDATA) ACCESS(UPDATE)

RDEFINE PTKTDATA IMSDBAPL SSIGNON(0123456789ABCDEF)) APPLDATA(‘NO REPLAY PROTECTION’) UACC(NONE)
RDEFINE PTKTDATA IRRPTAUTH.IMSDBAPL.* UACC(NONE)
PERMIT IRRPTAUTH.IMSDBAPL ID(LIBSERV) CLASS(PTKTDATA) ACCESS(UPDATE)
```

Flowing an identity to IMS Connect (TM)



```
HWS=(ID=IMS15HWS,XIBAREA=100,RACF=Y,RRS=Y)
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,TIMEOUT=5000)
DATASTORE=(GROUP=OTMAGRP,ID=IVP1,MEMBER=HWSMEM,DRU=HWSYDRU0,
TMEMBER=OTMAMEM,APPL=IMSTMAPL)
```

Authentication options:

1. **User ID / password**
2. **PassTicket support**

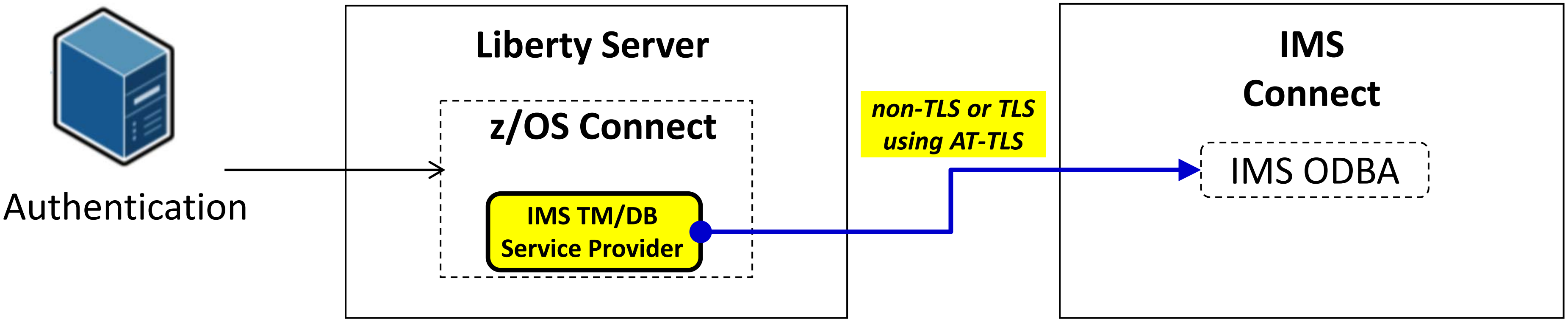
```
<connectionFactory containerAuthDataRef="Connection1_Auth" id="IVP1">
<properties.gmoa hostName="wg31.washington.ibm.com" portNumber="4000"/>
</connectionFactory>
<authData id="Connection1_Auth"  user="USER1“password="{xor}GhIPExAGDwg="/>
```

Specify a user identity and password to be used in the request to IMS Connect

```
<connectionFactory containerAuthDataRef="Connection1_Auth" id="IVP1">
<properties.gmoa hostName="wg31.washington.ibm.com" portNumber="4000
applicationName="IMSTMAPL"/>
</connectionFactory>
```

Request a PassTicket
And use it in the request
to IMS Connect

Flowing an identity to IMS Connect (DB)



```
HWS=(ID=IMS15HWS,XIBAREA=100,RACF=Y,RRS=Y)
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,TIMEOUT=5000)
ODACCESS=(ODBMAUTOCONN=Y,IMSPLEX=(MEMBER=IMS15HWS,TMEMBER=PLEX1),
   DRDAPORT=(ID=5555,PORTTMOT=6000),ODBMTMOT=6000,APPL=IMSDBAPL)
```

Authentication options:

1. **User ID / password**
2. **PassTicket support**

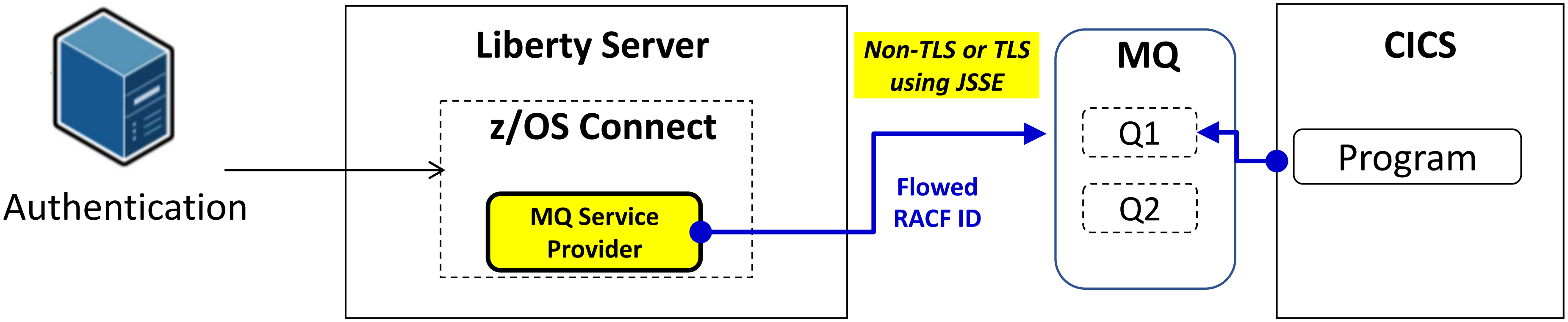
```
<connectionFactory id="DFSIVPAConn">  <properties.imsudbJLocal
databaseName="DFSIVPA"  datastoreName="IVP1"  portNumber="5555"
driverType="4" datastoreServer="wg31.washington.ibm.com"  flattenTables="True"
user="USER1 "  password="USER1" />
</connectionFactory>
```

Specify a user identity and password to be used in the request to IMS Connect

```
<connectionFactory id="DFSIVPAConn">    <properties.imsudbJLocal
databaseName="DFSIVPA" datastoreName="IVP1" portNumber="5555"
datastoreServer="wg31.washington.ibm.com«  driverType="4" flattenTables="True"
applicationName="IMSDBAPL"  "/>
</connectionFactory>
```

Request a PassTicket
And use it in the request
to IMS Connect

Flowing a user ID with MQ service provider

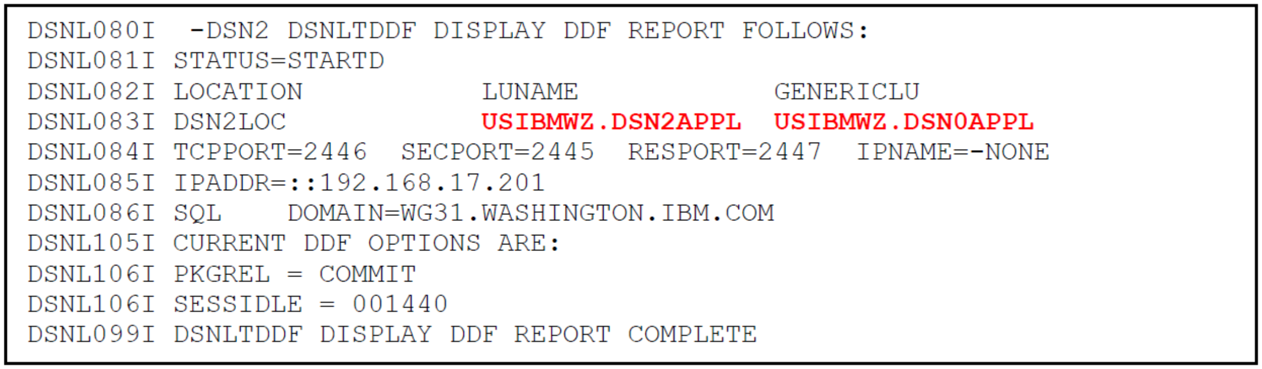


Set **`useCallerPrincipal=true`** to flow the authenticated RACF user ID

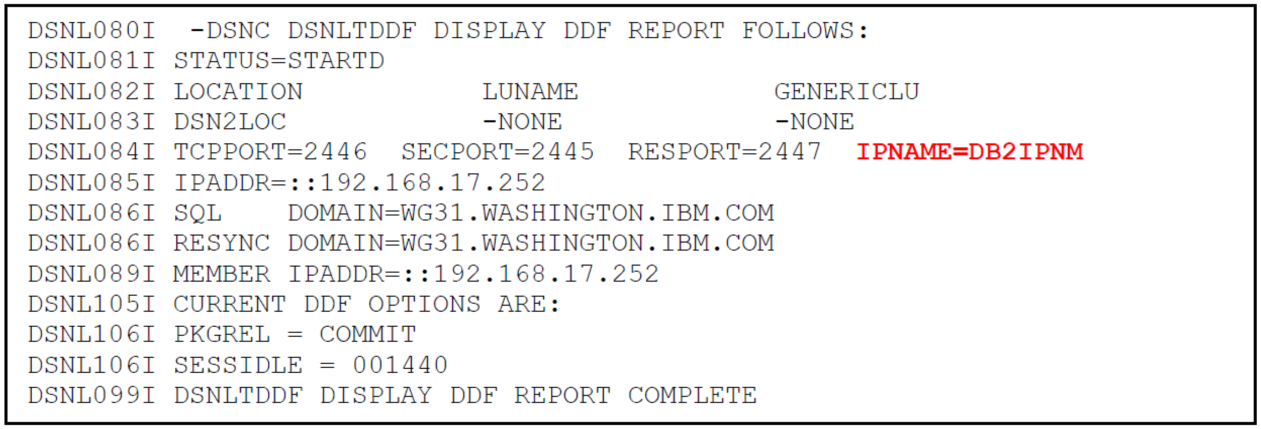
```
<zosconnect_services>
    <service name="mqPut">
        <property name="destination" value="jms/default"/>
        <property name="useCallerPrincipal" value="true"/>
    </service>
</zosconnect_services>
```

PassTickets and Db2

❑ Basic authentication Db2 using a PassTicket depends on the Db2 configuration.



```
DSNL080I  -DSN2 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I DSN2LOC            USIBMWZ.DSN2APPL  USIBMWZ.DSN0APPL
DSNL084I TCPPORT=2446  SECPORT=2445  RESPORT=2447  IPNAME=-NONE
DSNL085I IPADDR=::192.168.17.201
DSNL086I SQL    DOMAIN=WG31.WASHINGTON.IBM.COM
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL106I SESSIDLE = 001440
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```



```
DSNL080I  -DSNC DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I DSN2LOC            -NONE             -NONE
DSNL084I TCPPORT=2446  SECPORT=2445  RESPORT=2447  IPNAME=DB2IPNM
DSNL085I IPADDR=::192.168.17.252
DSNL086I SQL    DOMAIN=WG31.WASHINGTON.IBM.COM
DSNL086I RESYNC DOMAIN=WG31.WASHINGTON.IBM.COM
DSNL089I MEMBER IPADDR=::192.168.17.252
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL106I SESSIDLE = 001440
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

Which value should be used for applName is determine for use in RACF resources is determine as shown below.

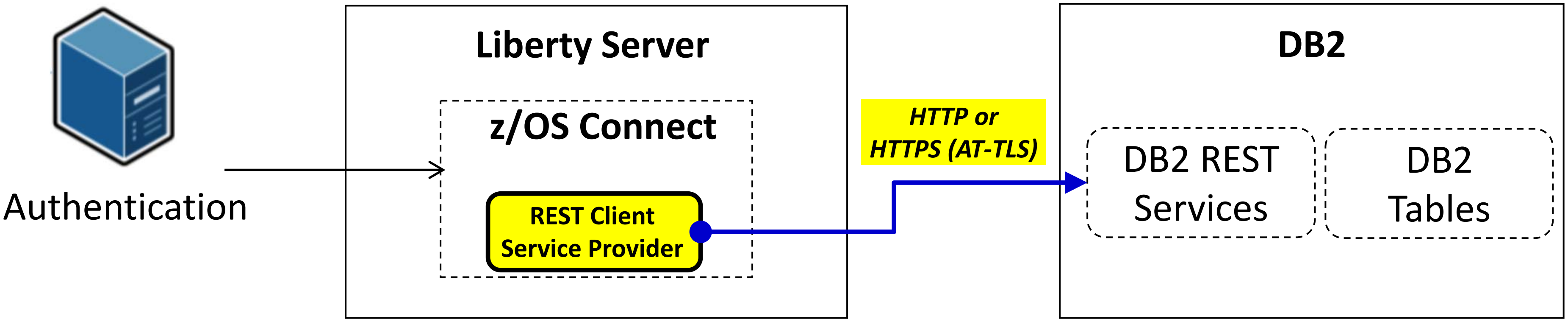
□ If GENERICLU is defined, use the second part of GENERICLU for *applName*, e.g., ***DSN0APPL***

□ If GENERICLU is not defined, use the second part of LUNAME for *applName*, e.g., ***DSN2APPL***

□ If neither GENERICLU or LUNAME is defined, use the value of the IPNAME for *applName,* e.g., ***DB2IPNM***

```
RDEFINE PTKTDATA DSN2APPL SSIGNON(0123456789ABCDEF)) APPLDATA(‘NO REPLAY PROTECTION’) UACC(NONE)
RDEFINE PTKTDATA IRRPTAUTH.DSN2APPL.* UACC(NONE)
PERMIT IRRPTAUTH.DSN2APPL ID(LIBSERV) CLASS(PTKTDATA) ACCESS(UPDATE)
```

Flowing the identity for the REST client SP (Db2)



```
<zosconnect_zosConnectServiceRestClientConnection id="Db2Conn"
    host="wg31.washington.ibm.com"
    port="2446"
    basicAuthRef="dsn2Auth" />
<zosconnect_zosConnectServiceRestClientBasicAuth id="dsn2Auth"
    userName="USER1"
    password="USER1"/>
```

Authentication options:

1. **User ID / password**
2. **TLS Client Certificate (JSSE)**
3. **PassTicket support**

Specify a user identity and password to be used in the HTTP header with the Db2 REST Service

```
<zosconnect_zosConnectServiceRestClientConnection id="Db2Conn"
    host="wg31.washington.ibm.com"
    port="2446"
    basicAuthRef="dsn2Auth" />

<zosconnect_zosConnectServiceRestClientBasicAuth id="dsn2Auth"
    applName="DSN2APPL"/>
```

z/OS Connect requests a PassTicket from RACF

Tech/Tip: Db2 REST Security

- ❑ Access to Db2 REST services requires READ access to the Db2 subsystem DSNR REST resource. i.e., permit READ access to this resource to the identity in question, for example

  PERMIT DSN2.REST CLASS(DSNR) ID(USER2) ACC(READ) where DSN2 is the Db2 subsystem ID
  SETROPTS RACLIST(DSNR) REFRESH

- ❑ Db2 package access is also required. If a user is not able to display a valid Db2 REST services in the z/OS Connect Db2 services development tooling or by using a **POST** to the Db2 provided REST interface URL of http://wg31.washington.ibm.com:2446/services/DB2ServiceDiscover, then they may not have sufficient access to the package containing the service.

  For example, if service *zCEEService.selectEmployee* is defined to Db2 but not visible in the z/OS Connect tooling or if a **GET** request to URL http://wg31.washington.ibm.com:2446/services/zCEEService/selectEmployee fails with message:

```
{
  "StatusCode": 500,
  "StatusDescription": "Service zCEEService.selectEmployee discovery failed due to
  SQLCODE=-551 SQLSTATE=42501, USER2 DOES NOT HAVE THE PRIVILEGE TO PERFORM OPERATION EXECUTE
  PACKAGE ON OBJECT zCEEService.selectEmployee. Error Location:DSNLJACC:35"
}
```

  The user needs to be granted execute authority on package *zCEEService.selectEmployee* with command:

  GRANT EXECUTE ON PACKAGE "zCEEService"."selectEmployee" TO USER2 or
  GRANT EXECUTE ON PACKAGE "zCEEService"."*“ TO USER2

WOLA Security

❑ MVS Batch

```
  <zosLocalAdapters wolaGroup="ZCEESRVR"
      wolaName2="ZCEESRVR"
      wolaName3="ZCEESRVR"/>
```

```
RDEFINE CBIND BBG.WOLA.ZCEESRVR.ZCEESRVR.ZCEESRVR UACC(NONE) OWNER(SYS1)
PERMIT BBG.WOLA.ZCEESRVR.ZCEESRVR.ZCEESRVR CLASS(CBIND) ACCESS(READ) ID(USER1,START1)
SETROPTS RACLIST(CBIND) REFRESH
```

❑ Data Virtualization Manager

```
  "DEFINE ZCPATH",
    "  NAME(ZCEE)                   ",
    "  RNAME(ZCEEDVM)               ",
    "  WNAME(ZCEEDVM)               ",
    ""
```

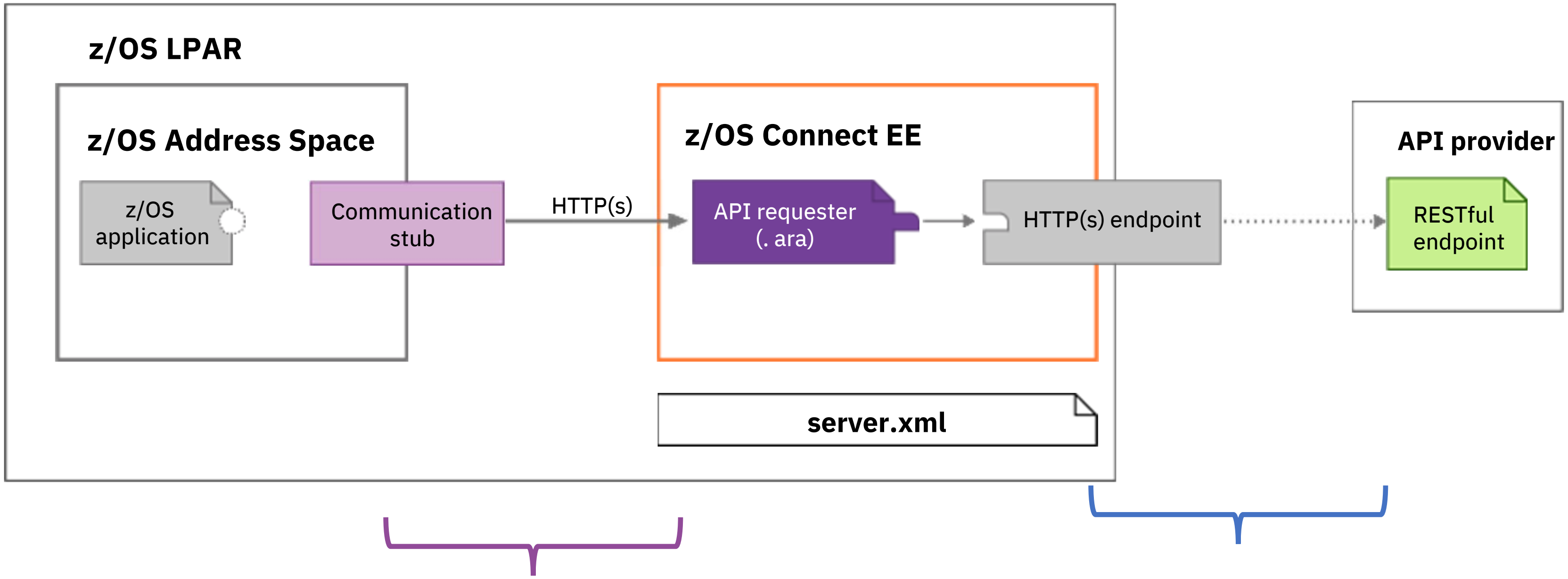
```
  <!-- Adapter Details with WOLA Group Name (ZCEEDVM) -->
       <zosLocalAdapters wolaName3="NAME3"
         wolaName2="NAME2"
         wolaGroup="ZCEEDVM"/>
```

```
RDEFINE CBIND BBG.WOLA.ZCEEDVM.** UACC(NONE)
PERMIT BBG.WOLA.ZCEEDVM.** CLASS(CBIND)  ID(LIBSERV) ACC(READ)
SETROPTS RACLIST(CBIND) REFRESH
```

z/OS Connect API Requester Security

Details

Authentication

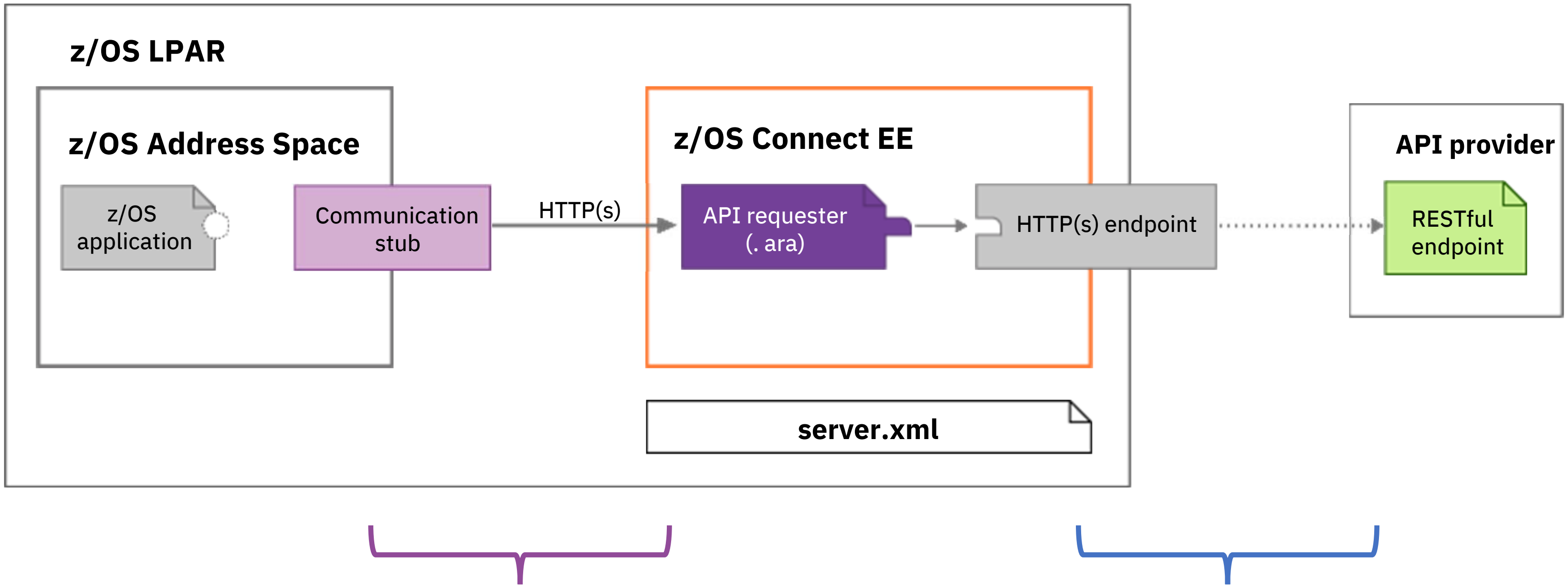


Options:

1. Basic Authentication
2. TLS Client/Server

1. Basic Authentication
2. TLS Client/Server
3. Third Party token

Encryption

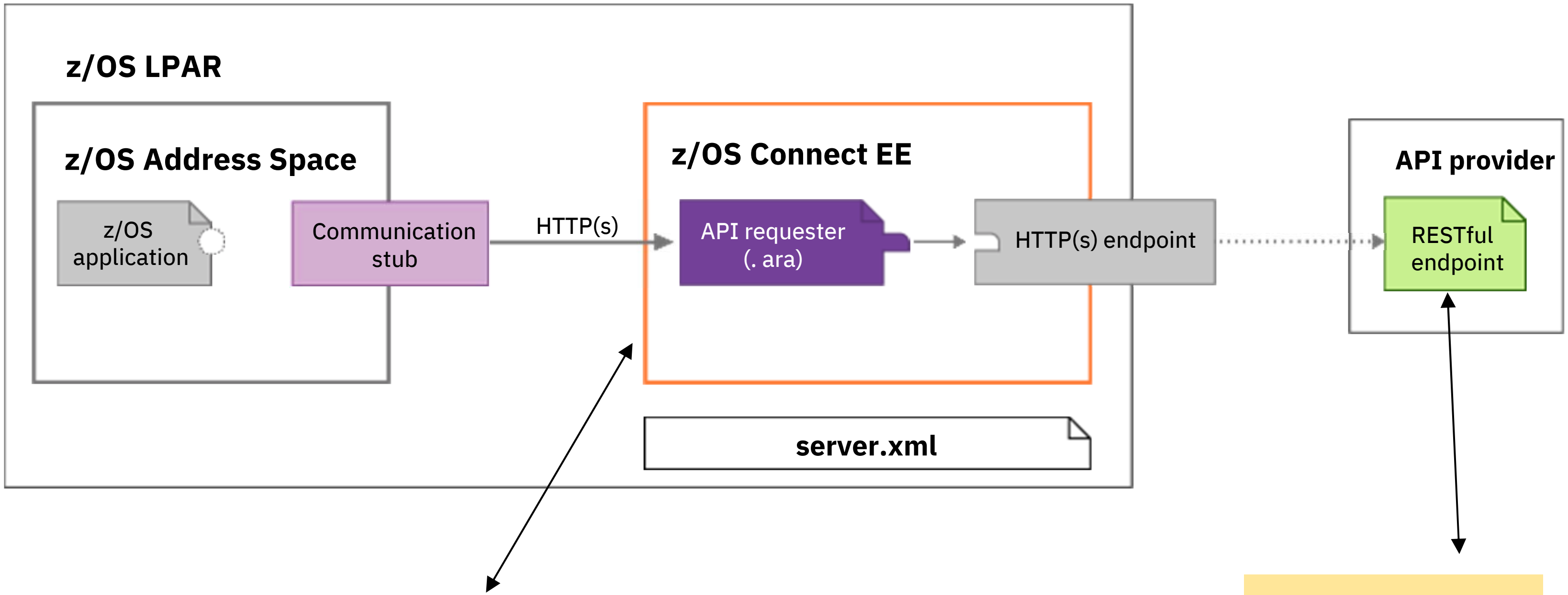


Options:

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

1. JSSE
2. AT-TLS

Authorization



Authorization interceptor options:

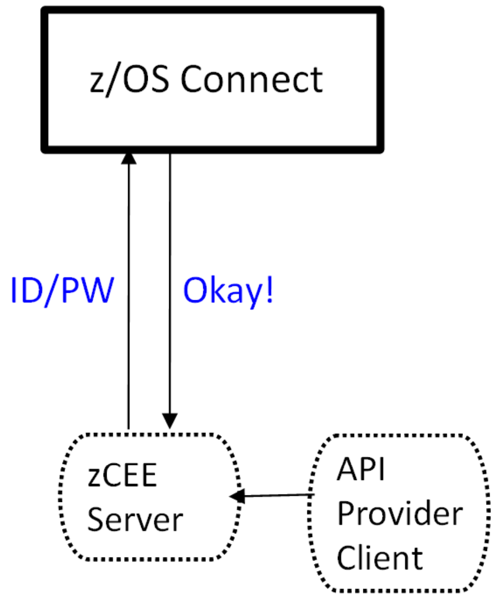
- Global level
- API requester level

OAuth or JWT access token

API Requester- API Provider Authentication

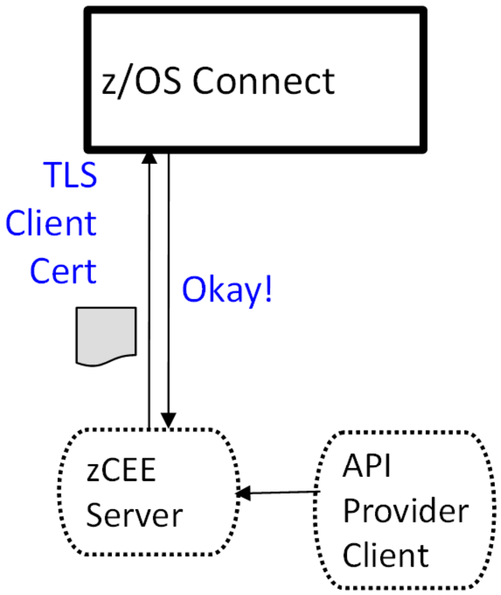
Several different ways this can be accomplished:

Basic Authentication



zCEE server supplies ID/PW or ID/PassTicket

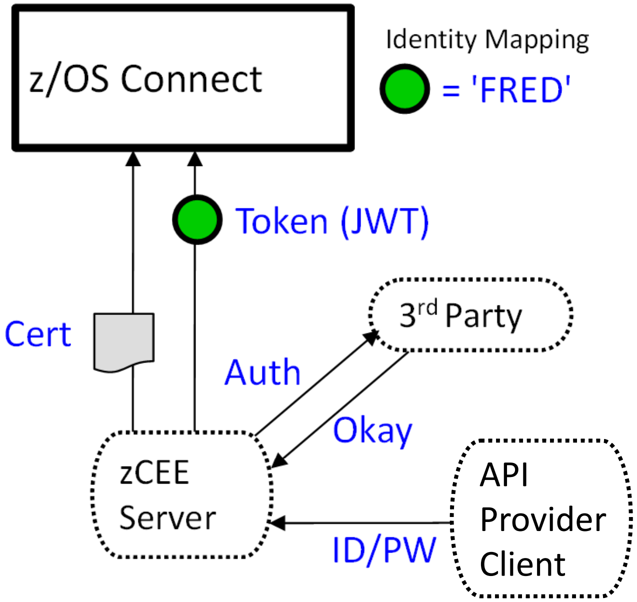
Client Certificate



Server requests a client certificate

zCEE supplies a client certificate

Third Party Authentication



zCEE Server authenticates to 3rd party server

zCEE Server receives a trusted 3rd party token

Token flows to API Provider

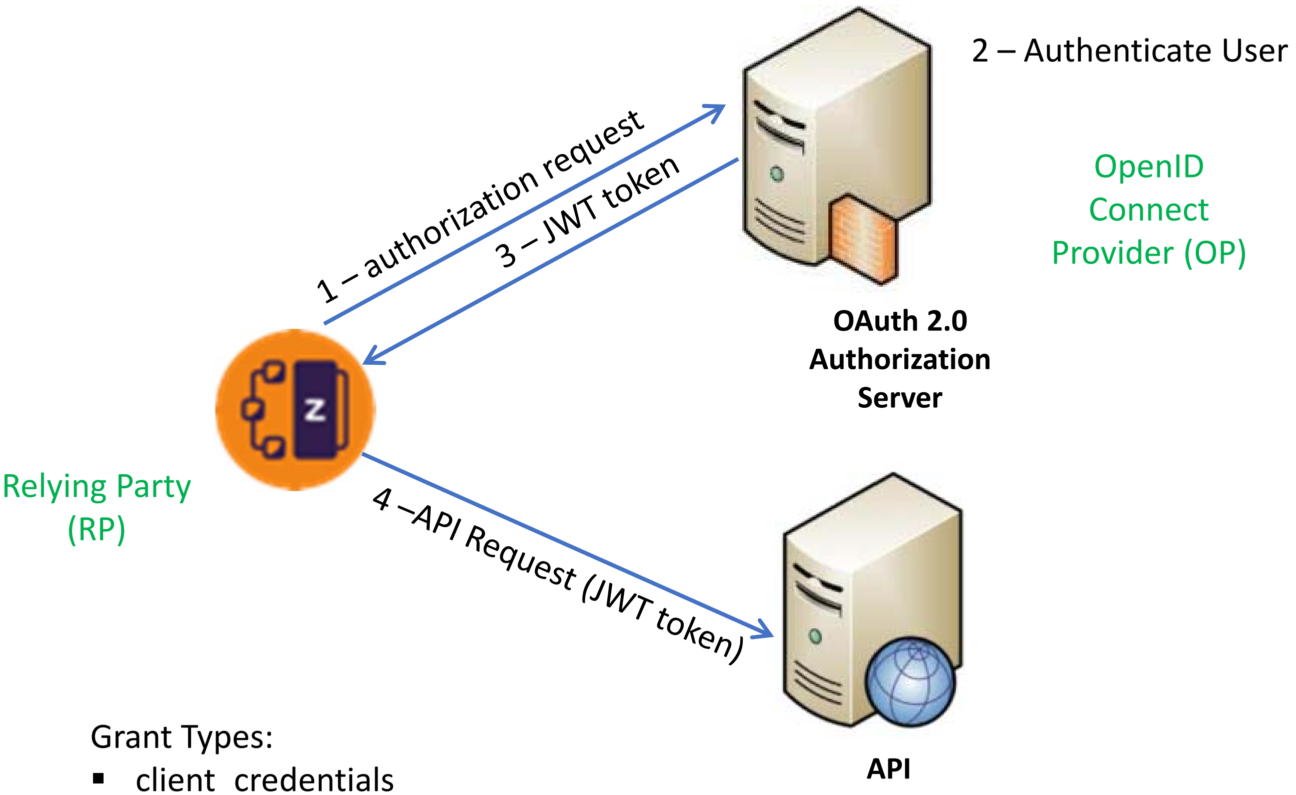
z/OS Connect API Requester - Token Support

z/OS Connect EE provides ***two*** ways of calling an API secured with a token

1. Use the OAuth 2.0 support when the request is part of an OAuth 2.0 flow. With OAUTH configured, the token can be an opaque token or a JWT token.

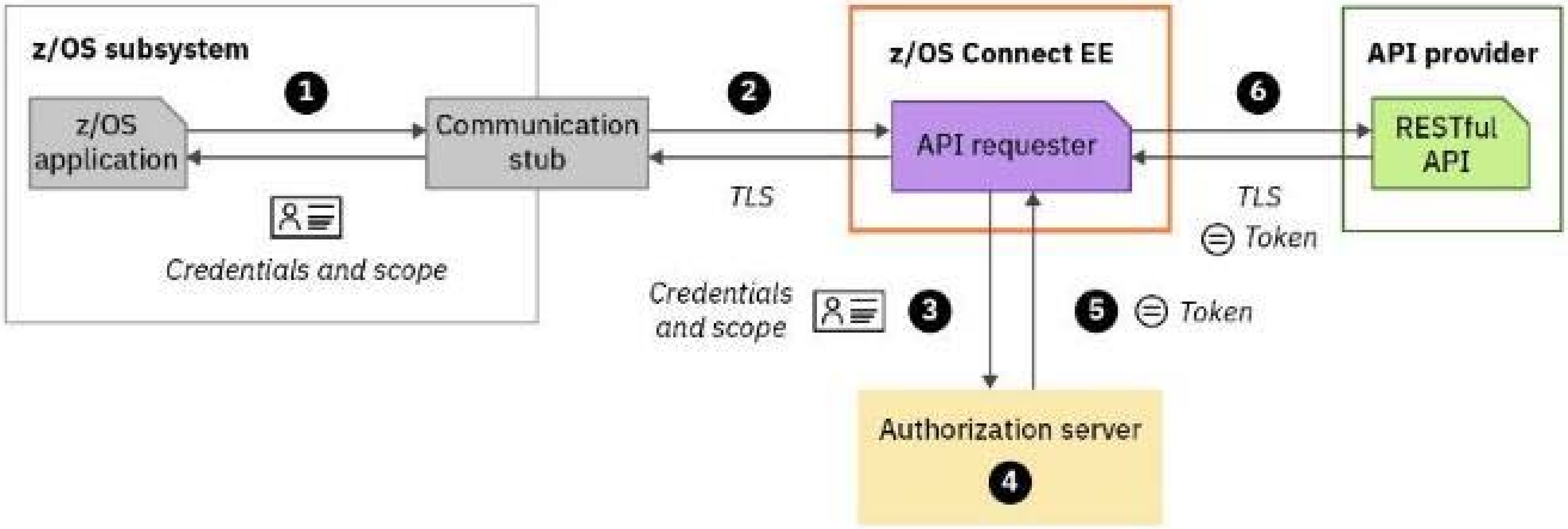
2. In a non-OAuth 2.0 scenario, a JWT token is used in a custom flow, for example: When you need to specify the HTTP verb that is used in the request to the authentication server.
   - When you need to specify how the JWT is returned from the authentication server (for example, in an HTTP header or in a custom field in a JSON response message).
   - When you need to use a custom header name for sending the JWT to the request endpoint.

z/OS Connect OAuth Flow for API requester



Grant Types:
- client_credentials
- password

Calling an API with OAuth 2.0 support



z/OS Connect OpenID Connect terms

Resource owner - An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end user. *In a z/OS Connect EE API requester scenario, the resource owner might be the* ***user*** *of the CICS, IMS, or z/OS application.*

Resource server - The server that hosts the protected resources and accepts and responds to protected resource requests by using access tokens. *In a z/OS Connect EE API requester scenario, the resource server is the* ***request endpoint*** *for the remote RESTful API.*

Client - An application that makes protected resource requests on behalf of the resource owner and with its authorization. *In a z/OS Connect EE API requester scenario, the client is a combination of the CICS, IMS, or z/OS application* ***and*** *the z/OS Connect EE server that calls the RESTful API on behalf of the CICS, IMS, or z/OS application.*

Authorization server - The server that issues access tokens to the client after authenticating the resource owner and obtaining authorization. *In a z/OS Connect EE API requester scenario, the authorization server is called by the z/OS Connect EE server to retrieve an access token.*

OAuth Grant Types Supported by z/OS Connect

***client_credentials* -** *the identity associated with the combination of the CICS, IMS, or z/OS application, and the z/OS Connect EE server that calls the RESTful API on behalf of the CICS, IMS, or z/OS application When this grant type is used, the z/OS Connect EE server sends the client credentials and the access scope to the authorization server.*

```
<zosconnect_oAuthConfig id="myoAuthConfig"
      grantType="client_credentials"
      authServerRef="myoAuthServer"/>
```

password* -** *The identity of the user of the CICS, IMS, or z/OS application, or it might be another entity.** *When this grant type is used, the z/OS Connect EE server sends the resource owner's credentials, the client credentials, and the access scope to the authorization server.*

```
<zosconnect_oAuthConfig id="myoAuthConfig"
      grantType=“password"
      authServerRef="myoAuthServer"/>
```

Configuring OAuth support – BAQRINFO copy book

```
05 BAQ-OAUTH.
  07 BAQ-OAUTH-USERNAME           PIC X(256).
  07 BAQ-OAUTH-USERNAME-LEN       PIC S9(9) COMP-5 SYNC VALUE 0.
  07 BAQ-OAUTH-PASSWORD           PIC X(256).
  07 BAQ-OAUTH-PASSWORD-LEN       PIC S9(9) COMP-5 SYNC VALUE 0.
  07 BAQ-OAUTH-CLIENTID           PIC X(256).
  07 BAQ-OAUTH-CLIENTID-LEN       PIC S9(9) COMP-5 SYNC VALUE 0.
  07 BAQ-OAUTH-CLIENT-SECRET      PIC X(256).
  07 BAQ-OAUTH-CLIENT-SECRET-LEN  PIC S9(9) COMP-5 SYNC VALUE 0.
  07 BAQ-OAUTH-SCOPE-PTR          USAGE POINTER.
  07 BAQ-OAUTH-SCOPE-LEN          PIC S9(9) COMP-5 SYNC.
```

Grant Type: *client_credentials* - the identity associated with the combination of the CICS, IMS, or z/OS application, and the z/OS Connect EE server that calls the RESTful API on behalf of the CICS, IMS, or z/OS application

Grant Type: *password* - The identity of the user provided by the CICS, IMS, or z/OS application, or it might be another entity. Client_credentials can be supplied by the program or in the server XML configuration.

Scope is always required.

OAuth 2.0 specification entity	password	client_credentials	Where Set
Client ID	required	Required	server.xml or by application
Client Secret	optional	Required	server.xml or by application
Username	required	N/A	by application
Password	required	N/A	by application

Configuring OAuth support – z/OS Connect API Requester

```
<zosconnect_endpointConnection id="cscvincAPI"
       host="http://wg31.washington.ibm.com" port="9080"
       authenticationConfigRef="myoAuthConfig"/>

<zosconnect_oAuthConfig id="myoAuthConfig"
      grantType="client_credentials|password"
      authServerRef="myoAuthServer"/>

<zosconnect_authorizationServer id="myoAuthServer"
      tokenEndpoint=https://wg31.washington.ibm.com:59443/oidc/endpoint/OP/token[1]
      basicAuthRef="tokenCredential"  [2]
      sslCertsRef="OutboundSSLSettings" />

<zosconnect_authData id="tokenCredential" [2]
  user="zCEEClient" password="secret"/>
```

```
openidConnectProvider id="OP“
   signatureAlgorithm="RS256“
   keyStoreRef="jwtStore"
  oauthProviderRef="OIDCssl" >
</openidConnectProvider>
```

[1] See URL https://www.ibm.com/support/knowledgecenter/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_oidc_token_endpoint.html

[2] These credentials can be specified by the application

Sample program and JCL

COBOL Application

```
MOVE “ATSOAUTHUSERNAME" to envVariableName.
PERFORM CALL-CEEENV THRU CALL-CEEENV-END
MOVE VAR(1:valueLength) to BAQ-OAUTH-USERNAME
MOVE valueLength TO BAQ-OAUTH-USERNAME-LEN
MOVE “ATSOAUTHPASSWORD" to envVariableName.
PERFORM CALL-CEEENV THRU CALL-CEEENV-END
MOVE VAR(1:valueLength) to BAQ-OAUTH-PASSWORD
MOVE valueLength to BAQ-OAUTH-PASSWORD-LEN
MOVE " "         to BAQ-OAUTH-CLIENTID.
MOVE 0           to BAQ-OAUTH-CLIENTID-LEN.
MOVE " "         to BAQ-OAUTH-CLIENT-SECRET.
MOVE 0           to BAQ-OAUTH-CLIENT-SECRET-LEN.
MOVE "openid"    to BAQ-OAUTH-SCOPE.
MOVE 6           to BAQ-OAUTH-SCOPE-LEN.
SET BAQ-OAUTH-SCOPE-PTR TO ADDRESS OF BAQ-OAUTH-SCOPE.
```

Execution JCL

```
//GETAPI  EXEC PGM=GETAPIPT,PARM='111111'
//STEPLIB  DD DISP=SHR,DSN=USER1.ZCEE30.LOADLIB
//         DD DISP=SHR,DSN=ZCEE30.SBAQLIB
//         DD DISP=SHR,DSN=JOHNSON.ZCEE.SDFHLOAD
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEOPTS  DD *
 POSIX(ON),
 ENVAR("BAQURI=wg31.washington.ibm.com",
 "BAQPORT=9080",
 "BAQUSERNAME=USER1",
 “ATSAPPL=BBGZDFLT",
 “ATSOAUTHUSERNAME=distuser1",
 “ATSOAUTHPASSWORD=pwd“)
```

Note that this example is using environment variables to provide OAuth credentials, as documented in the z/OS Connect Advanced Topics Guide.

Security Scenarios

```
BAQ-OAUTH-USERNAME:  distuser1
BAQ-OAUTH-PASSWORD:  pwd
EmployeeNumber: 111111
EmployeeName:   C. BAKER
USERID:         USER1
```

distuser1 is mapped to RACF identity USER1 who has full access

```
BAQ-OAUTH-USERNAME:  distuserx
BAQ-OAUTH-PASSWORD:  pwd
Error code: 00000500
Error msg:{"errorMessage":"BAQR1092E: Authentication or authorization failed for the z/OS Connect EE server."}
```

distuserx is unknown by the OAuth Provider

```
BAQ-OAUTH-USERNAME:  auser
BAQ-OAUTH-PASSWORD:  pwd
Error code: 0000000403
rror msg:{"errorMessage":"BAQR1144E: Authentication or authorization failed for the z/OS Connect EE server."}
Syslog:
ICH408I USER(ATSSERV ) GROUP(ATSGRP   ) NAME(LIBERTY SERVER
  DISTRIBUTED IDENTITY IS NOT DEFINED:
  auser zCEERealm
```

auser is not mapped to a valid RACF identity

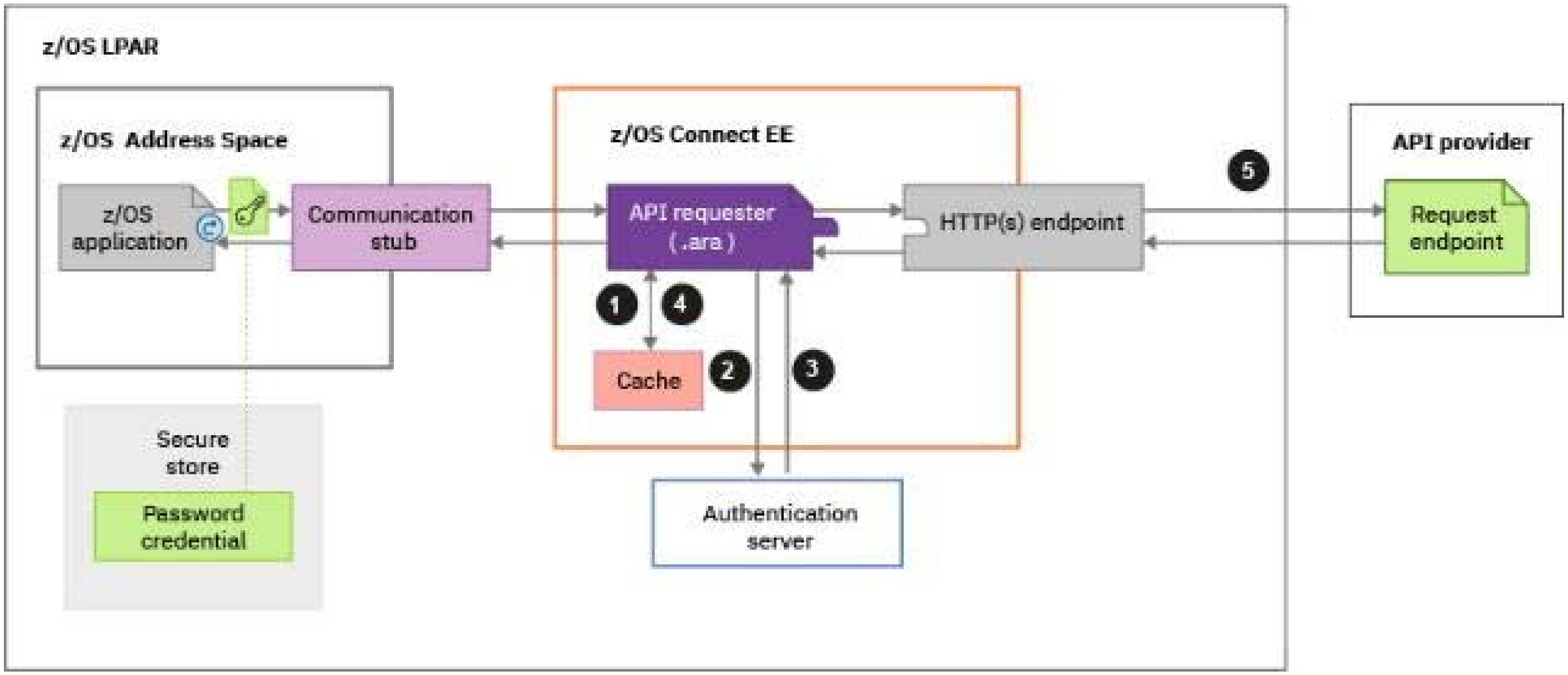
```
BAQ-OAUTH-USERNAME:  distuser2
BAQ-OAUTH-PASSWORD:  pwd
Error code: 0000000403
Error msg:{"errorMessage":"BAQR1144E: Authentication or authorization failed for the z/OS Connect EE server."}
Syslog:
ICH408I USER(USER2    ) GROUP(SYS1     ) NAME(WORKSHOP USER2        )
  ATSZDFLT.zos.connect.access.roles.zosConnectAccess
  CL(EJBROLE )
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ    )  ACCESS ALLOWED(NONE    )
```

distuser2 is mapped to RACF identity USER2 who has no access to the EJBRole protecting z/OS Connect

Calling an API with using a JWT custom flow

z/OS Connect provides two ways of calling an API secured with a JWT

- ❑ Use the OAuth 2.0 support when the request is part of an OAuth 2.0 flow as just described.
- ❑ In a non-OAuth 2.0 scenario, a JWT token is used in a custom flow, for example:
  - o When you need to specify the HTTP verb that is used in the request to the authentication server.
  - o When you need to specify how the JWT is returned from the authentication server (for example, in an HTTP header or in a custom field in a JSON response message).
  - o When you need to use a custom header name for sending the JWT to the request endpoint.



API Requester – JWT Custom flow

BAQRINFO copy book

```
05 BAQ-AUTHTOKEN.
   07 BAQ-TOKEN-USERNAME            PIC X(256).
   07 BAQ-TOKEN-USERNAME-LEN        PIC S9(9) COMP-5 SYNC VALUE 0.
   07 BAQ-TOKEN-PASSWORD            PIC X(256).
   07 BAQ-TOKEN-PASSWORD-LEN        PIC S9(9) COMP-5 SYNC VALUE 0.
```

COBOL application

```
         MOVE “ATSTOKENUSERNAME" to envVariableName.
         PERFORM CALL-CEEENV THRU CALL-CEEENV-END
         MOVE VAR(1:valueLength) to BAQ-TOKEN-USERNAME
         MOVE valueLength TO BAQ-TOKEN-USERNAME-LEN
         MOVE “ATSTOKENPASSWORD" to envVariableName.
         PERFORM CALL-CEEENV THRU CALL-CEEENV-END
         MOVE VAR(1:valueLength) to BAQ-TOKEN-PASSWORD
         MOVE valueLength to BAQ-TOKEN-PASSWORD-LEN
```

Note that this example is using environment variables to provide token credentials, as documented in the z/OS Connect Advanced Topics Guide.

Configuring JWT Custom flow

```
<zosconnect_endpointConnection id="cscvincAPI"
       host="http://wg31.washington.ibm.com" port="9080"
       authenticationConfigRef="myJWTConfig"/>

<zosconnect_authConfig id="myJWTConfig" authServerRef="myJWTServer"
        header="myJWT-header-name"
      <tokenRequest/>    See next slide
      <tokenReponse/>     See next slide
</zosconnect_authToken>

<zosconnect_authorizationServer id="myJWTServer"
      tokenEndpoint=https://wg31.washington.ibm.com:59443/oidc/endpoint/OP/token[1]
      basicAuthRef="tokenCredential"  [2]
      sslCertsRef="OutboundSSLSettings" />

<zosconnect_authData id="tokenCredential" [2]
  user="zCEEClient" password="secret"/>
```

[1] See URL https://www.ibm.com/support/knowledgecenter/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_oidc_token_endpoint.html

[2] These credentials can be specified by the application

Configuring Custom JWT flow

Request Token Example 1

```
<tokenRequest
     credentialLocation="header"
     header="Authorization"
     requestMethod="GET" />
```

Response Token

```
<tokenResponse
    tokenLocation="header"
    header="JWTAuthorization" />
```

Response Token Example 2

```
<tokenRequest credentialLocation="body"
    requestMethod="POST"
   // Use XML escaped characters in requestBody
   requestBody="{&quot;apiuser&quot;:&quot;${userid}&quot;,&quot;apipassword&quot;:&quot;${password}&quot;}" />
```

Response Token

```
<tokenResponse
     tokenLocation="body"
   responseFormat="JSON"
   tokenPath="$.tokenname" />
```

Summary

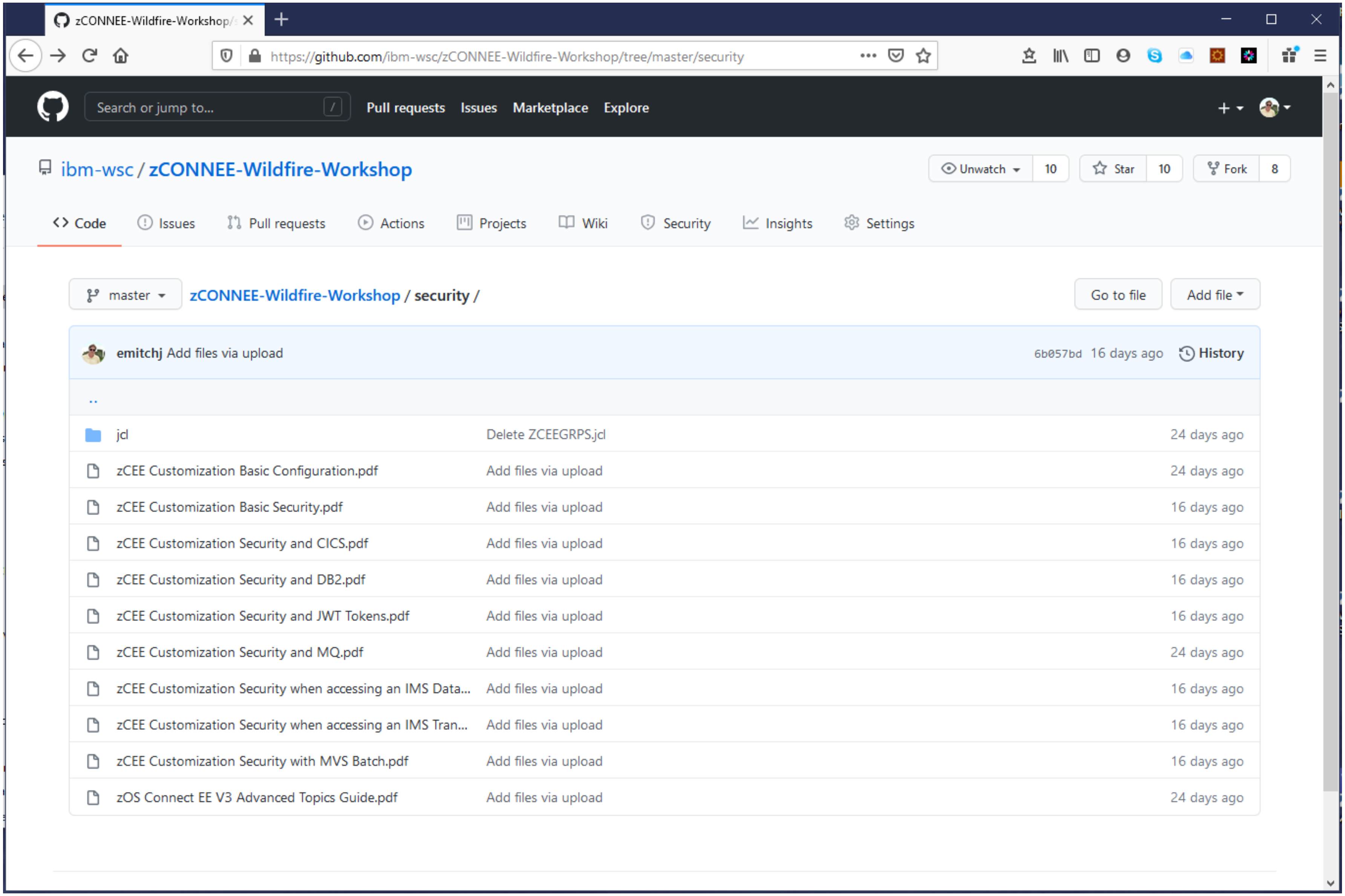
- Remember that because z/OS Connect EE is based on Liberty, it benefits from a wide range of Liberty security capabilities

- Security design needs to consider
  - Authentication
  - Encryption
  - Authorization

- Understand your security requirements, identify the waypoints and their security requirements.

- Consider security requirements from ending to beginning (not necessarily from beginning to end), e.g.
  - Do you need to flow the original authenticated identity all the way to the end?
  - Do you need to map individual identities to an application or server identity?

Step by step exercises available on GitHub



ibm-wsc / zCONNEE-Wildfire-Workshop

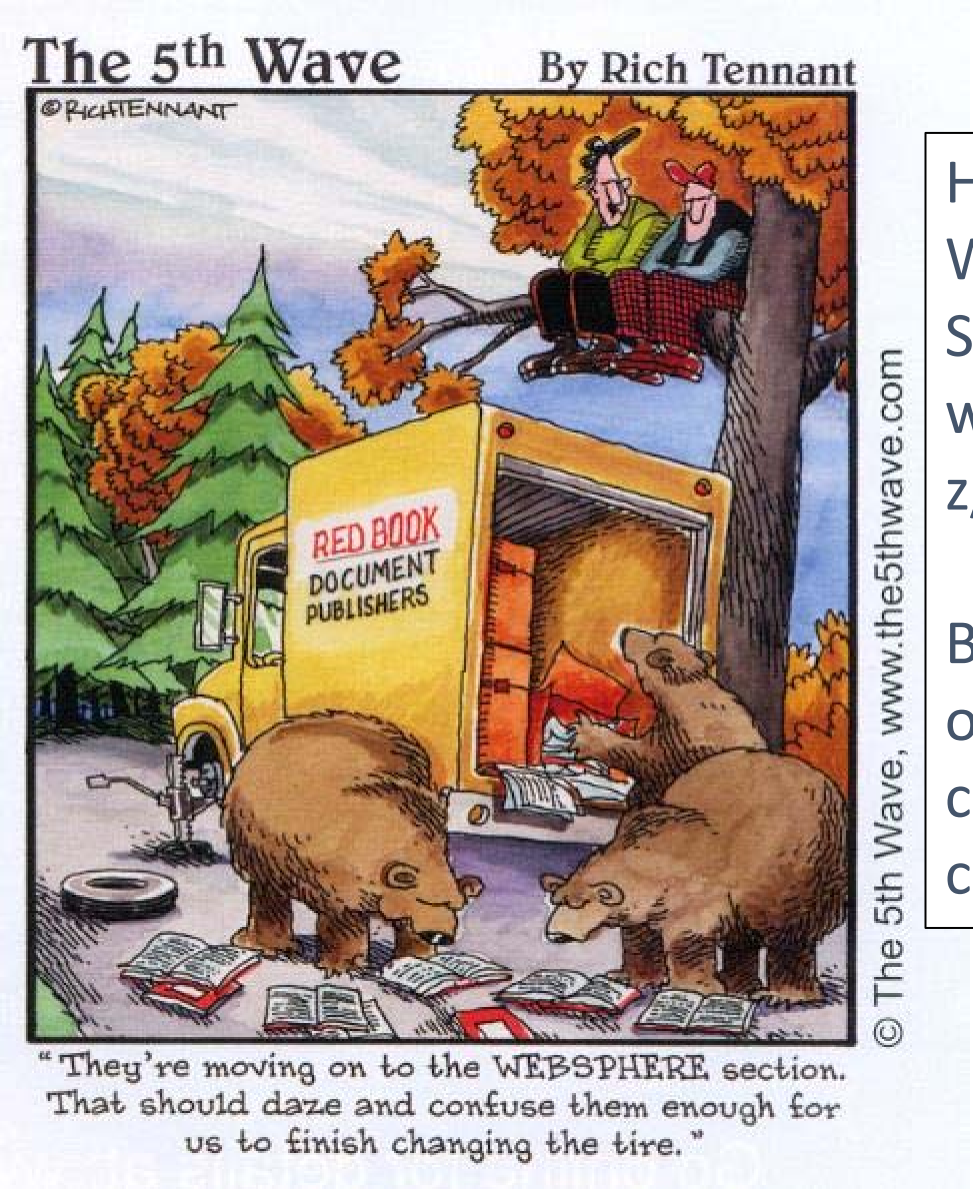
zCONNEE-Wildfire-Workshop / security /

emitchj Add files via upload — 6b057bd 16 days ago

..		
jcl	Delete ZCEEGRPS.jcl	24 days ago
zCEE Customization Basic Configuration.pdf	Add files via upload	24 days ago
zCEE Customization Basic Security.pdf	Add files via upload	16 days ago
zCEE Customization Security and CICS.pdf	Add files via upload	16 days ago
zCEE Customization Security and DB2.pdf	Add files via upload	16 days ago
zCEE Customization Security and JWT Tokens.pdf	Add files via upload	16 days ago
zCEE Customization Security and MQ.pdf	Add files via upload	24 days ago
zCEE Customization Security when accessing an IMS Data...	Add files via upload	16 days ago
zCEE Customization Security when accessing an IMS Tran...	Add files via upload	16 days ago
zCEE Customization Security with MVS Batch.pdf	Add files via upload	16 days ago
zOS Connect EE V3 Advanced Topics Guide.pdf	Add files via upload	24 days ago

https://github.com/ibm-wsc/zCONNEE-Wildfire-Workshop/tree/master/exercises

With apologies to Rich Tennant


The 5th Wave
By Rich Tennant
©RICHTENNANT
RED BOOK
DOCUMENT
PUBLISHERS
“They’re moving on to the WEBSPHERE section. That should daze and confuse them enough for us to finish changing the tire.”
© The 5th Wave, www.the5thwave.com

Hopefully, this situation with WebSphere Application Server does not apply now with security for Liberty or z/OS Connect.

But don’t hesitate to reach out to your local Tech Sales contact to ask for clarification or assistance.

/questions?thanks=true

Thank you for listening.