zOSSEC1 – IBM z/OS Connect EE Security

A dive into Liberty and z/OS Connect Security

Mitch Johnson
mitchj@us.ibm.com

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 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.

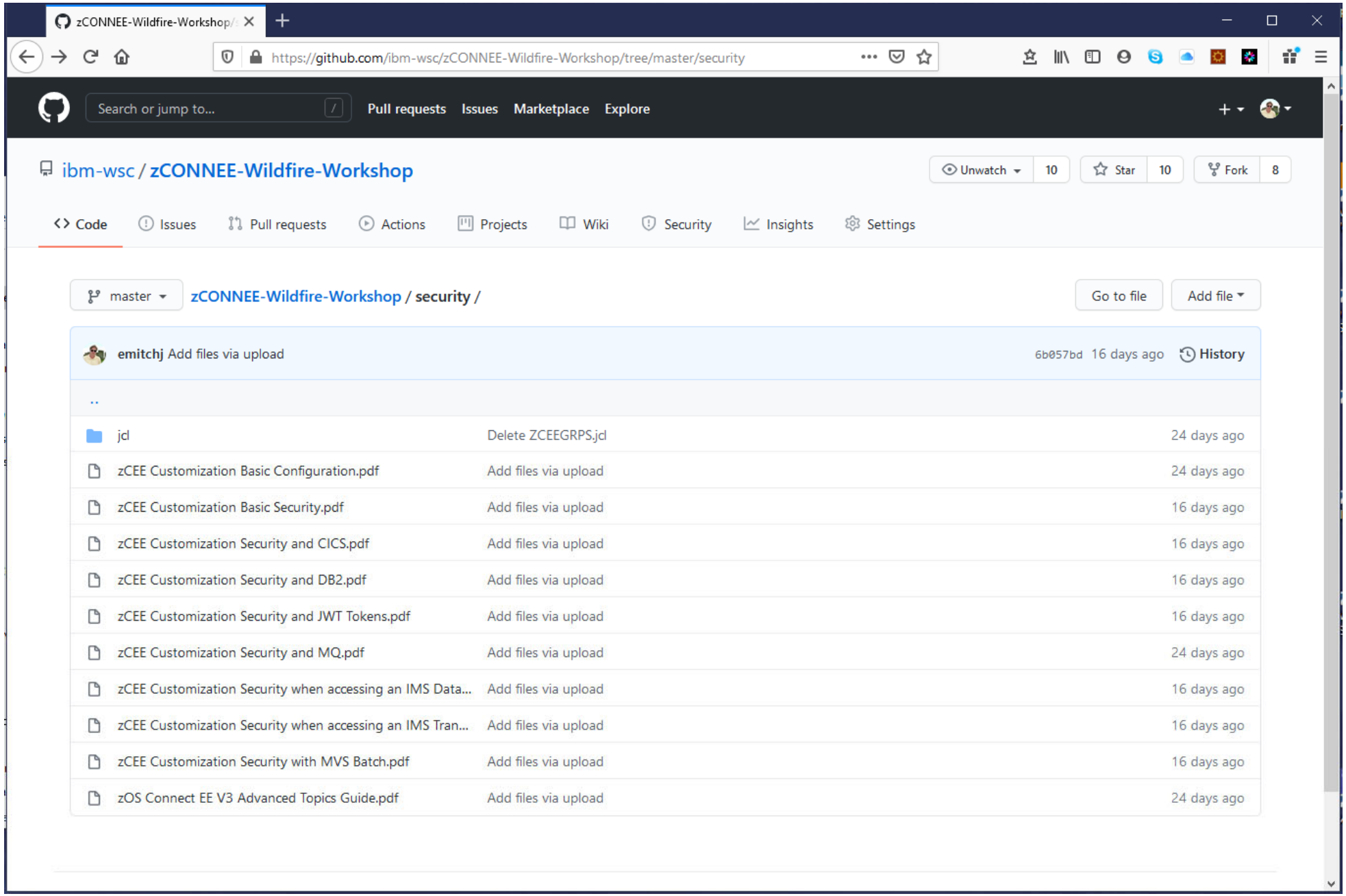
- There will be additional information on slides that will designated as Tech/Tips. These contain information that at perhaps at least interesting and hopefully, useful to the reader.

- 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.

mitchj@us.ibm.com

Detailed examples of topics covered today are available



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

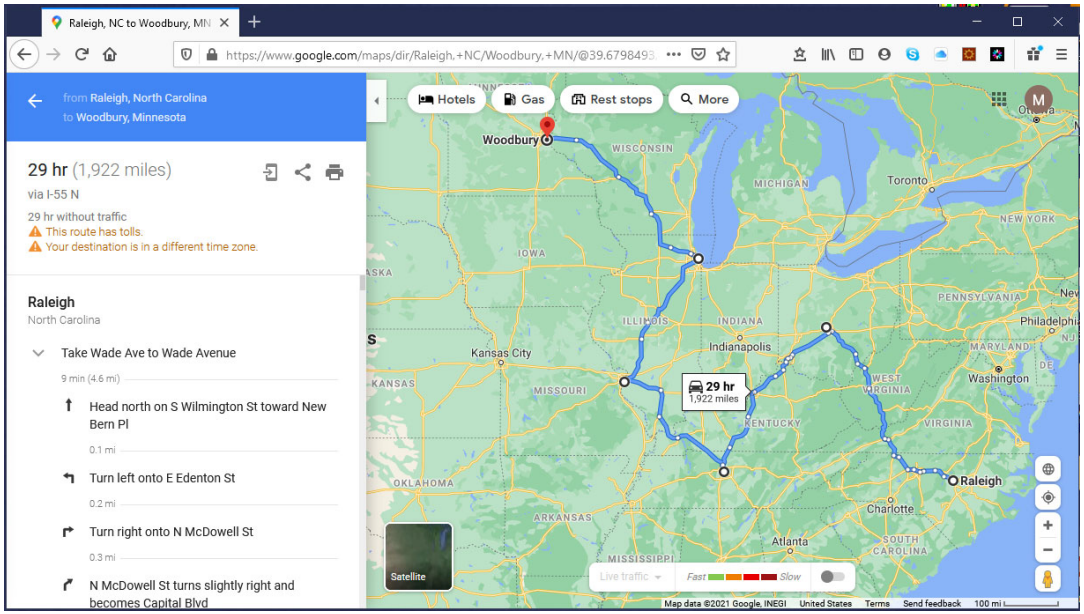
mitchj@us.ibm.com

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:



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 addressed.

mitchj@us.ibm.com

We need to understand the challenges

- Providing secure access between middleware components that use disparate security technologies e.g., registries like LDAP, SAF, TLS etc. in order 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

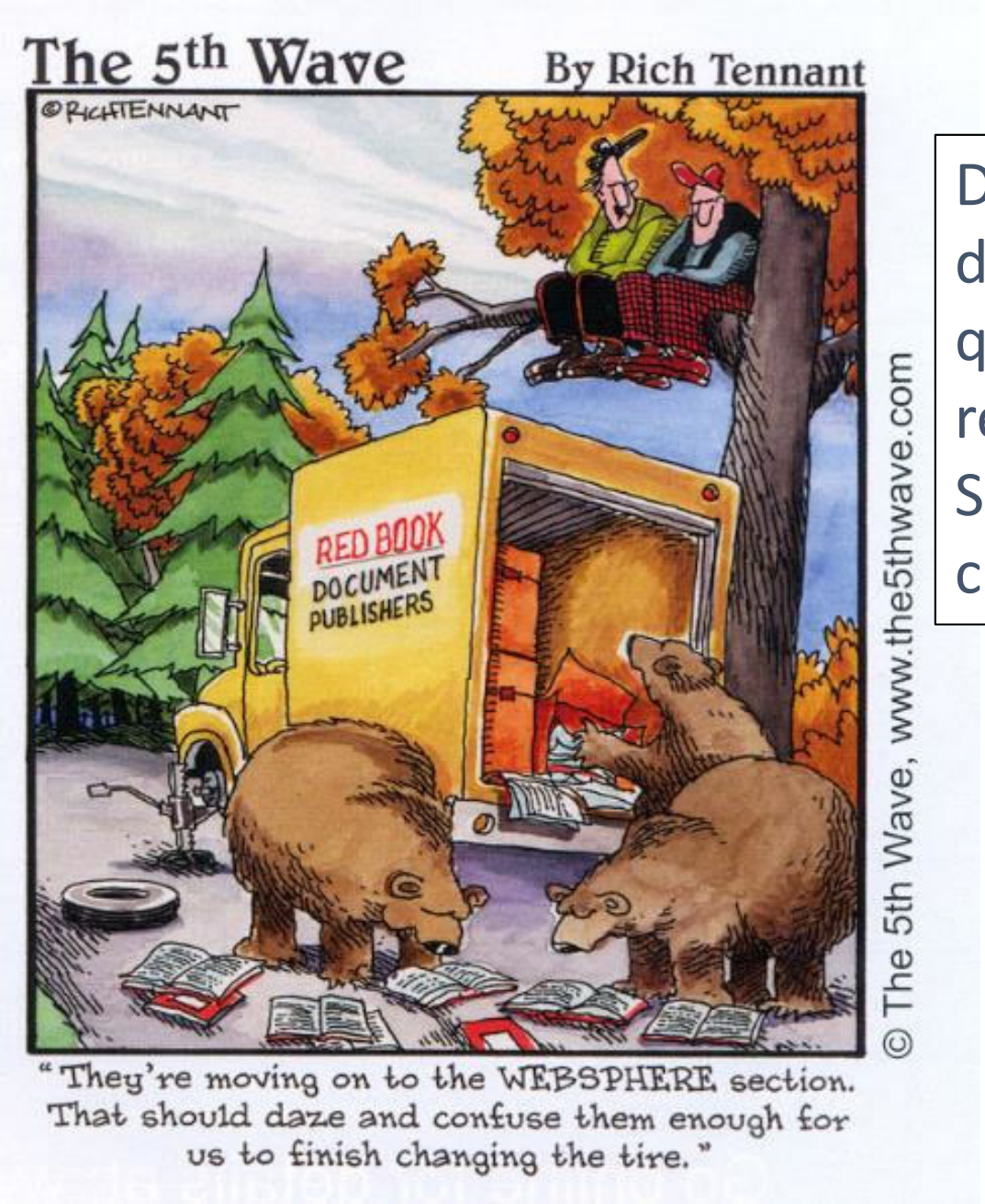
- Considering that security is often at odds with **performance**, the more secure techniques often mean more processing overhead, especially if not configured optimally

➢Remember security is probably not a choice but a requirement.

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.

mitchj@us.ibm.com

With apologies to Rich Tennant



During today's presentation don't hesitate to ask questions,or in the future, to reach out to your local Tech Sales contact to ask for clarification or assistance.

Let’s begin by reviewing some of the basic OMVS, Liberty and RACF security/configuration details and options

OMVS security - Unix file permissions

Owner

	Read	Write	Execute
Bit	1	1	1
Base-2 Value	[4]	[2]	[1]

4 + 2 + 1 =

7 The owner has READ, WRITE and EXECUTE

The **owner** of the file or directory

chmod –R* u+rwx zceesrv1

Group

	Read	Write	Execute
Bit	1	0	1
Base-2 Value	[4]	[2]	[1]

4 + 0 + 1 =

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

	Read	Write	Execute
Bit	0	0	0
Base-2 Value	[4]	[2]	[1]

0 + 0 + 0 =

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

-R* indicates recursion

mitchj@us.ibm.com

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**)
- If you have multiple people with a need to read or update configuration files, do you simply connect them to LIBGRP? (answer: **no**)

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)

```
export WLP_USER_DIR=/var/zosconnect
cd $WLP_USER_DIR
chmod a+x $WLP_USER_DIR/servers
chmod a+x $WLP_USER_DIR/servers/zceesrvr
```

mitchj@us.ibm.com

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              751  LIBSERV LIBGRP
  /servers                   751  LIBSERV LIBGRP
    /zceesrv1                751  LIBSERV LIBGRP
      /logs                  771  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/WRITE through that group.

CWWKB0121I: The server process UMASK value is set to 0000

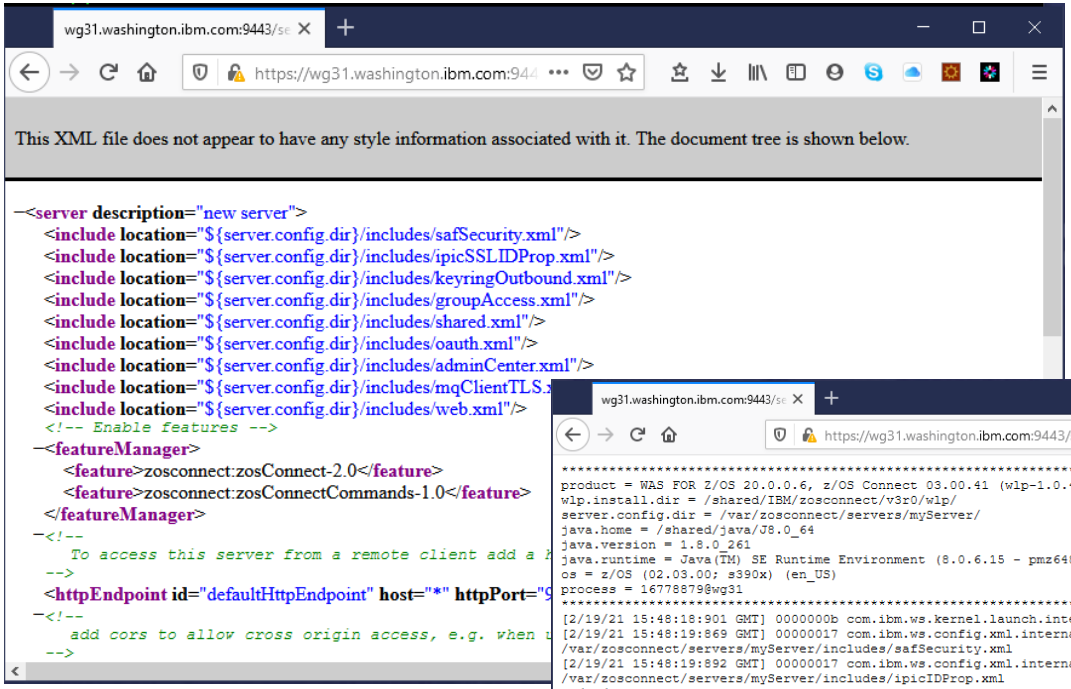
- sets permission bit for new files deployed using the RESTful artifacts to rw-rw-rw (666 x'OR 000)

~~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

mitchj@us.ibm.com

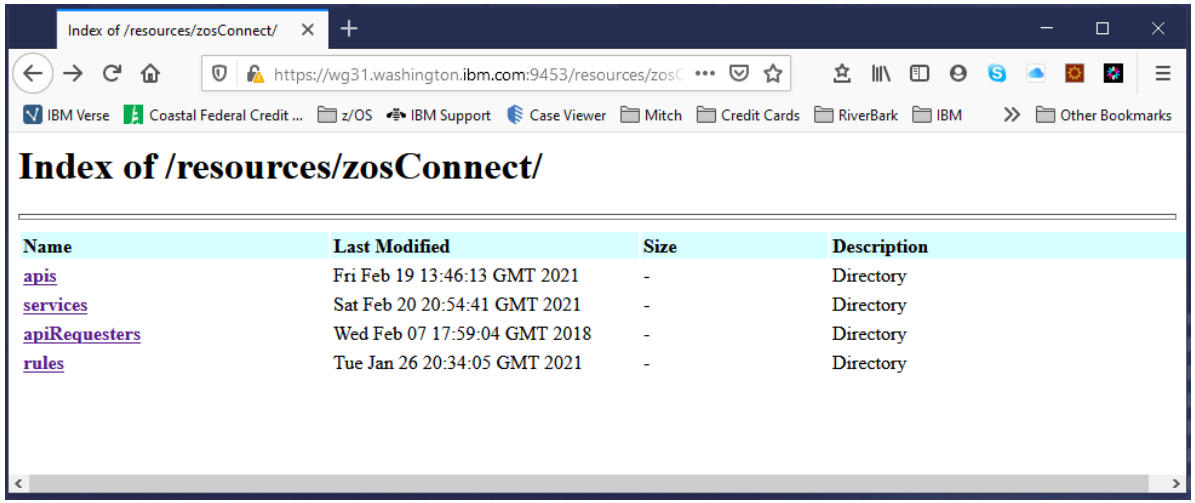
Tech/Tip: Providing access to configuration/log information



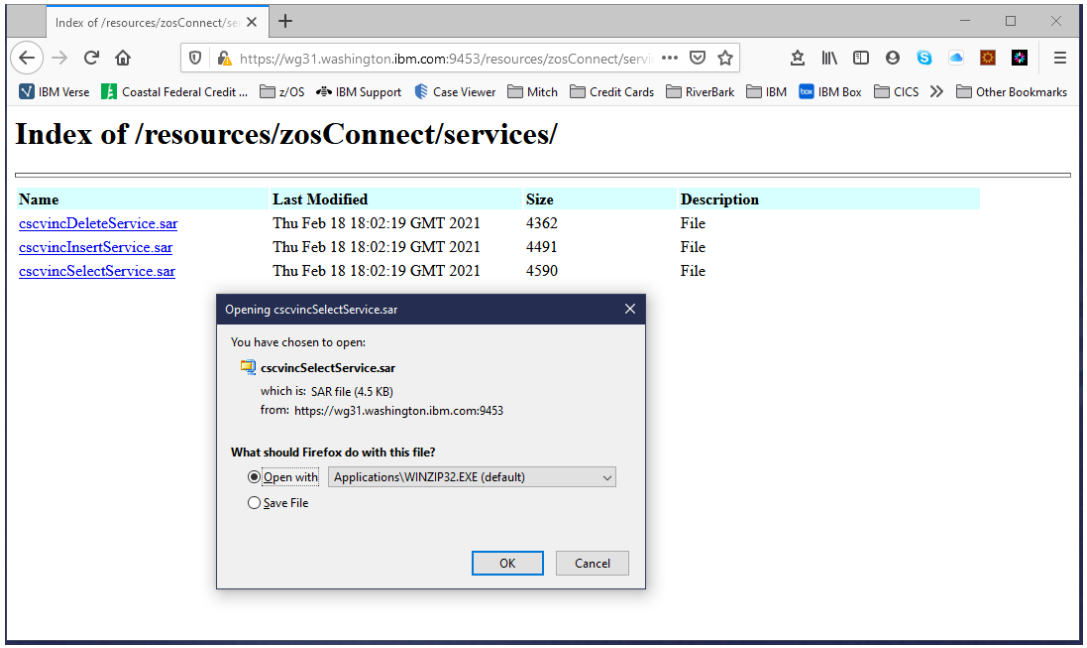

```
<webApplication id="serverConfig-location" name="serverConfig"
  location="${server.config.dir}">
  <web-ext context-root="/server/config"
    enable-file-serving="true" enable-directory-browsing="true">
    <file-serving-attribute name="extendedDocumentRoot"
      value="${server.config.dir}" />
  </web-ext>
</webApplication>
```


mitchj@us.ibm.com

Tech/Tip: Providing access to service archives files

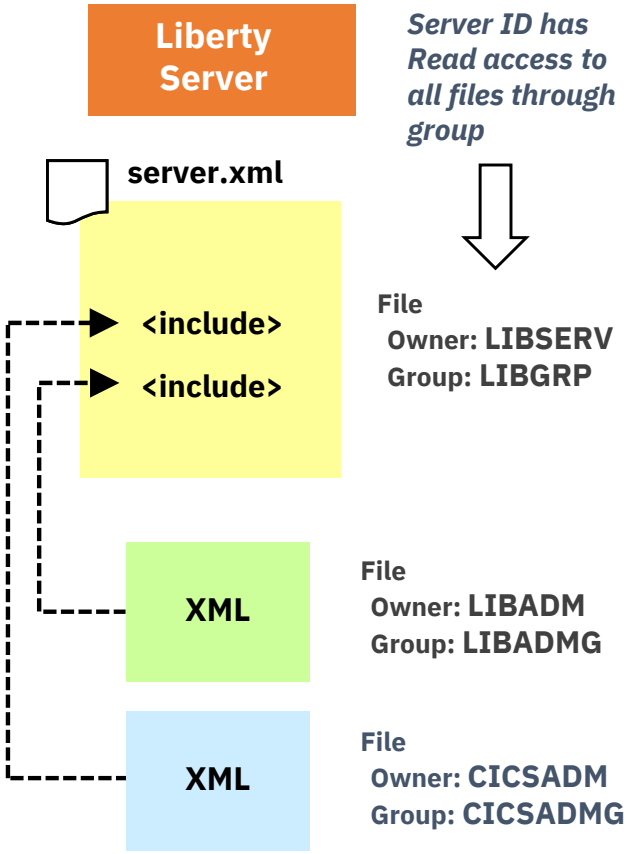



```
<webApplication id="resources-location" name="resources"
  location="${server.config.dir}/resources/zosconnect">
  <web-ext context-root="/resources/zosConnect"
    enable-file-serving="true" enable-directory-browsing="true">
    <file-serving-attribute name="extendedDocumentRoot"
      value="${server.config.dir}/resources/zosconnect" />
  </web-ext>
</webApplication>
```



mitchj@us.ibm.com

Liberty supports server XML “include” file processing



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 ensure 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.

mitchj@us.ibm.com

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

- Setup a server.xml using ‘include’ statements and allow other administrator to manage 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”/>
<include location="${server.config.dir}/includes/imsDB.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>
```

imsDB.xml (owned by a IMS administarator)

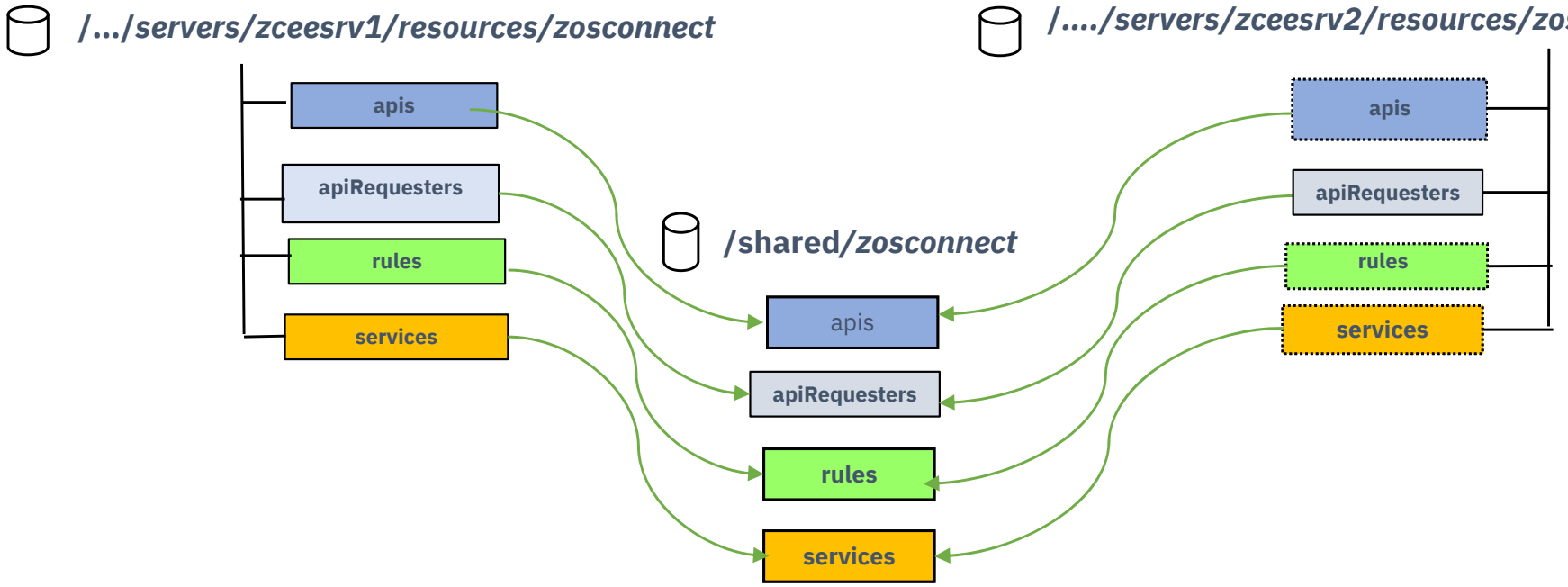
```
<server description="IMS DATABASE">
<featureManager>
    <feature>zosconnect:dbService-1.0</feature>
</featureManager>
<connectionFactory id="DFSIVPAConn">
<properties.imsudbJLocal  databaseName="DFSIVPA“
datastoreName="IVP1“driverType="4“   portNumber="5555"
datastoreServer="wg31.washington.ibm.com"
 user="USER1"   password="USER1“   flattenTables="True"/>
 </connectionFactory>
</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>
```


mitchj@us.ibm.com

Remember the use of symbolic links to share artifacts



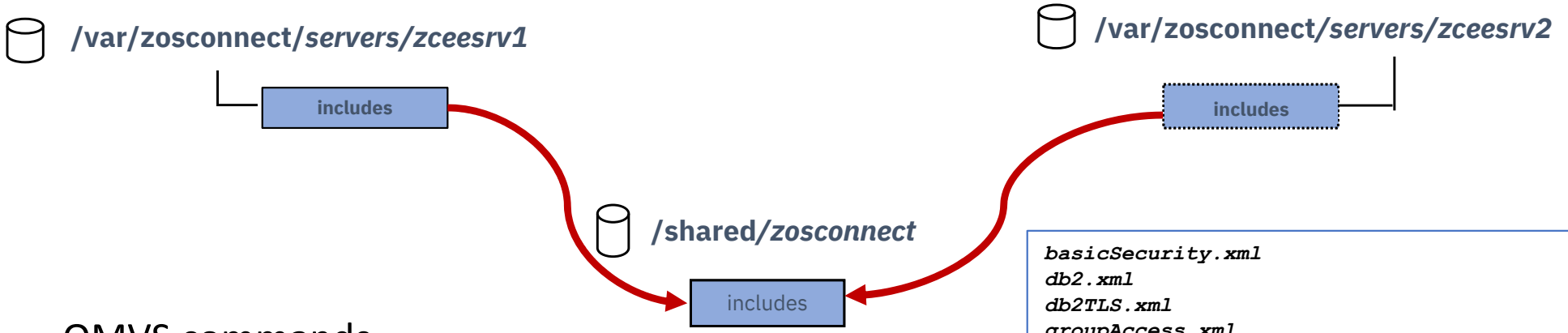
Use symbolic links

```
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

mitchj@us.ibm.com

Sharing security XML configuration files between servers




```
basicSecurity.xml
db2.xml
db2TLS.xml
groupAccess.xml
ipic.xml
ipicIDProp.xml
keyringInbound.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 shared location which then can be accessed from multiple servers

F BAQSTRT,REFRESH,CONFIG

mitchj@us.ibm.com

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

tinyurl.com/3xc76kpp

mitchj@us.ibm.com

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>
```


mitchj@us.ibm.com

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"/>
```


mitchj@us.ibm.com

z/OS : Starting Liberty Servers

All three options result in a Liberty z/OS server, and functionally there's very little difference.

When started as a UNIX process, the MODIFY command interface is not present. For production use, the best practice is to use a started task.

UNIX Process
Start with shell script

1. **Liberty z/OS Server Instance**

Started Task
Start with shell script

2. **Liberty z/OS Server Instance**

Started Task
Use z/OS START

3. **Liberty z/OS Server Instance**

1. UNIX Process
   - Use the 'server' shell script in the installation /bin directory
   - Syntax: *server start zceesvr1*
   - ID of server will be based on ID that issued the command
2. Started Task using server shell script (server start zceesrv1)
   - Set **WLP_ZOS_PROCEDURE** environment variable in server.env file
   - Example: WLP_ZOS_PROCEDURE=ZCEEPROC,JOBNAME=ZCEESVR1,PARMS='ZCEESVR1'
   - ID of the server will be based on the SAF STARTED profile that takes effect
3. ***Started Task using START command***
   - ***Common procedure: START ZCEEPROC,JOBNAME=ZCEESVR1,PARMS='ZCEESVR1'***
   - ***Dedicated proc: START ZCEEPROC***
   - ***ID of the server will be based on the SAF STARTED profile that takes effect***

Expectation is for production servers #3 will be used

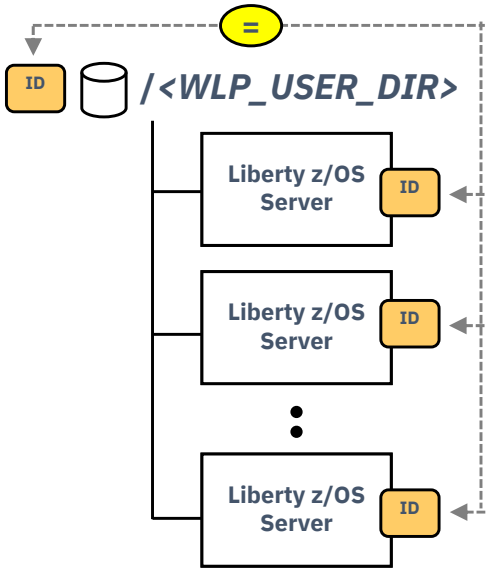
Liberty z/OS good practices:

https://www.ibm.com/support/pages/node/6355605

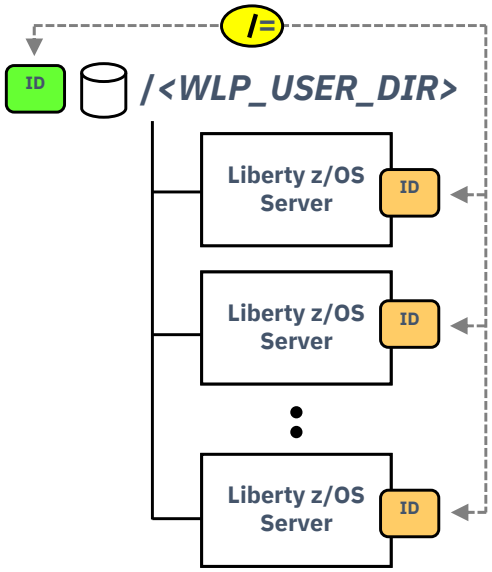
mitchj@us.ibm.com

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

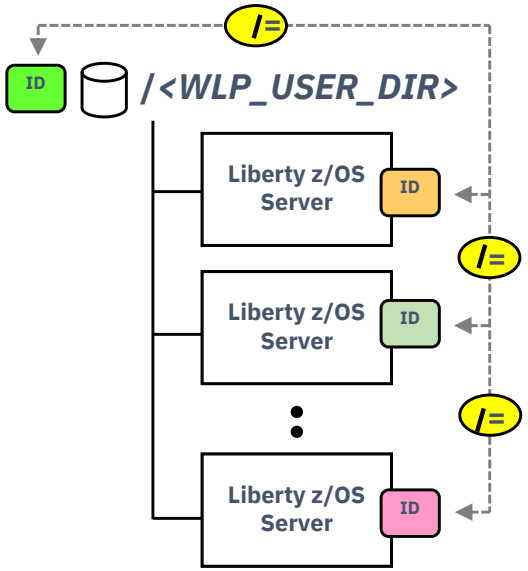
On z/OS, the best practice for Liberty servers in production is that they run as 'Started Tasks' (STCs).



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



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



- Multiple servers
- Different STC IDs
- STC IDs ≠ 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

mitchj@us.ibm.com

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 Identity per task	*Unique Identities per task*

Common JCL Procedure	1
	`RDEFINE STARTED ZCEEPROC.*`
	
	`S ZCEEPROC,JOBNAME=server1,PARMS='server1'`
	`S ZCEEPROC,JOBNAME=server2,PARMS='server2'`
		3
		`RDEFINE STARTED ZCEEPROC.server1`
		`RDEFINE STARTED ZCEEPROC.server1`
		
		`S ZCEEPROC,JOBNAME=server1,PARMS='server1'`
		`S ZCEEPROC,JOBNAME=server2,PARMS='server2'`

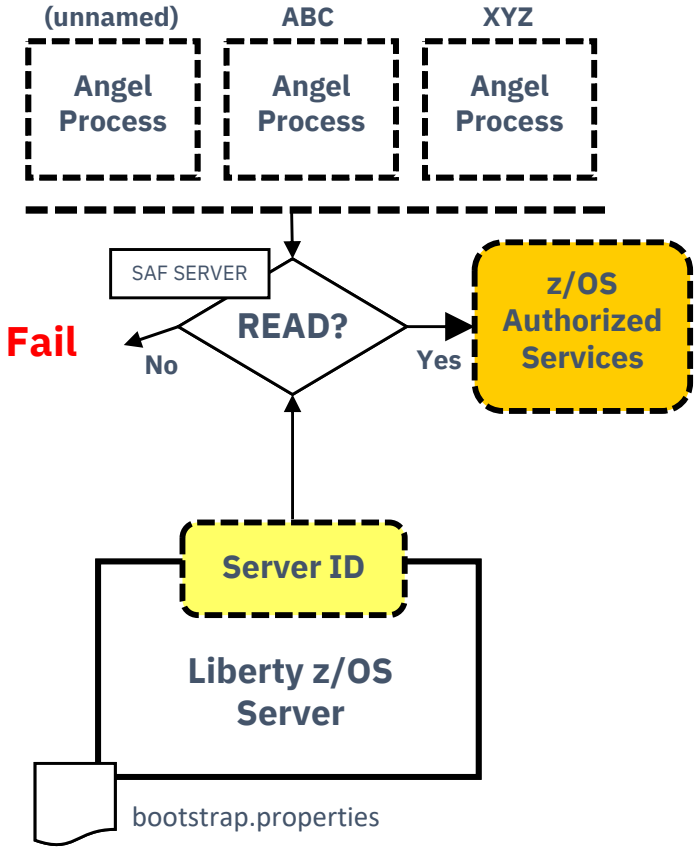
Unique JCL Procedure per server	2
	`RDEFINE STARTED ZCEE*.*`
	
	`S ZCEESRV1,JOBNAME=server1,PARMS='server1'`
	`S ZCEESRV2,JOBNAME=server2,PARMS='server2'`
		4
		`RDEFINE STARTED ZCEESRV1.*`
		`RDEFINE STARTED ZCEESRV2.*`
		
		`S ZCEESRV1,JOBNAME=server1,PARMS='server1'`
		`S ZCEESRV2,JOBNAME=server2,PARMS='server2'`

Note: Using unique JCL procedure eliminates the need to specify 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.

mitchj@us.ibm.com

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




```
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 privileged or 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:

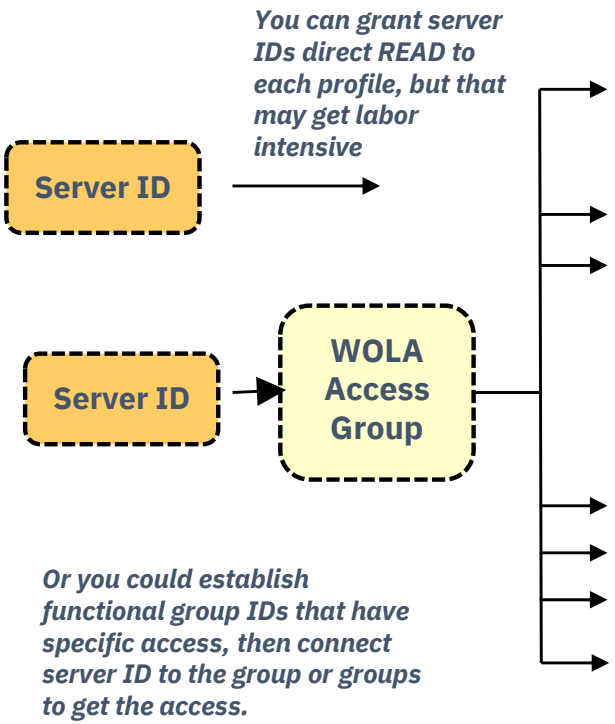
- **You may create separate named Angels for isolation of Test and Production, but do not take this practice too far. A few Angels, yes; dozens, no.**
- **Establish automation routines to start the Angels at IPL**
- **Grant SAF GROUP access to the SERVER profiles, then connect server IDs as needed**

Current list of Liberty Features

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

mitchj@us.ibm.com

z/OS : SAF SERVER profiles related to the Angel



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

Best practice:

- **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 parameter 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 server 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
Liberty 21.0.6 add a new property to identity required services, com.ibm.ws.zos.core.angelRequiredServices, for more details see URL
https://www.ibm.com/docs/en/was-liberty/zos?topic=overview-process-types-zos

mitchj@us.ibm.com

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 EJBROLE 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
https://www.ibm.com/support/knowledgecenter/SS4SVW_beta/securing/saf_unauthenticated_id.html#concept_saf_unauthenticated_id

mitchj@us.ibm.com

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 any identity in group *adminGrp* 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.

mitchj@us.ibm.com

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

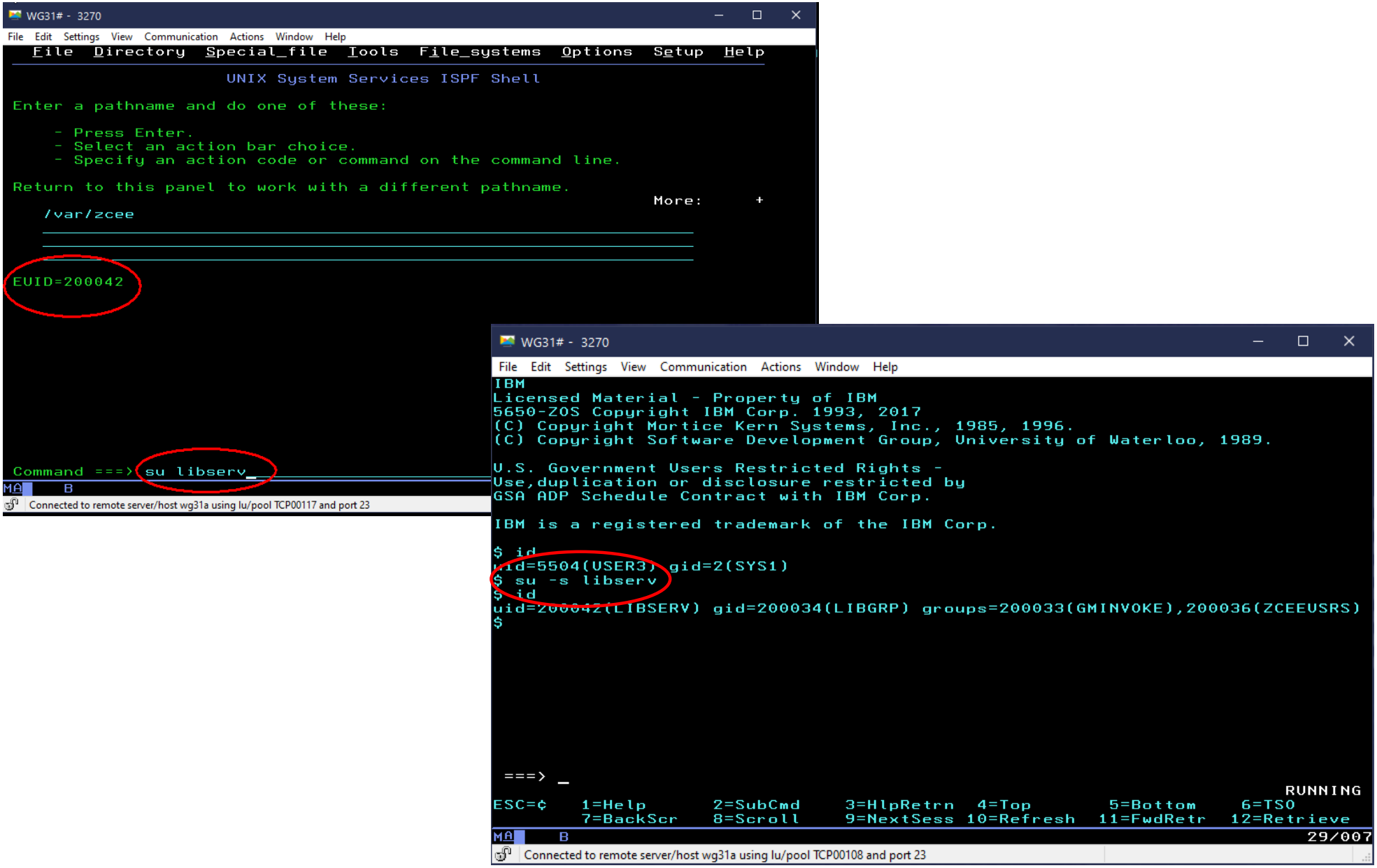
```
//MYSERVER JOB 'ZCEE',CLASS=A,REGION=0M,NOTIFY=&SYSUID,USER=LIBSERV
//*************************************************************
//*  SET SYMBOLS
//*************************************************************
//EXPORT EXPORT SYMLIST=(*)
// SET JAVAHOME='/usr/lpp/java/J8.0_64'
// SET ZCEEPATH='/usr/lpp/IBM/zosconnect/v3r0'
// 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
//*************************************************************
//*  Step CHOWN - Change directory and file ownership
//*************************************************************
//CHOWN  EXEC PGM=IKJEFT01,REGION=0M
//SYSERR   DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *,SYMBOLS=EXECSYS
 BPXBATCH SH +
 export WLP_USER_DIR=&WLPUSER; +
 chown -R &USER:&GROUP $WLP_USER_DIR/servers/&SERVER
```

There is no need provide LIBSERV's password, in fact LIBSERV may be protected and not even have a password. Any files or directories created will be owned by LIBSERV.

Alternatively, use the change ownership command, *chown,* to change the user and group attributes.

mitchj@us.ibm.com

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



mitchj@us.ibm.com

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

mitchj@us.ibm.com

z/OS Connect Security

Overview

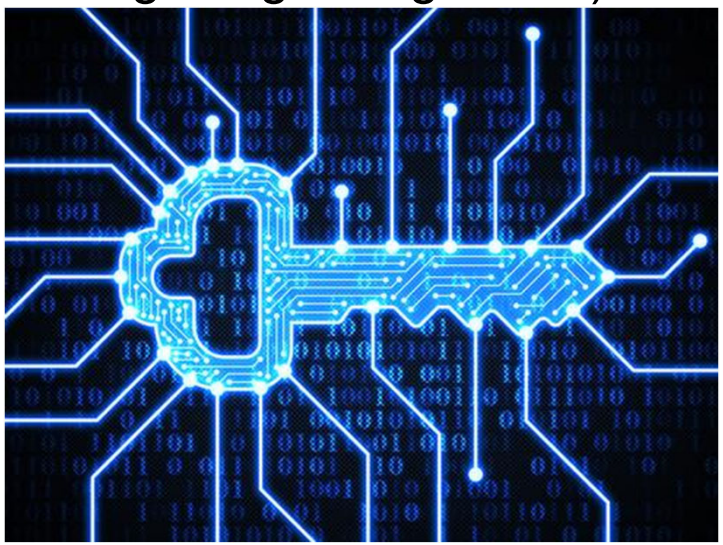
General security terms or considerations

Security involves

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

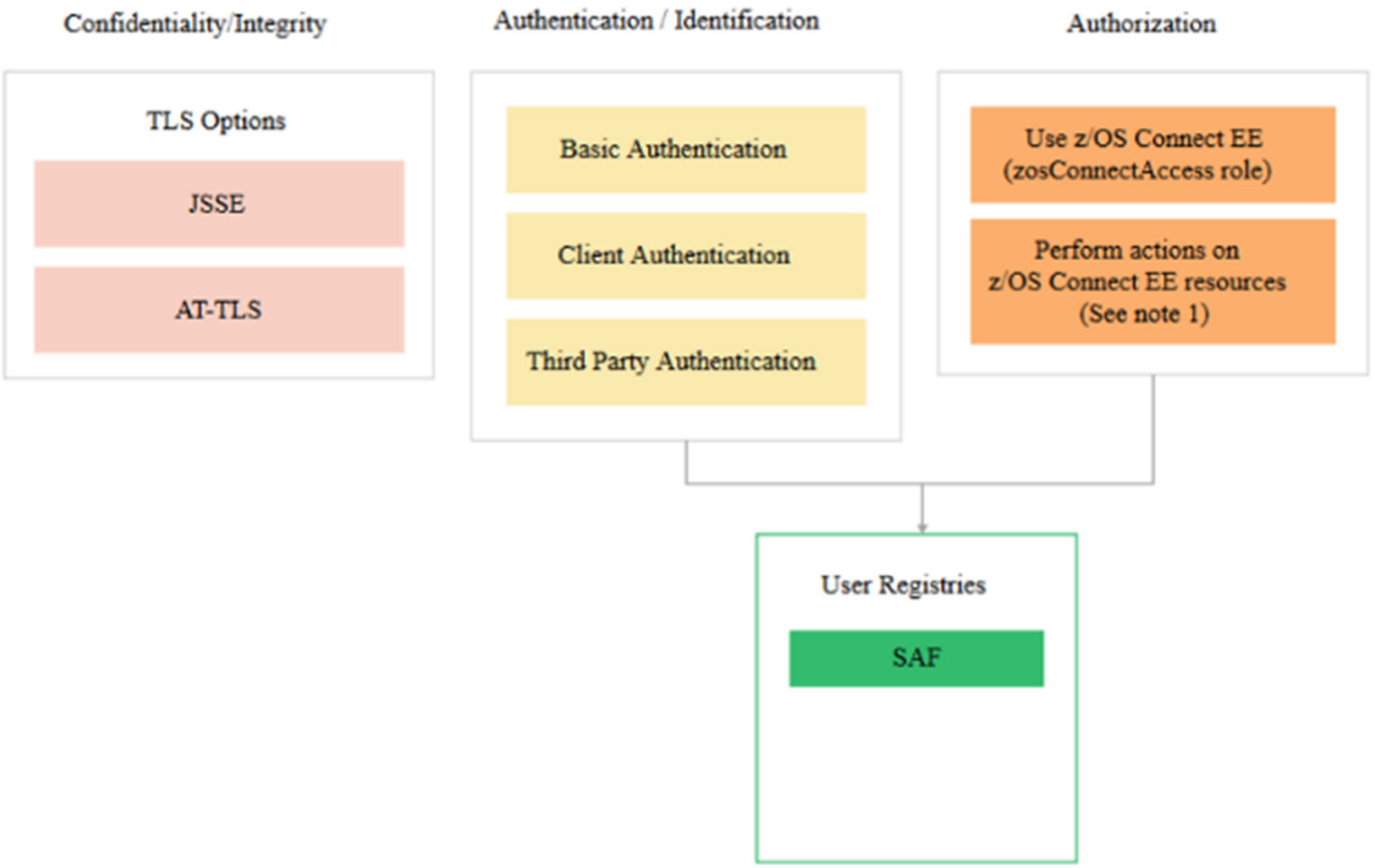
- Ensuring that the message has not been altered in transit (**Data Integrity)** and ensuring the confidentiality of the message in transit (**Encryption)**
  - TLS (encrypting messages and generating/sending a digital signature)

- 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.



mitchj@us.ibm.com

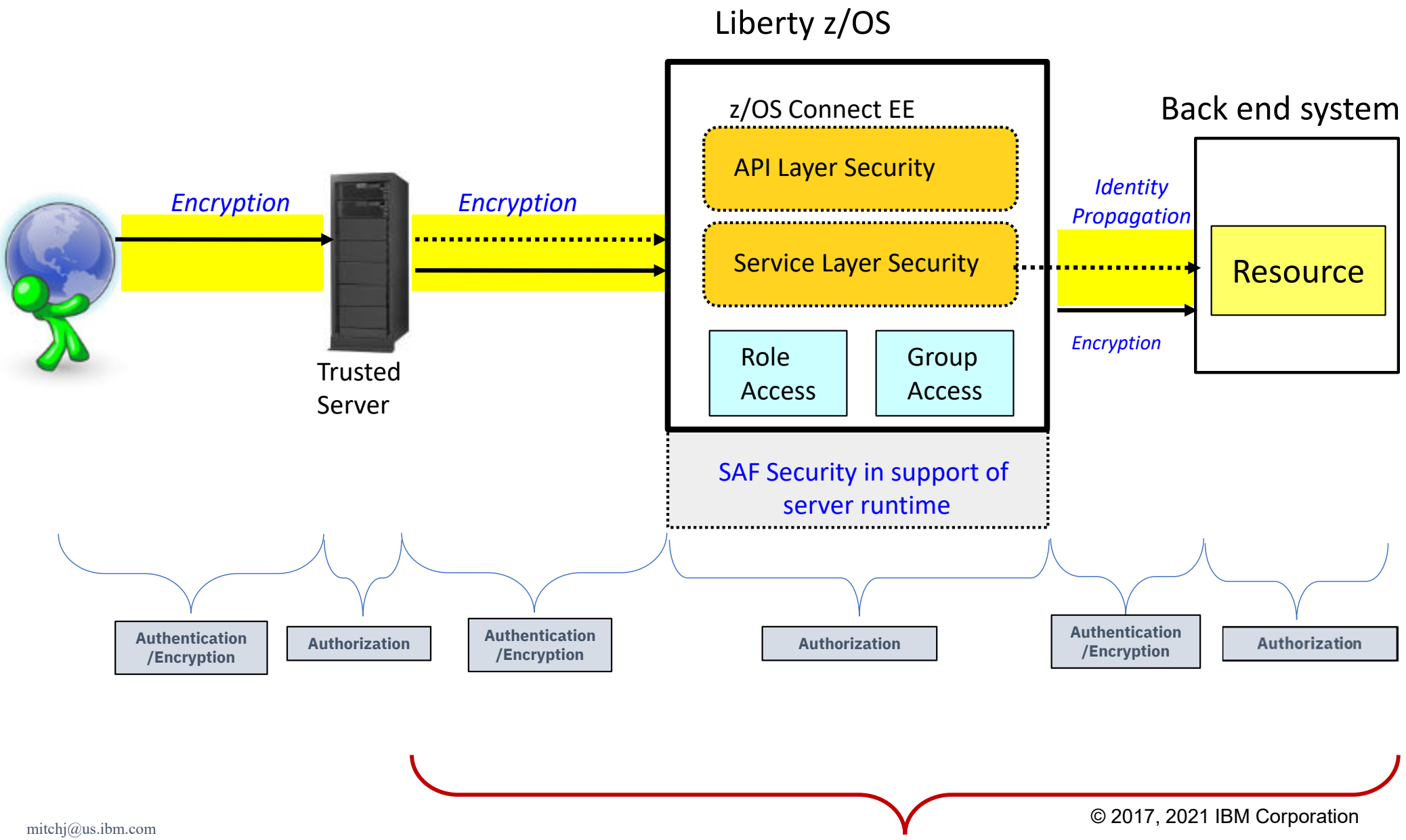
Liberty and z/OS Connect EE security options



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

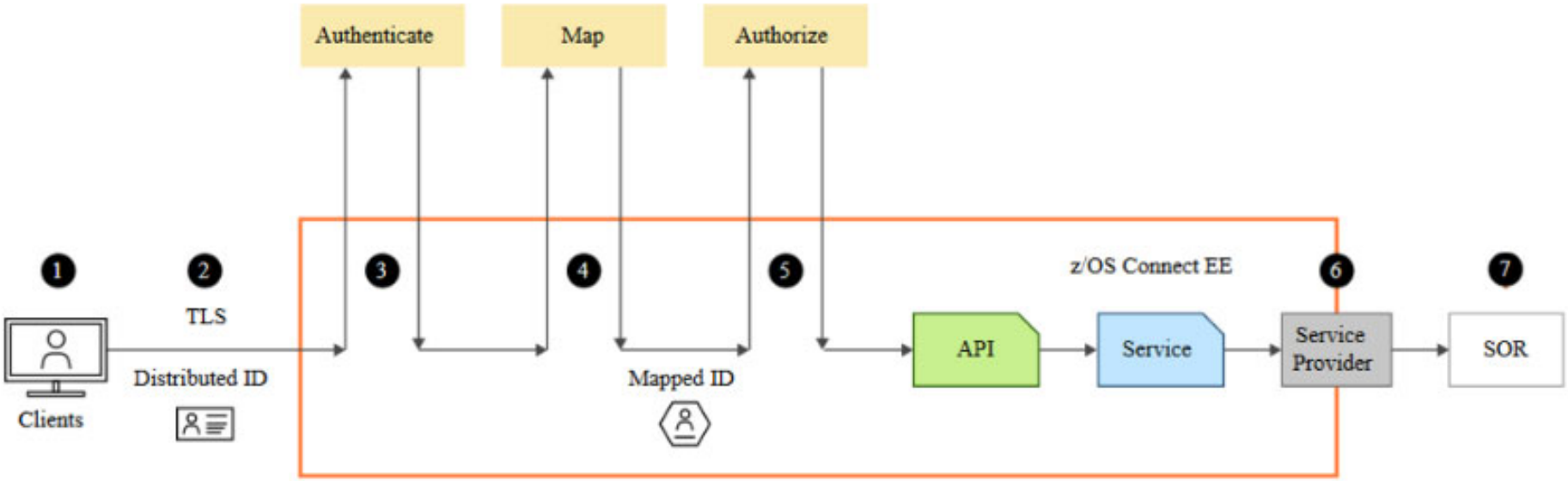
mitchj@us.ibm.com

API provider security overview (waypoints)



mitchj@us.ibm.com

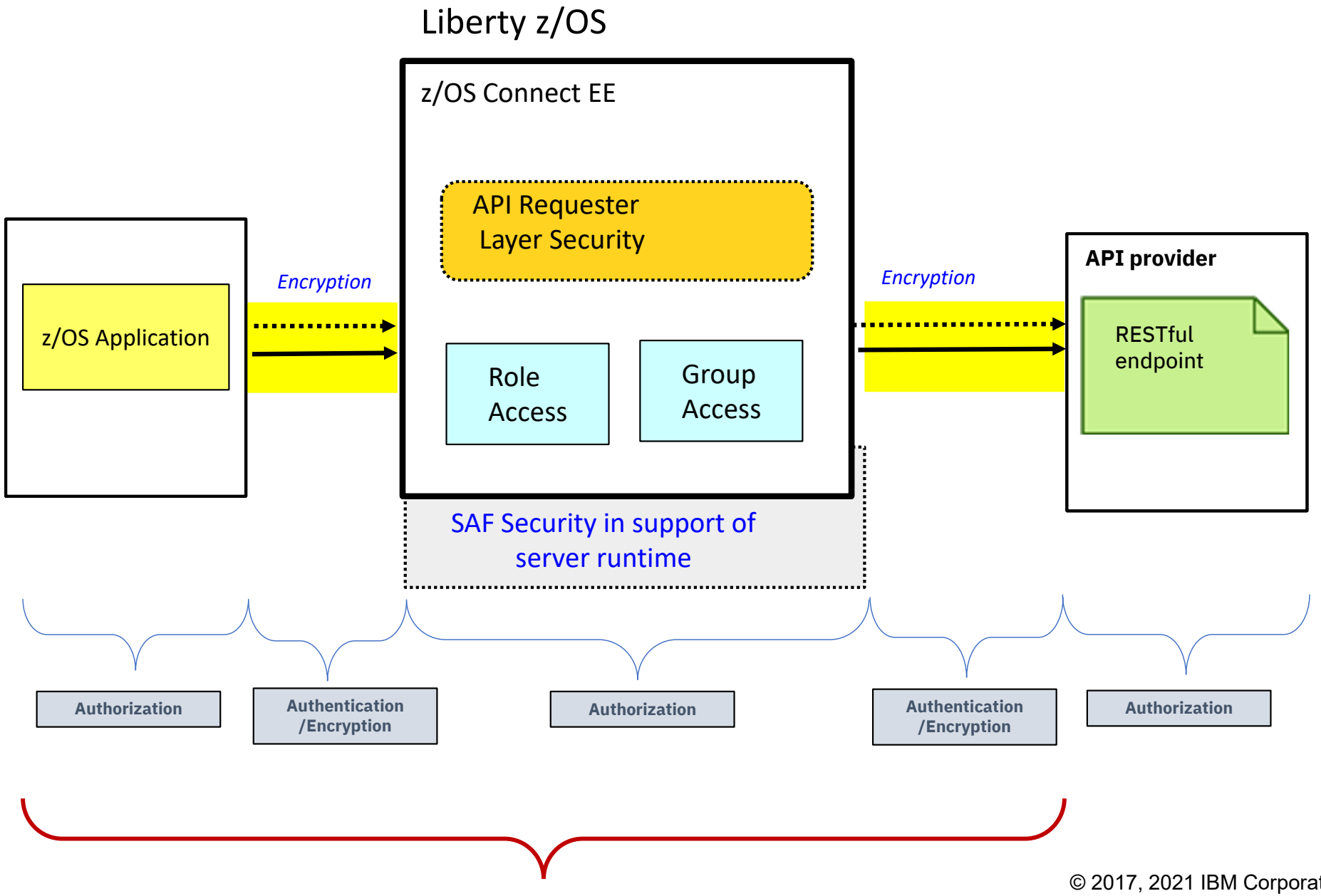
Details of a typical z/OS Connect EE API Provider security flow



1. The credentials provided by the client
2. Secure the connection to the Liberty server
3. Authenticate the client. This can be within the Liberty 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

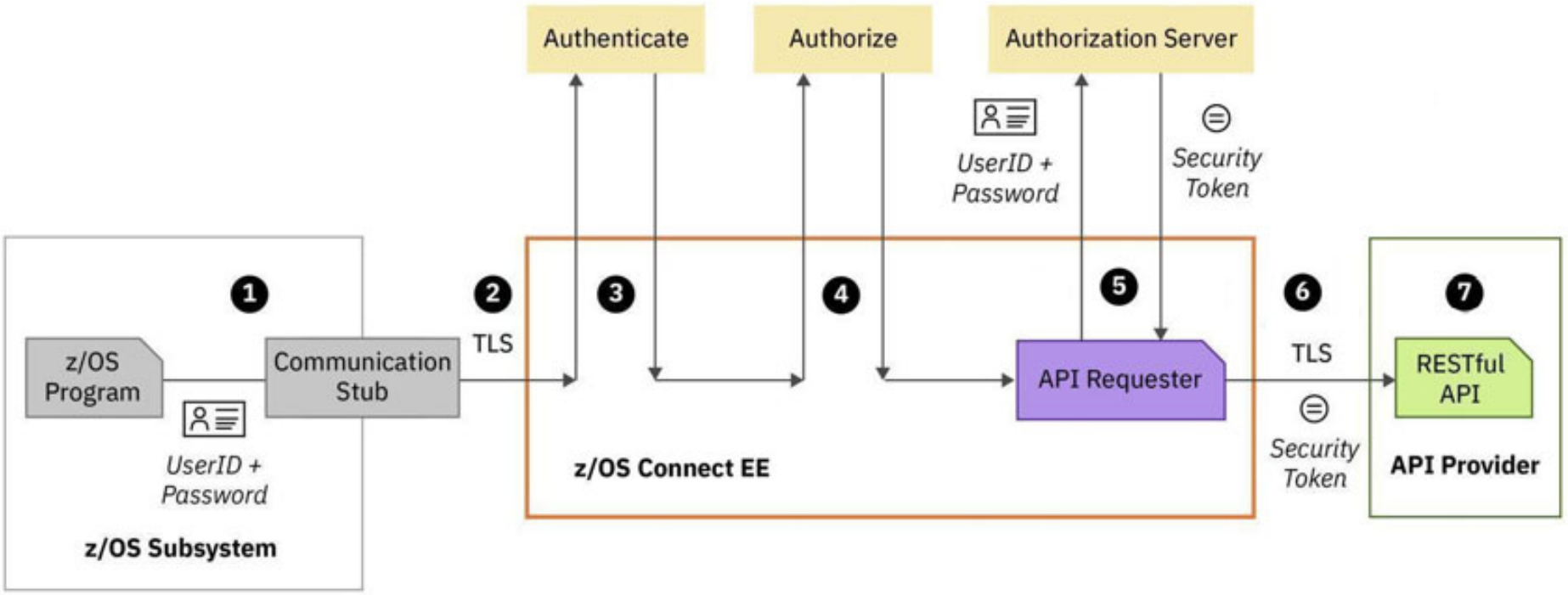
mitchj@us.ibm.com

API requester security – overview (waypoints)



mitchj@us.ibm.com

Details of a typical z/OS Connect EE API Requester security flow



1. A user ID and password can be used for basic authentication by the Liberty EE server
2. Connection between the CICS, IMS, or z/OS application and the Liberty server can use TLS
3. Authenticate the CICS, IMS, or z/OS application.
4. Authorize the authenticated user ID to connect to Liberty 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

mitchj@us.ibm.com

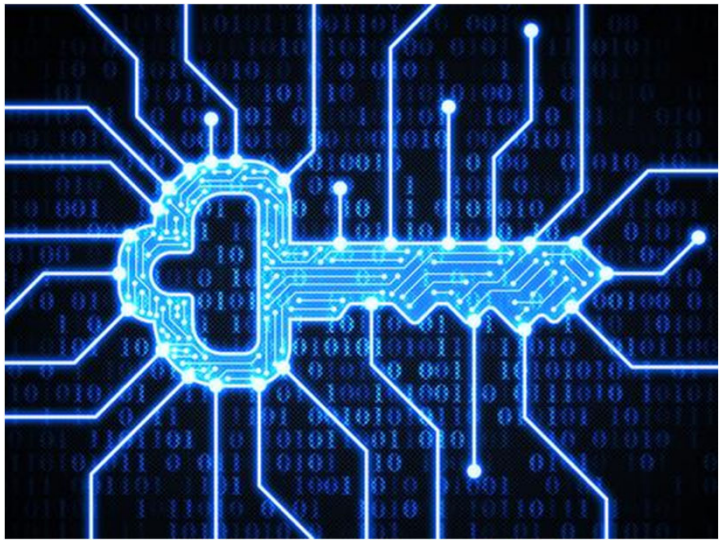
Now let’s explore security for inbound connections

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.

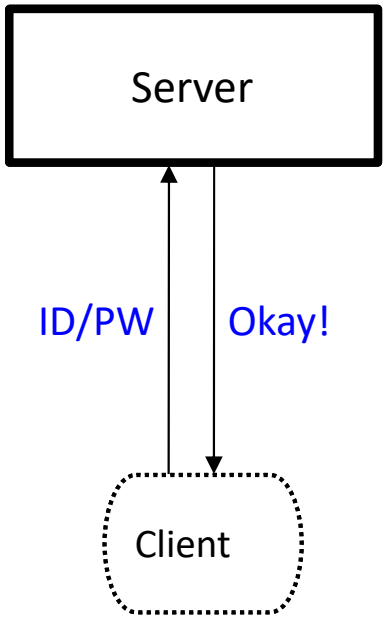


mitchj@us.ibm.com

Liberty Authentication Options

Several different ways this can be accomplished:

Basic Authentication

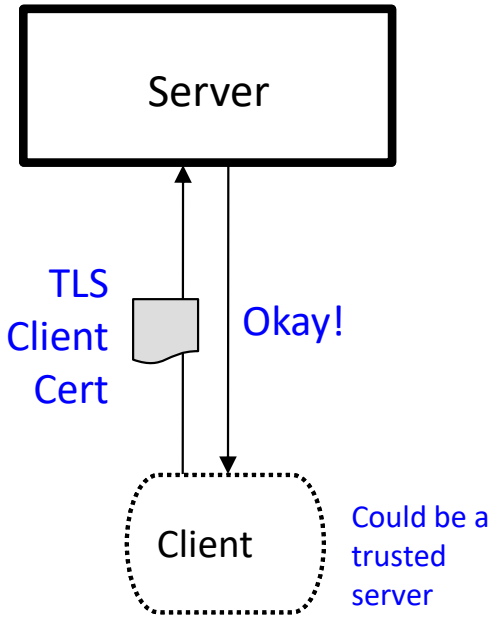


Client supplies ID/PW or ID/PassTicket

Server checks registry:

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

Client Certificate



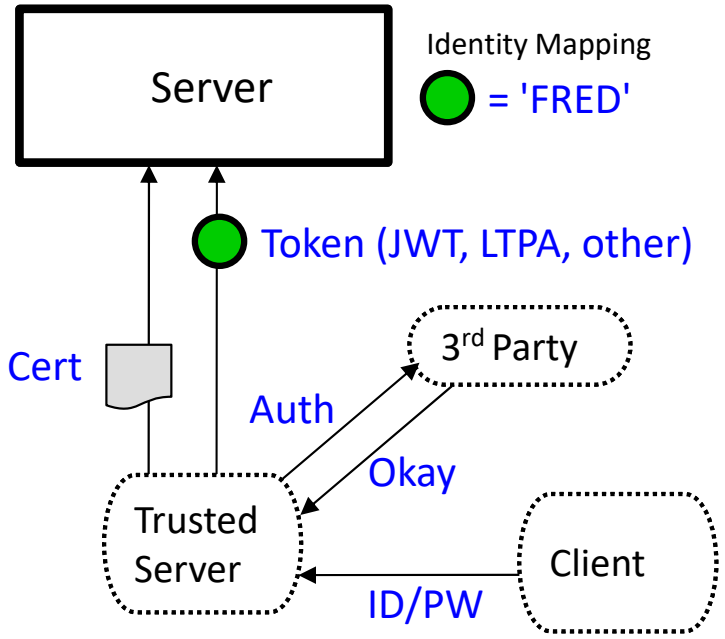
Client supplies client personal certificate

Server validates client personal certificate and maps it 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 not need to know these details.**

mitchj@us.ibm.com

z/OS Connect Security server XML Configuration

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

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

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

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

<zosconnect_zosconnect_service
  id="selectByEmployee"
  name="selectEmployee"
  requireAuth="true|false"
  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 directly access the service. This attribute is ignored when the service is invoked from an API, then only the API requireAuth attribute is relevant.

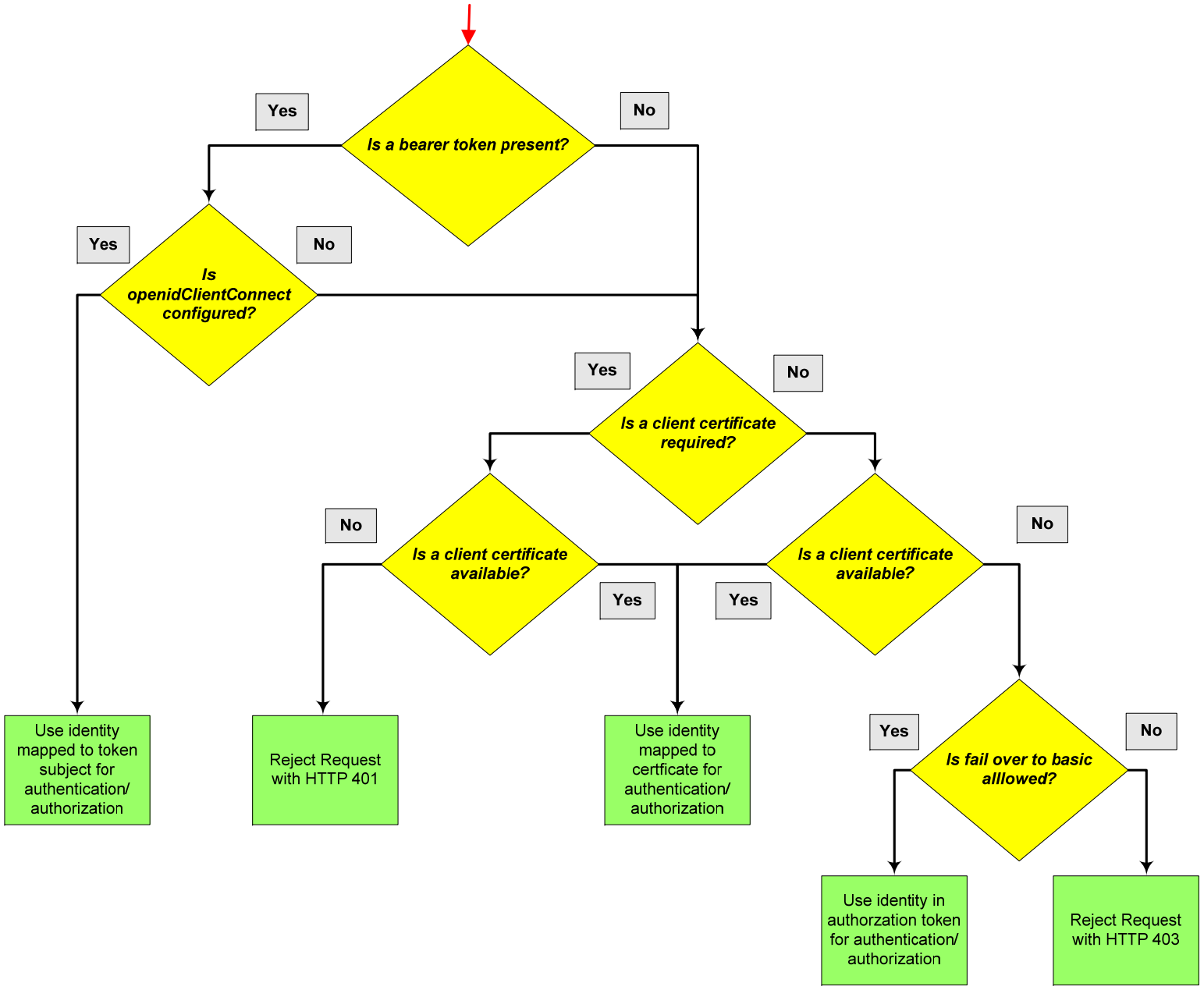
Requires that users specify security credentials to be authenticated in order to access all API requesters. If the requireAuth attribute is not set, the global setting on the zosconnect_zosConnectManager element is used instead, unless the requireAuth attribute is overridden on the specific API requester.

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

The requireAuth attribute controls whether an inbound request must provide credentials

mitchj@us.ibm.com

Authentication/Authorization Precedence

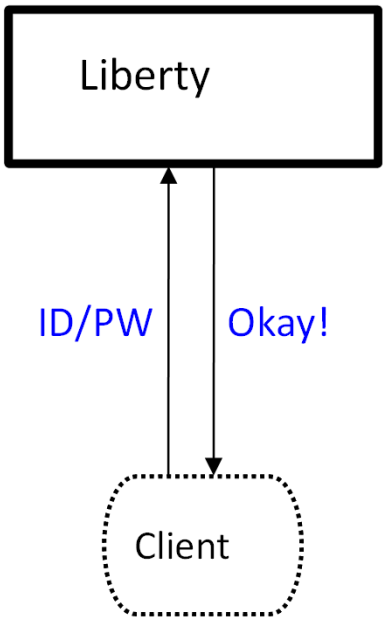


mitchj@us.ibm.com

Authentication - Basic Authentication

Several different ways this can be accomplished:

Basic Authentication

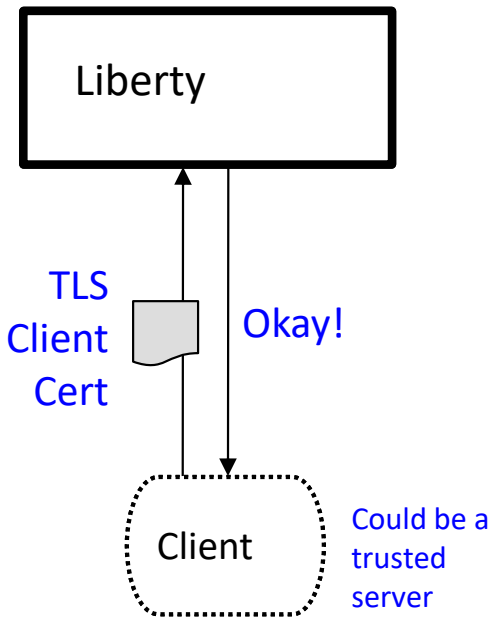


Client supplies an identity/password or an identity/PassTicket

Server checks registry:

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

Client Certificate



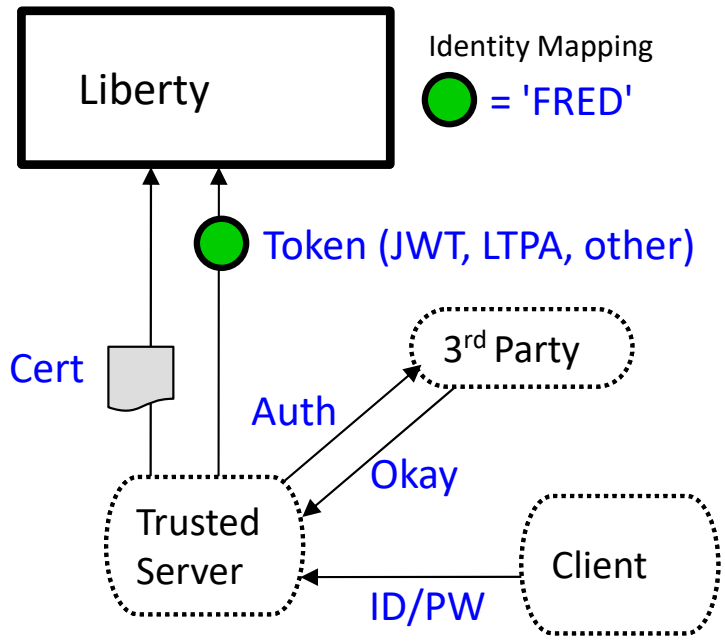
Client supplies client certificate

Server validates client certificate and maps it 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 do not need to know the details.

mitchj@us.ibm.com

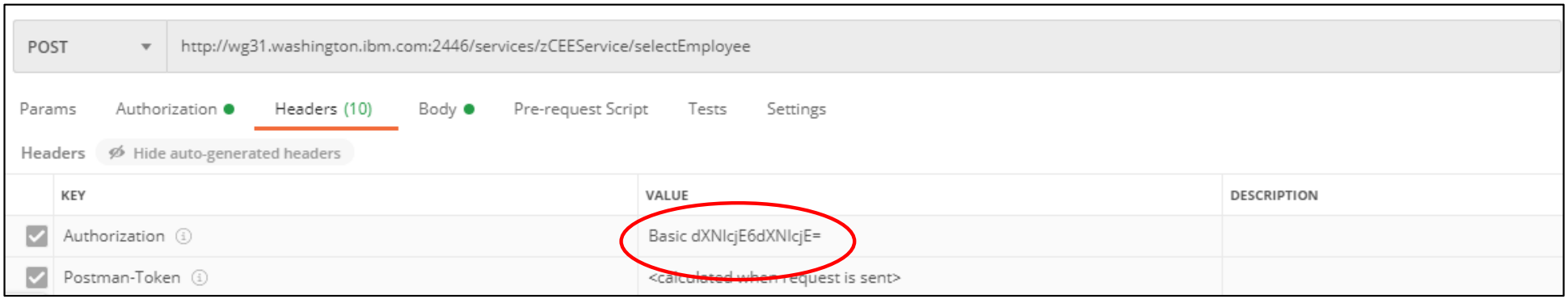
Basic authentication – REST Client

- server XML security 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.

- 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 t*oken with the identity and password encoded or formatted using Base64.
  - An example with Postman:



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

mitchj@us.ibm.com

Basic Authentication – identity and password

ZceeGet.java

```
49          URL url = new URL("https://wg31.washington.ibm.com:9453/db2/department?dept1=C01&dept2=C01");
50          System.out.println("URL: " + url);
51          HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
52          conn.setRequestMethod("GET");
53          conn.setRequestProperty("Content-Type", "application/json");
54          byte[]   bytesEncoded = Base64.encodeBase64("Fred:fredpwd".getBytes());
55          conn.addRequestProperty("Authorization", new String(bytesEncoded));
56
57          try {
58
59              if (conn.getResponseCode() != 200) {
60                  throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());
61              }
62              BufferedReader bufferReader = new BufferedReader(new InputStreamReader((conn.getInputStream())));
63              String output;
64              StringBuilder stringBuffer = new StringBuilder();
65              while ((output = bufferReader.readLine()) != null) {
66                  stringBuffer.append(output);
67              }
68              JSONObject json = new JSONObject(stringBuffer.toString());
69              JSONArray jsonArray = json.getJSONArray("ResultSet 1 Output");
70              JSONObject jsonEntry = new JSONObject();
71              for (int index = 0; index < jsonArray.length(); index++) {
72                  jsonEntry = jsonArray.getJSONObject(index);
73                  if (jsonEntry.has("employeeNumber")){
```

Problems @ Javadoc Declaration Console

```
<terminated> ZCEEGet [Java Application] C:\Program Files\IBM\Java80\jre\bin\javaw.exe (Apr 27, 2021, 4:08:52 PM)
URL: https://wg31.washington.ibm.com:9453/db2/department?dept1=C01&dept2=C01

Employee Number:  000030
First Name  :     SALLY
Last Name:        KWAN
Middle Initial:   A
Phone Number:     4738
Department:       C01

Employee Number:  000130
First Name  :     DOLORES
Last Name:        QUINTANA
Middle Initial:   M
Phone Number:     4578
Department:       C01

Employee Number:  000140
First Name  :     HEATHER
Last Name:        NICHOLLS
Middle Initial:   A
Phone Number:     1793
Department:       C01

Employee Number:  200140
First Name  :     KIM
Last Name:        NATZ
Middle Initial:   N
Phone Number:     1793
Department:       C01

StatusCode: 200
StatusDescription: Execution Successful
```


mitchj@us.ibm.com

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")
```

Note the communications stub generates the Authentication header token

- Or, provided by using a CEEROPT or CEEUOPT 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

mitchj@us.ibm.com

Tech/Tip: 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*.

mitchj@us.ibm.com

Tech/Tip: 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, e.g., BBGZDFLT
keymaskValue	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)
```


mitchj@us.ibm.com

Tech/Tip: Generating PassTickets on z/OS

- On z/OS, a COBOL user application can generate PassTickets by
  - invoking the IBM provided class *com.ibm.eserver.zos.racf.IRRPassTicket*:

PassTicket.java

```
1  package com.ibm.ats;
2
3  import com.ibm.eserver.zos.racf.IRRPassTicket;
4  import com.ibm.eserver.zos.racf.IRRPassTicketGenerationException;
5
6  public class PassTicket {
7
8       public static String getPassTicket(String userid, String appl) throws Exception {
9           // New instance of IRRPassTicket class
10          IRRPassTicket irrpt;
11          String passTicket = null;
12          try {
13            // Get new IRR PassTicket object
14            irrpt = new IRRPassTicket();
15             // Generate new PassTicket.
16            passTicket= irrpt.generate(userid,appl);
17            // Catch SAF exception
18          } catch (IRRPassTicketGenerationException irrException) {
19            System.out.println("Generation Exception caught: " + irrException);
20            irrException.printStackTrace();
21            return "NOTICKET";
22          }
23          return passTicket;
24      }
25 }
```

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

  - calling RACF service IRRSPK00:

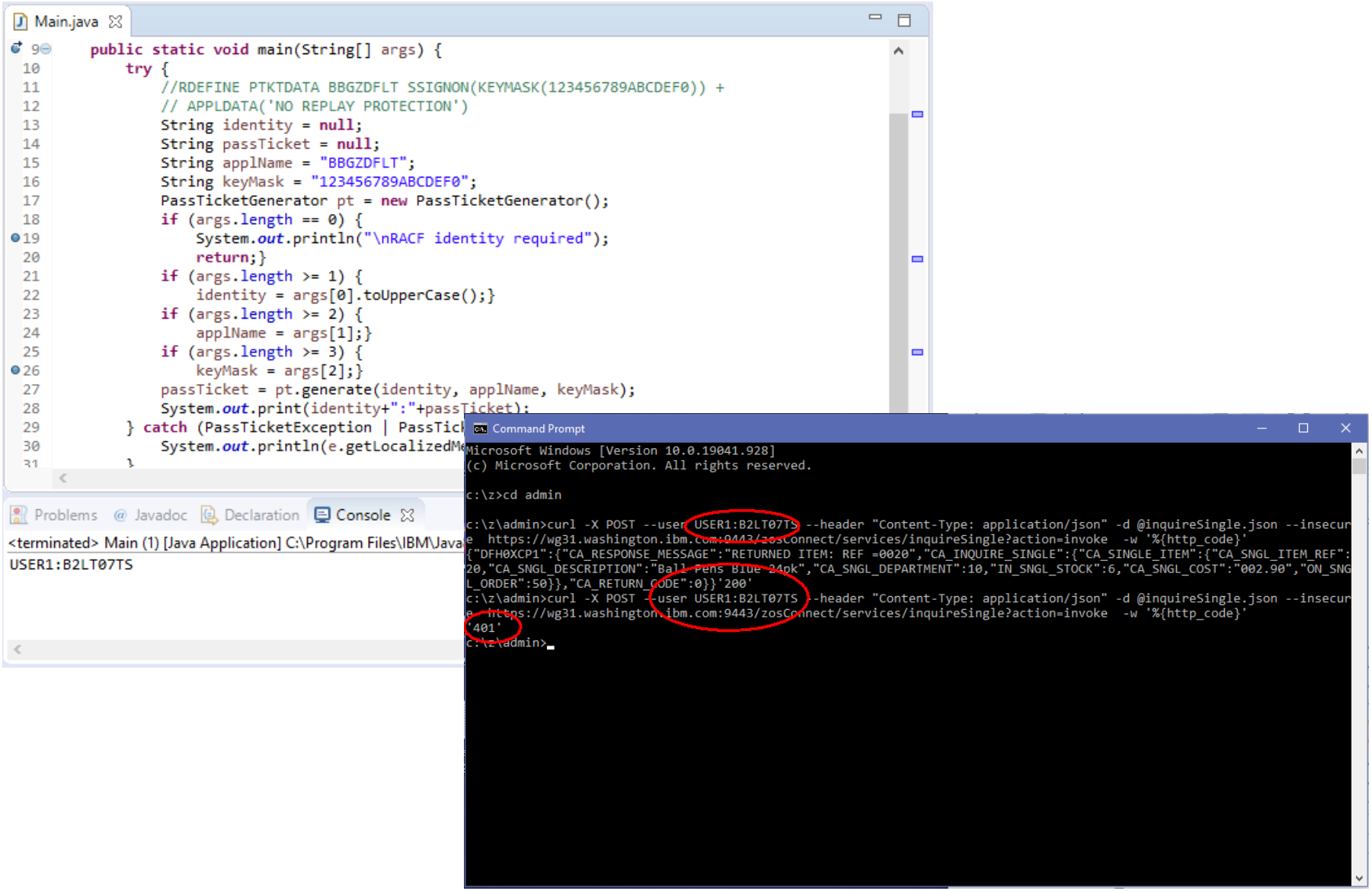
```
CALL W-IRRSPK00 USING irr-workarea,
                IRR-ALET, irr-safrc,
                IRR-ALET, irr-racfrc,
                IRR-ALET, irr-racfrsn,
                IRR-ALET, irr-functionCode,
                irr-optionWord,
                IRR-PASSTICKET,
                irr-ticketOptions-ptr,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

mitchj@us.ibm.com

Examples of both are available on request

Tech/Tip: Generating PassTickets on z/OS

- On non z/OS systems, a PassTicket can be generated in Java and other means



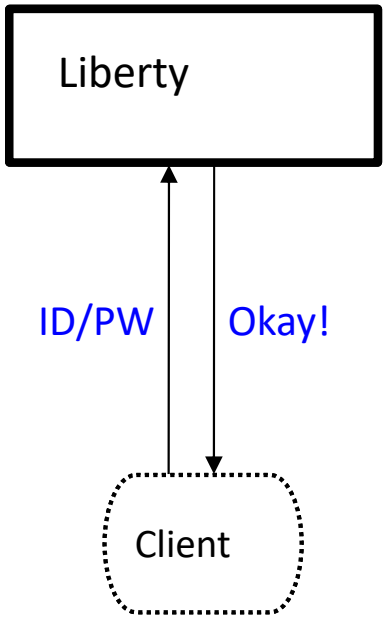
Examples of both are available

mitchj@us.ibm.com

Authentication - TLS Mutual Authentication

Several different ways this can be accomplished:

Basic Authentication



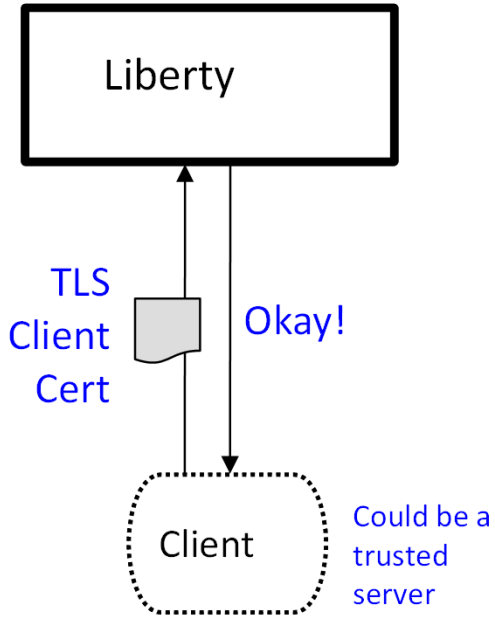
Server prompts for ID/PW

Client supplies ID/PW or ID/PassTicket

Server checks registry:

- Basic (server.xml)
- SAF

Client Certificate



Server prompts for client certificate.

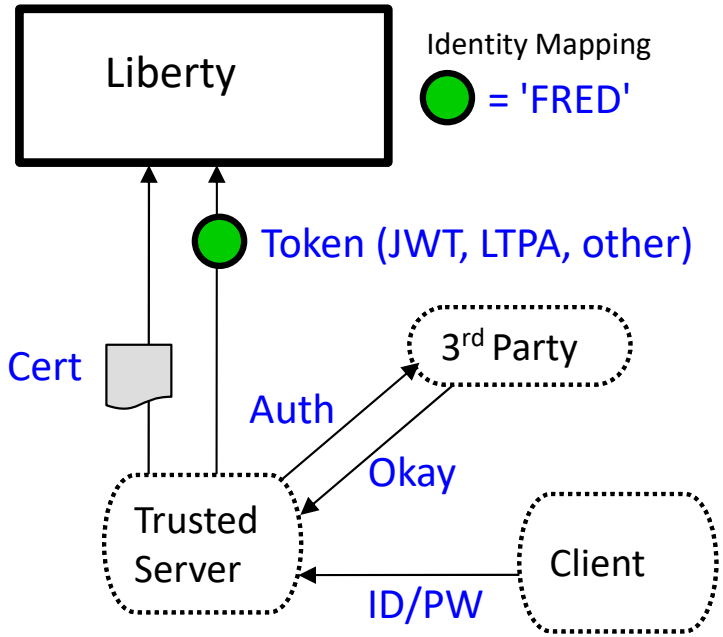
Client supplies personal certificate

Server validates client certificate and maps it 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 not need to know these details.

mitchj@us.ibm.com

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

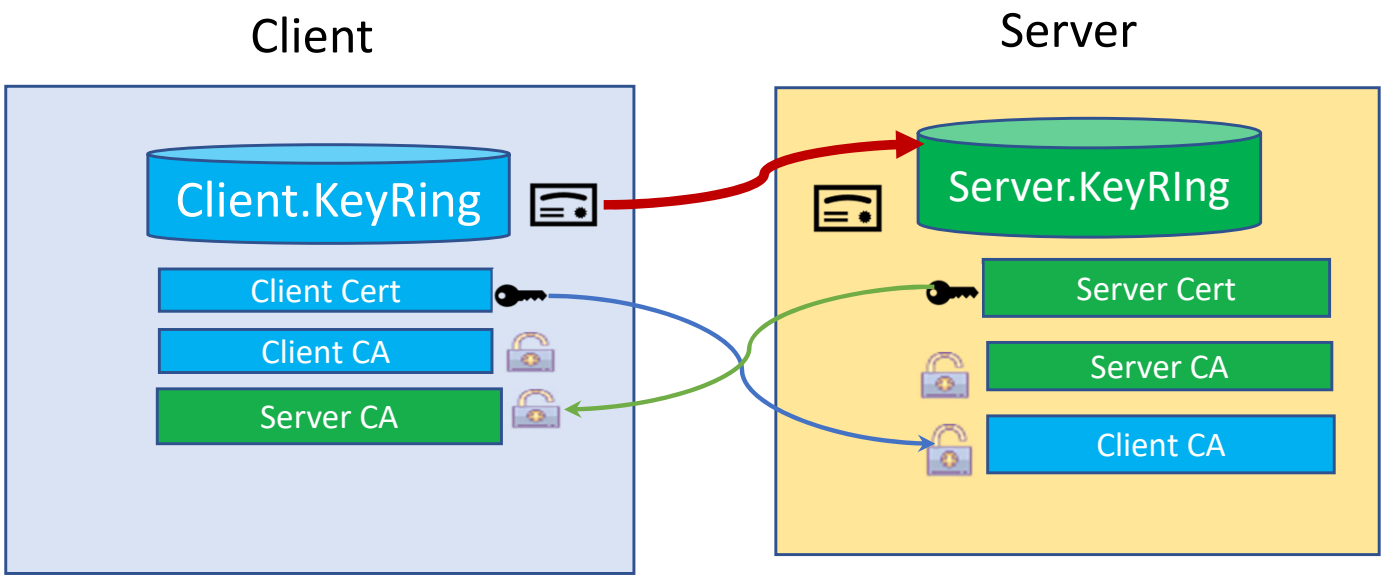
mitchj@us.ibm.com

Let's 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.

mitchj@us.ibm.com

RACF Key Rings are protected by resource profiles

- Two types of profiles are checked: **Ring Specific** or **Global**

- **Ring Specific** RDATALIB class profiles:
  - **<ring owner>.<ring name>.LST**
  - **<virtual ring owner>.IRR_VIRTUAL_KEYRING.LST**
    - **READ** access – read all certificates and own private key
    - **UPDATE** access – read other user's private keys
    - **CONTROL** access – read CA / SITE private keys
- **Global** FACILITY class profiles:
  - **IRR.DIGTCERT.LISTRING**:
    - **READ** access – read own key rings and own private keys and read SITE and CA Virtual key rings.
    - **UPDATE** access – read other user's rings (Can not read others user's private keys)
  - **IRR.DIGTCERT.GENCERT:**
    - **CONTROL** access – read CA / SITE private keys

- **Note:** Private keys are only returned when certificate usage is **PERSONAL**
- **Remember:** When switching from Global FACILITY class profiles to Ring Specific RDATALIB class profiles, the Ring specific will be checked first.

Let's explore the TLS Handshake Flow

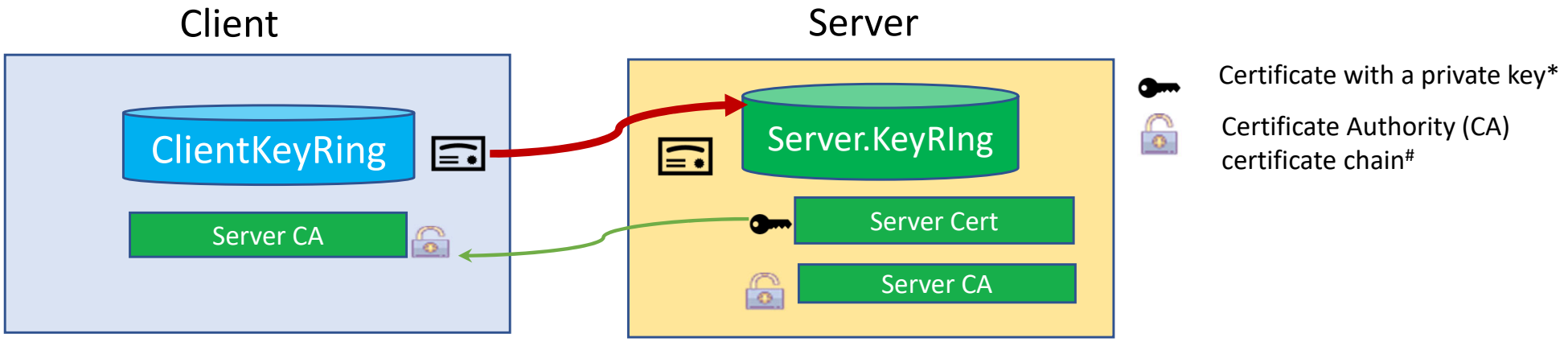
TLS handshake –

Server Authentication

❑Shared key ring

JSSE - safkeyring://STZCONID/ClientKeyRing
AT-TLS - STZCONID/ClientKeyRing

RACF RDATALIB resources
- RDEFINE RDATALIB STZCONID.ClientKeyRing.LST UACC(NONE)
- PERMIT STZCONID.ClientKeyRing.LST CLASS(RDATALIB) ID(ZCEEUSRS) ACCESS(READ)



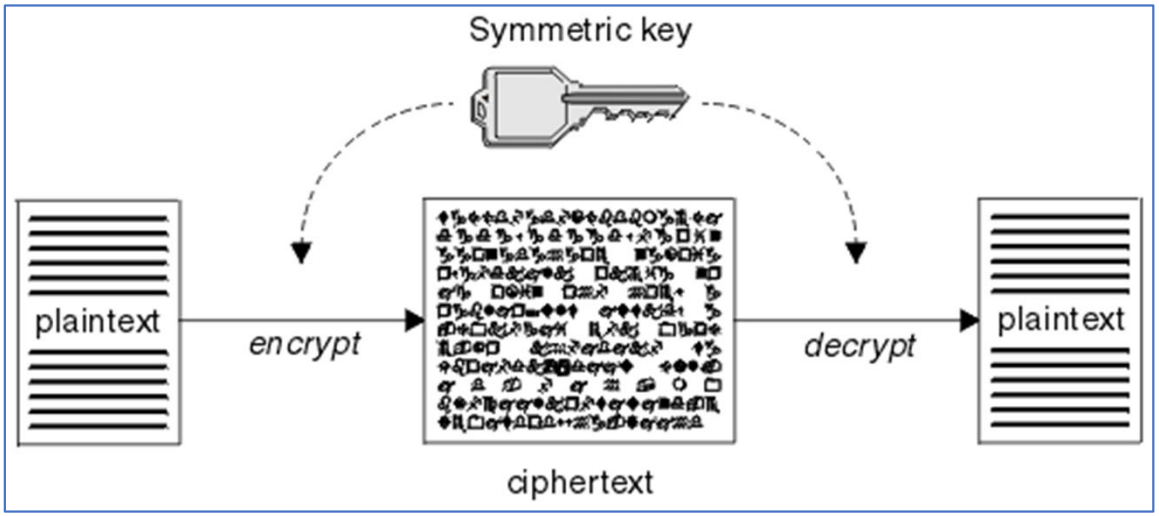
❑Shared Virtual key ring

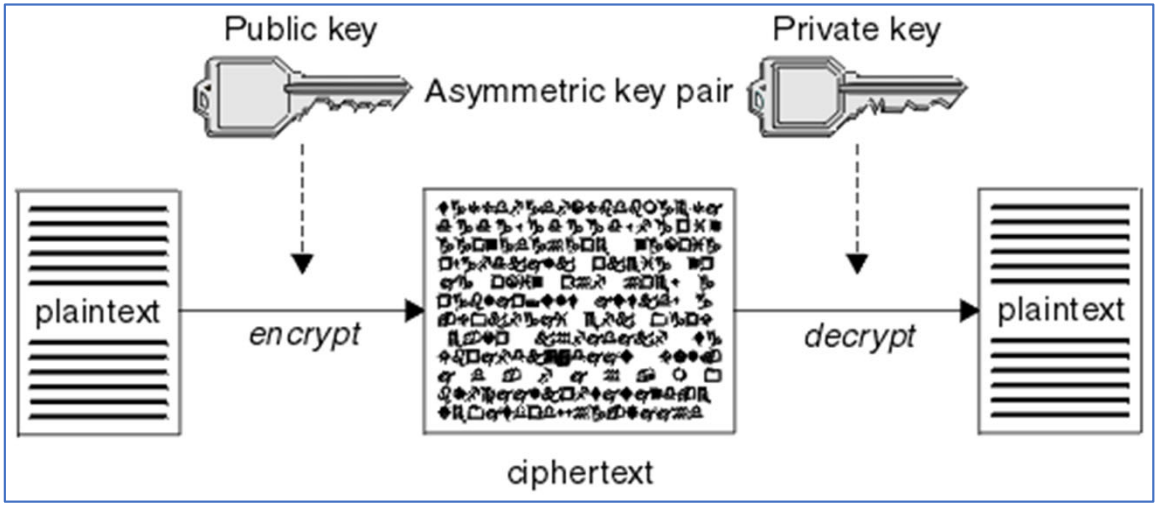
RACF RDATALIB resources
- RDEFINE RDATALIB CERTIFAUTH.IRR_VIRTUAL_KEYRING_LST UACC(NONE)
- PERMIT CERTIFAUTH.IRR_VIRTUAL_KEYRING_LST CLASS(RDATALIB) ID(ZCEEUSRS) ACCESS(READ)

JSSE - safkeyring://*AUTH*/*
AT-TLS - *AUTH*/*

mitchj@us.ibm.com

Symmetric key v. Asymmetric key pair





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

mitchj@us.ibm.com

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
```

Broadcom Support web pages

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

mitchj@us.ibm.com

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
```

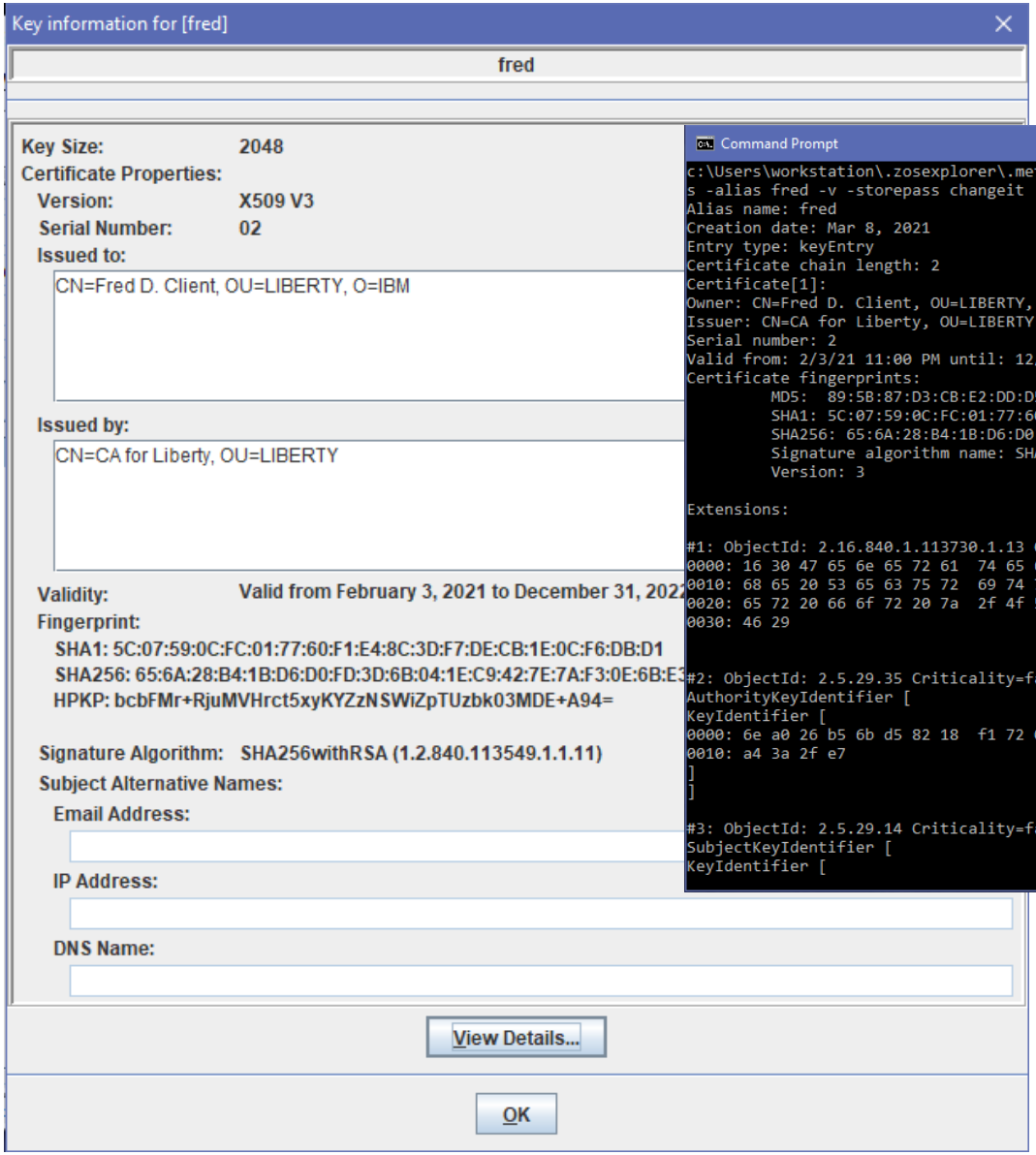

mitchj@us.ibm.com

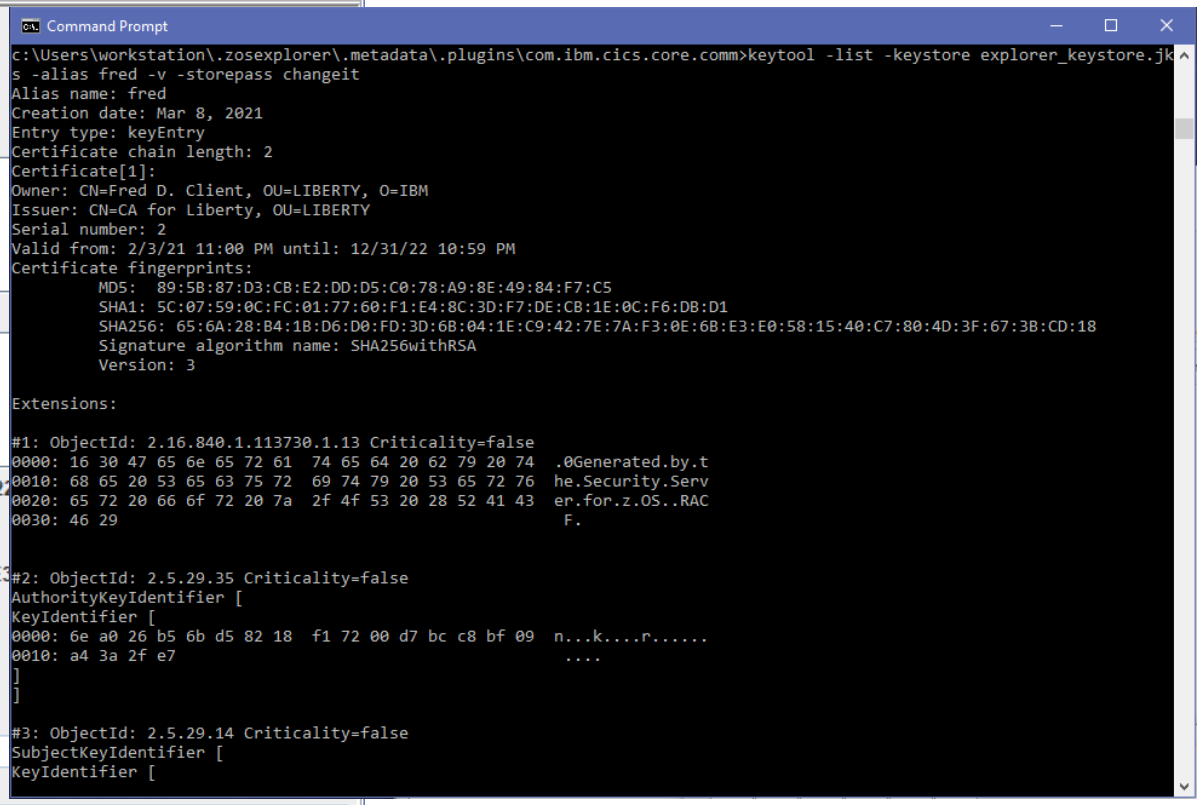
Tech/Tip: Anatomy of a RACF Personal Digital Certificate

```
Digital certificate information for user ATSSERV:

  Label: RPServer-Server
  Certificate ID: 2QfB4+Lixdnl2dfihZmlhZlg4oWZpYWZ
  Status: TRUST
  Start Date: 2020/11/12 00:00:00
  End Date:   2029/12/31 23:59:59
  Serial Number:
       >01<
  Issuer's Name:
       >CN=RPServer-CertAuth.OU=CertAuth<
  Subject's Name:
       >CN=RPServer-Server.OU=ATS.O=IBM.C=USA<
  Subject's AltNames:
    Domain: wg31.washington.ibm.com
  Signing Algorithm: sha1RSA
  Key Type: RSA
  Key Size: 2048
  Private Key: YES
  Ring Associations:
    Ring Owner: ATSSERV
    Ring:
       >RpServer.KeyRing<
    Ring Owner: LIBSERV
    Ring:
       >RpServer.KeyRing<
```


mitchj@us.ibm.com

Tech/Tip: Anatomy of a certificate – IkeyMan/keytool

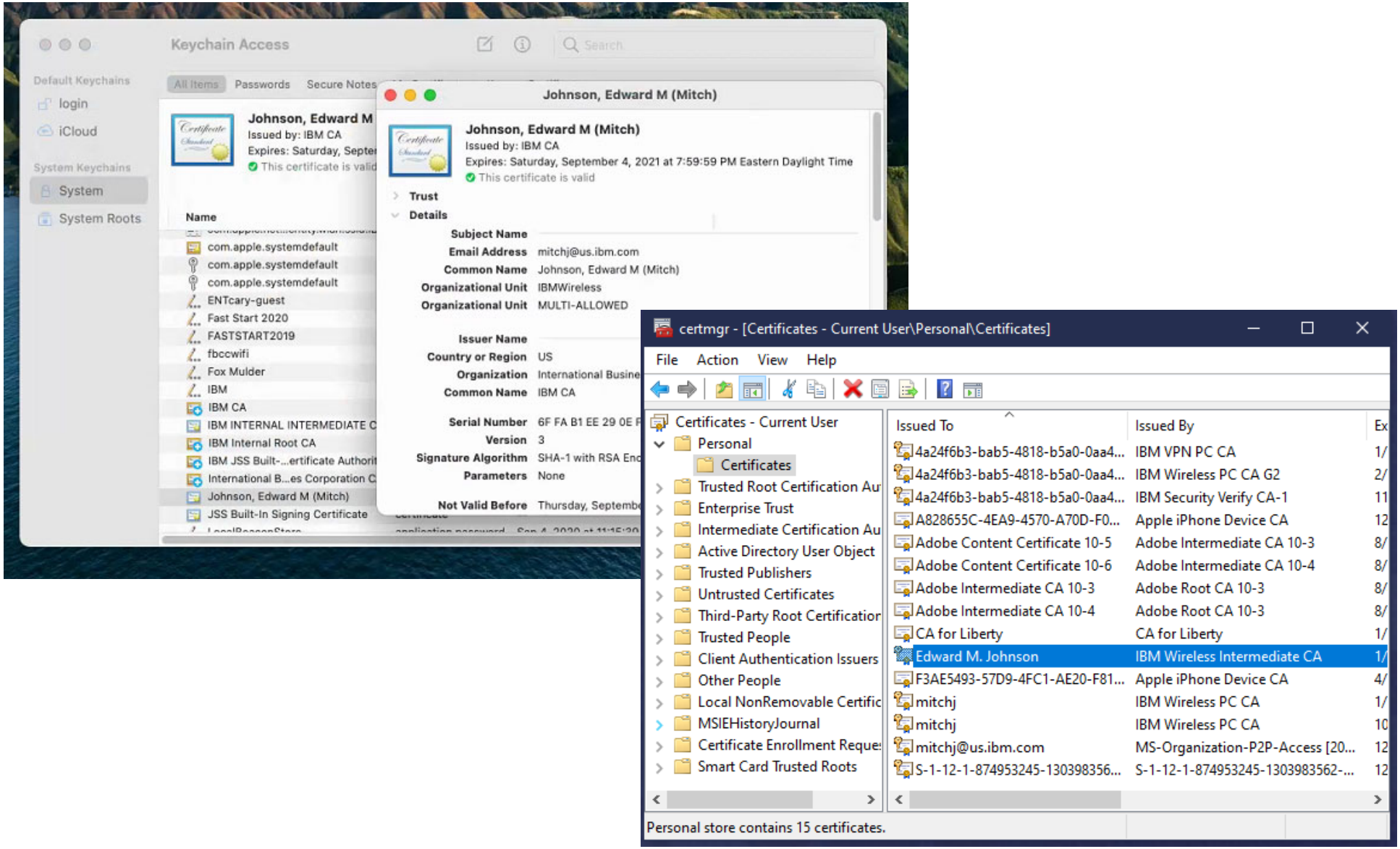




Both can be found in /usr/lpp/java/J8.0_64/bin or c:/Program Files/IBM/Java80/jre/bin

mitchj@us.ibm.com

Tech/Tip: Anatomy of a certificate – Mac/Windows

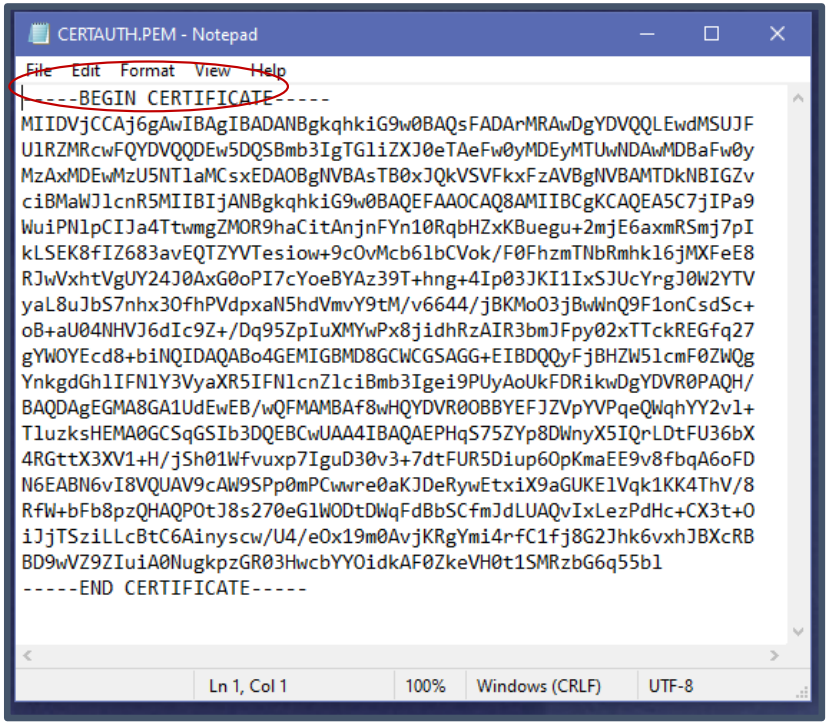


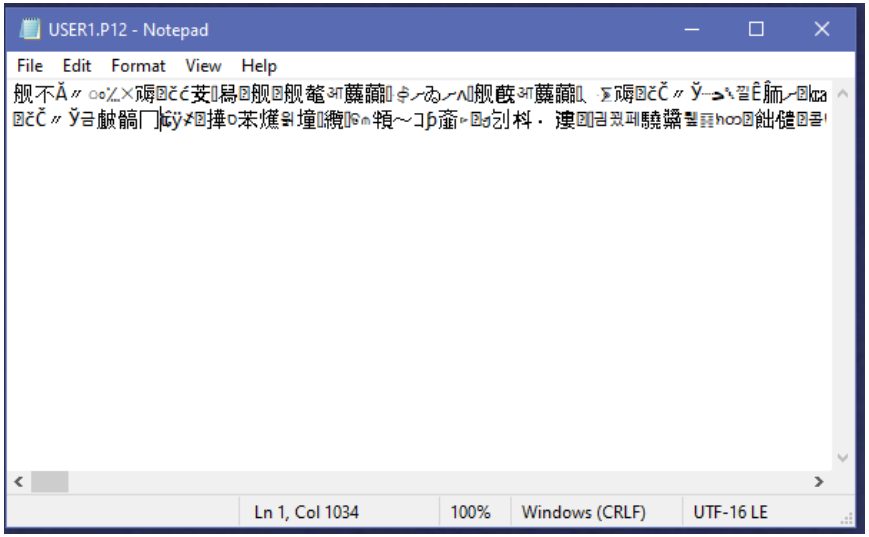
mitchj@us.ibm.com

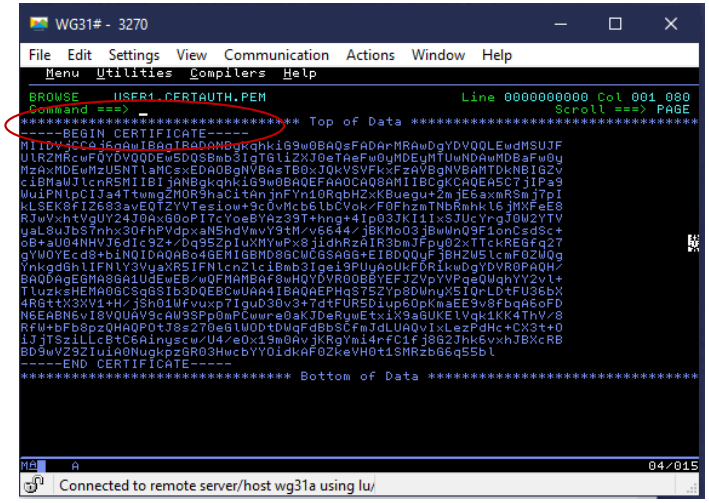
Tech/Tip: How to tell the difference?

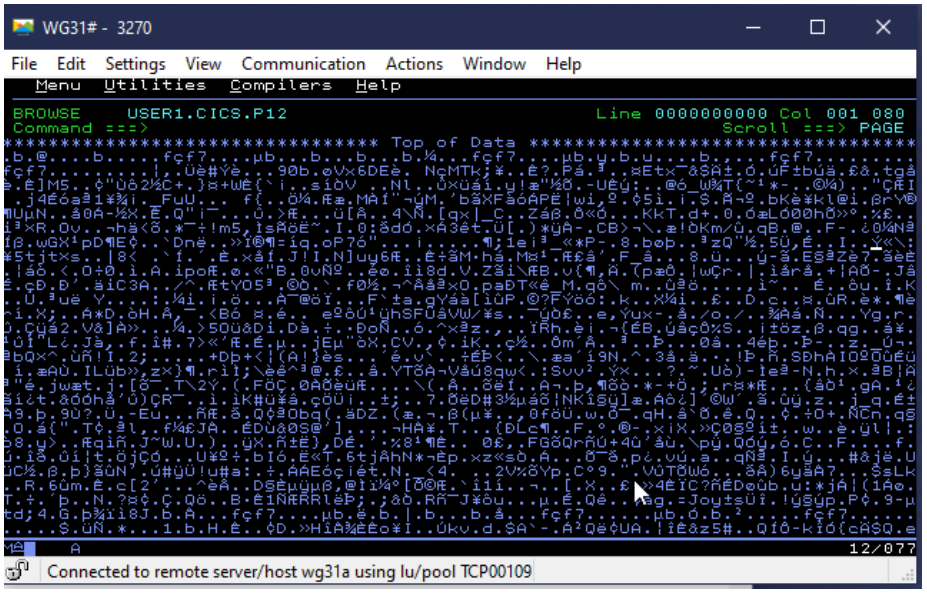
Public certificate (site or certificate authority or certificate request)

Personal certificate with private key









mitchj@us.ibm.com

Tech/Tip: Combining TLS mutual 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:9445/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

mitchj@us.ibm.com

Tech/Tip: Combining TLS and pass tickets authentication

```
//EXPORT EXPORT SYMLIST=(*)
// SET CURL='/usr/lpp/rocket/curl'
// SET USERCRED='FRED:Z0HWKDSF'
// SET CACERT='/u/johnson/zceesrvr.pem'
//IKJEFT01 PROC
//IKJEFT01 EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSERR   DD SYSOUT=*
//STDOUT   DD SYSOUT=*
// PEND
//*************************************************************
//*  Copy a CA certificate to OMVS
//*************************************************************
//COPYPEM EXEC IKJEFT01
//INDD     DD DISP=SHR,DSN=USER1.CERTAUTH.PEM
//OUTDD    DD PATH='&CACERT',PATHOPTS=(ORDWR,OCREAT)
//SYSTSIN  DD *,SYMBOLS=EXECSYS
OCOPY INDD(INDD) OUTDD(OUTDD) FROM1047 CONVERT((BPXFX311))
//*************************************************************
//*  Use cURL to invoke an administrative API
//*************************************************************
//INVOKE EXEC IKJEFT01
//SYSTSIN  DD *,SYMBOLS=EXECSYS
BPXBATCH SH export CURL=&CURL; +
$CURL/bin/curl -X GET  +
--cacert &CACERT --user &USERCRED +
https://wg31.washington.ibm.com:9445/zosConnect/apis
//*************************************************************
//*  Use cURL to invoke an administrative API
//*************************************************************
//INVOKE EXEC IKJEFT01
//SYSTSIN  DD *,SYMBOLS=EXECSYS
BPXBATCH SH export CURL=&CURL; +
$CURL/bin/curl -X GET  +
--cacert &CACERT --user &USERCRED +
https://wg31.washington.ibm.com:9445/cscvinc/employee/000100
```

Consider using a program that will customize template JCL and then submit it for execution

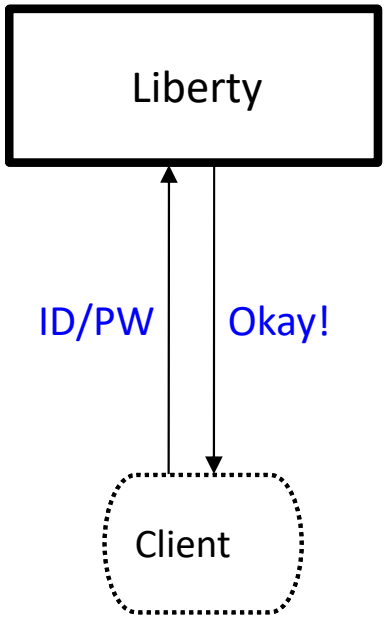
```
//BUILDJCL EXEC PGM=BUILDJCL,REGION=0M
//SYSPRINT DD SYSOUT=*
//INJCL    DD DISP=SHR,DSN=JOHNSON.JCLLIB.CNTL(CURLSKEL)
//INTRDR   DD SYSOUT=(A,INTRDR)
```

mitchj@us.ibm.com

Authentication - Third Party Authentication

Several different ways this can be accomplished:

Basic Authentication



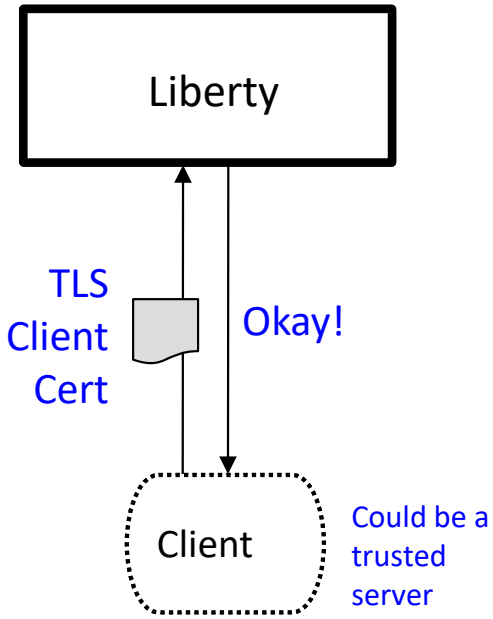
Server prompts for ID/PW

Client supplies ID/PW or ID/PassTicket

Server checks registry:

- Basic (server.xml)
- SAF

Client Certificate



Server prompts for client certificate.

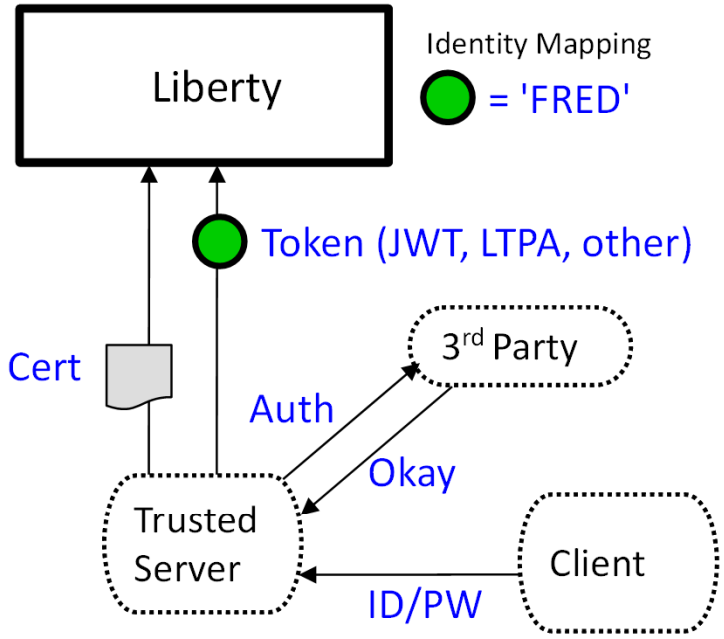
Client supplies certificate

Server validates client certificate and maps to an identity

Registry options:

- SAF

Third Party Authentication



Client authenticates to 3rd party sever

Client receives a trusted 3rd party token

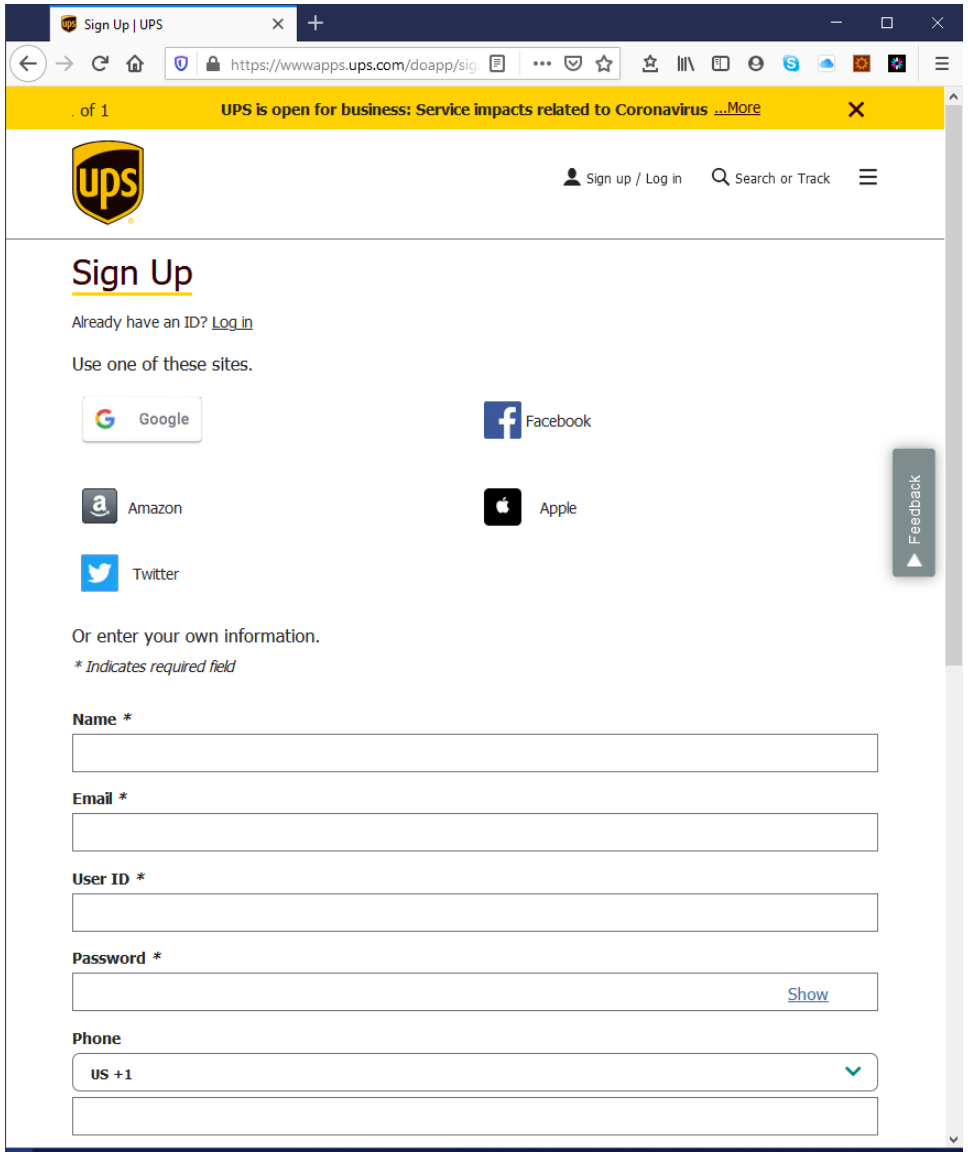
Token flows to Liberty z/OS and is mapped to an identity

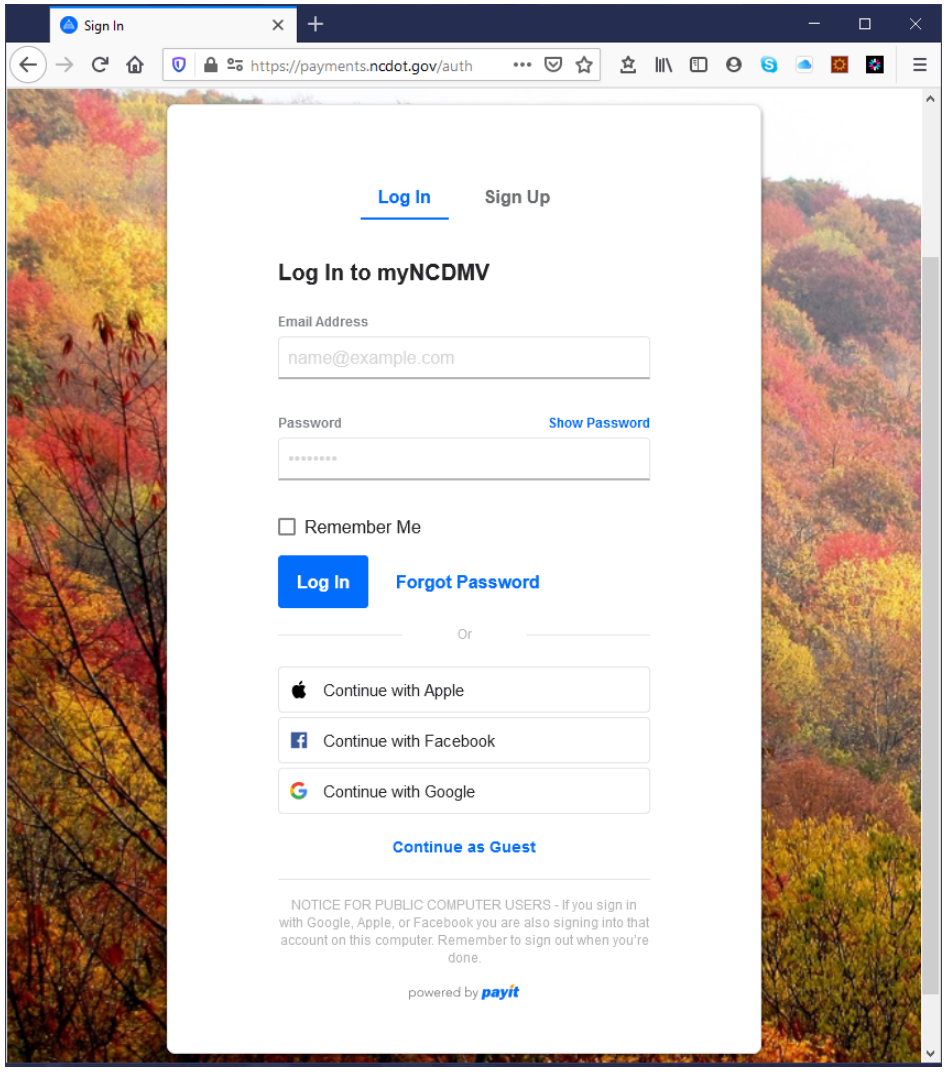
Registry options:

- **We may know these detail.**

mitchj@us.ibm.com

Third Party Authentication Examples





mitchj@us.ibm.com

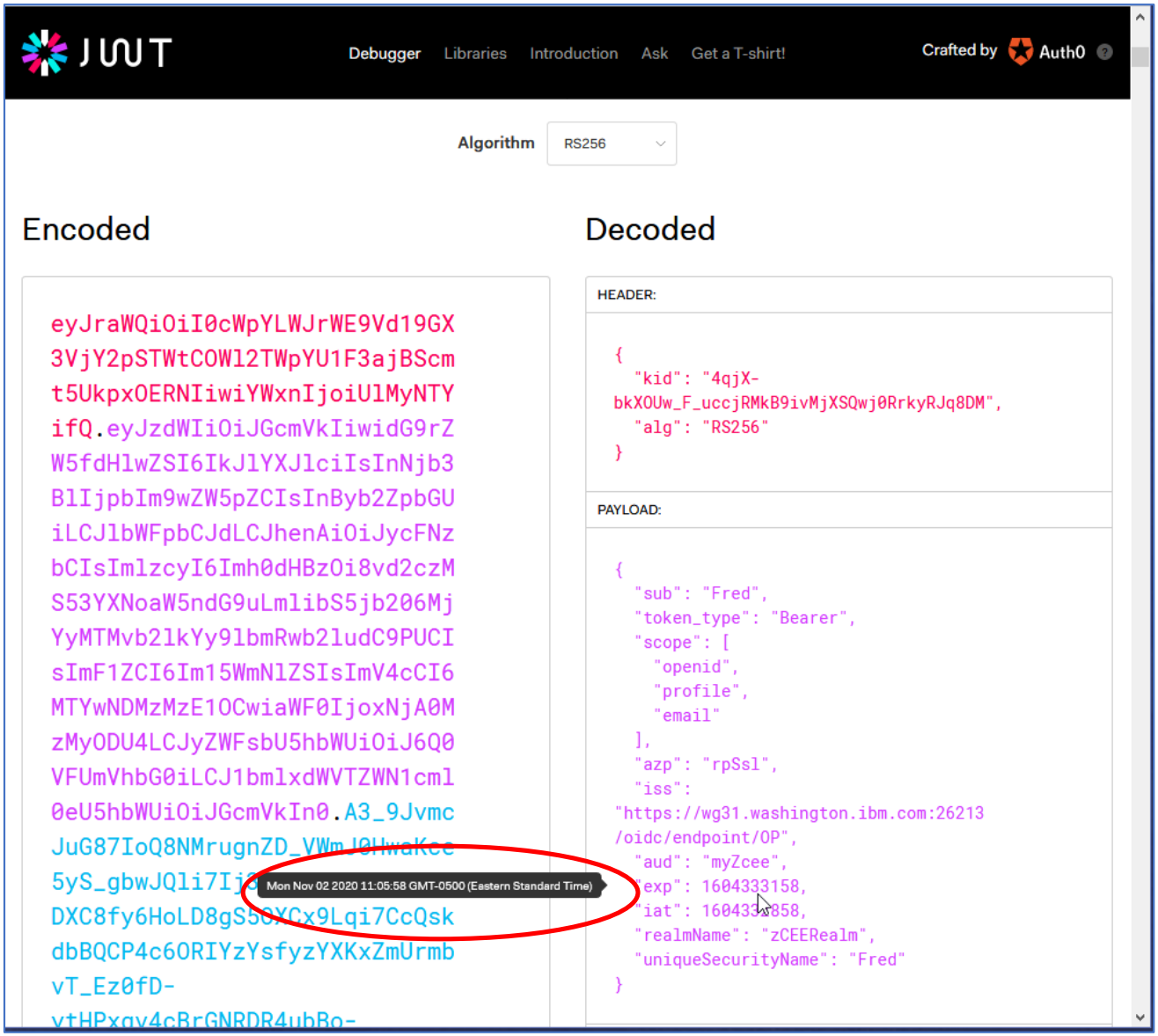
Security token types by Liberty

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	

mitchj@us.ibm.com

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



Values derived from the OAUTH configuration:

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

https://jwt.io

mitchj@us.ibm.com

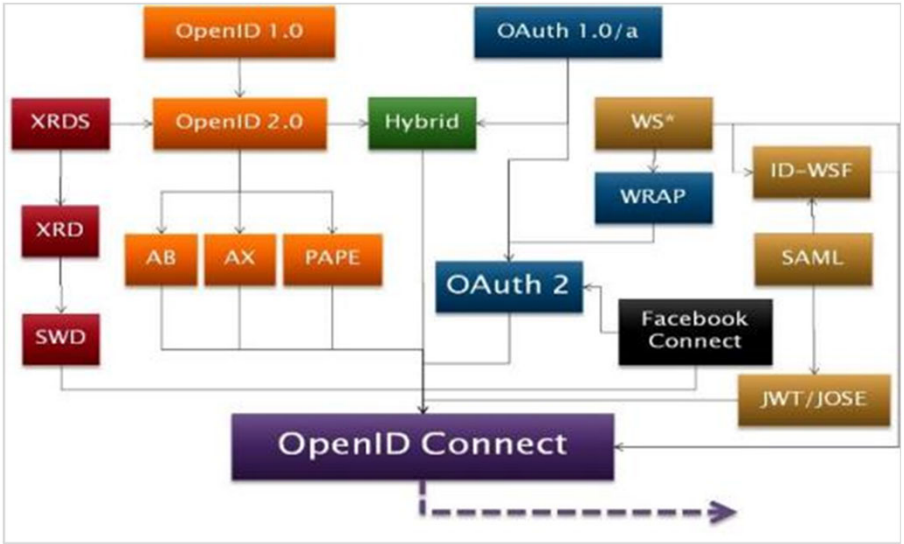
Open security standards

- **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:

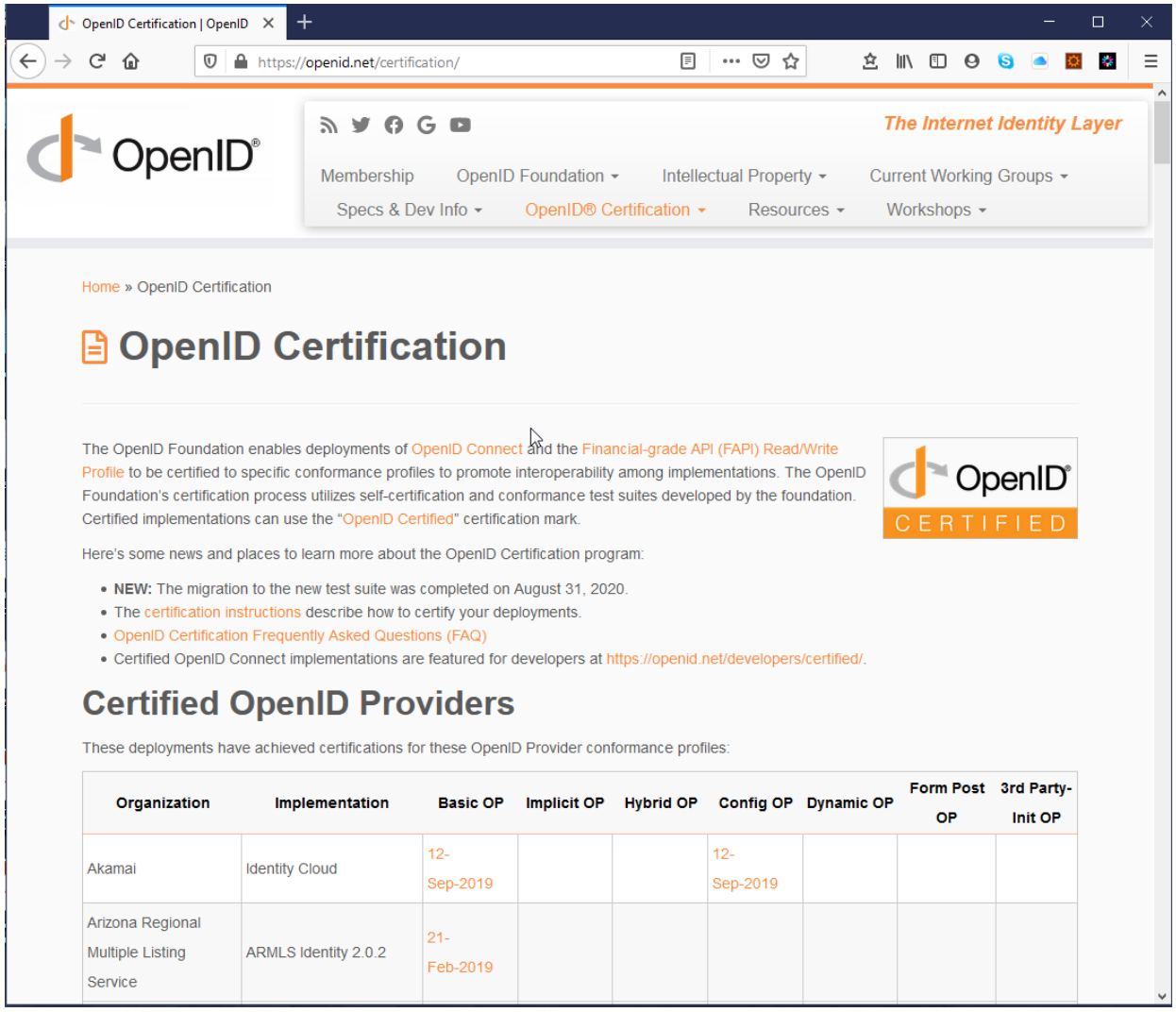
OAuth 2.0 and OpenID Connect (in plain English)
https://www.youtube.com/watch?v=996OiexHze0

OpenID Connect on Liberty
https://www.youtube.com/watch?v=fuajCS5bG4c



mitchj@us.ibm.com

There are a multitude of OpenID Certified Providers



https://openid.net/certification/

mitchj@us.ibm.com

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

mitchj@us.ibm.com

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.

mitchj@us.ibm.com

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.*

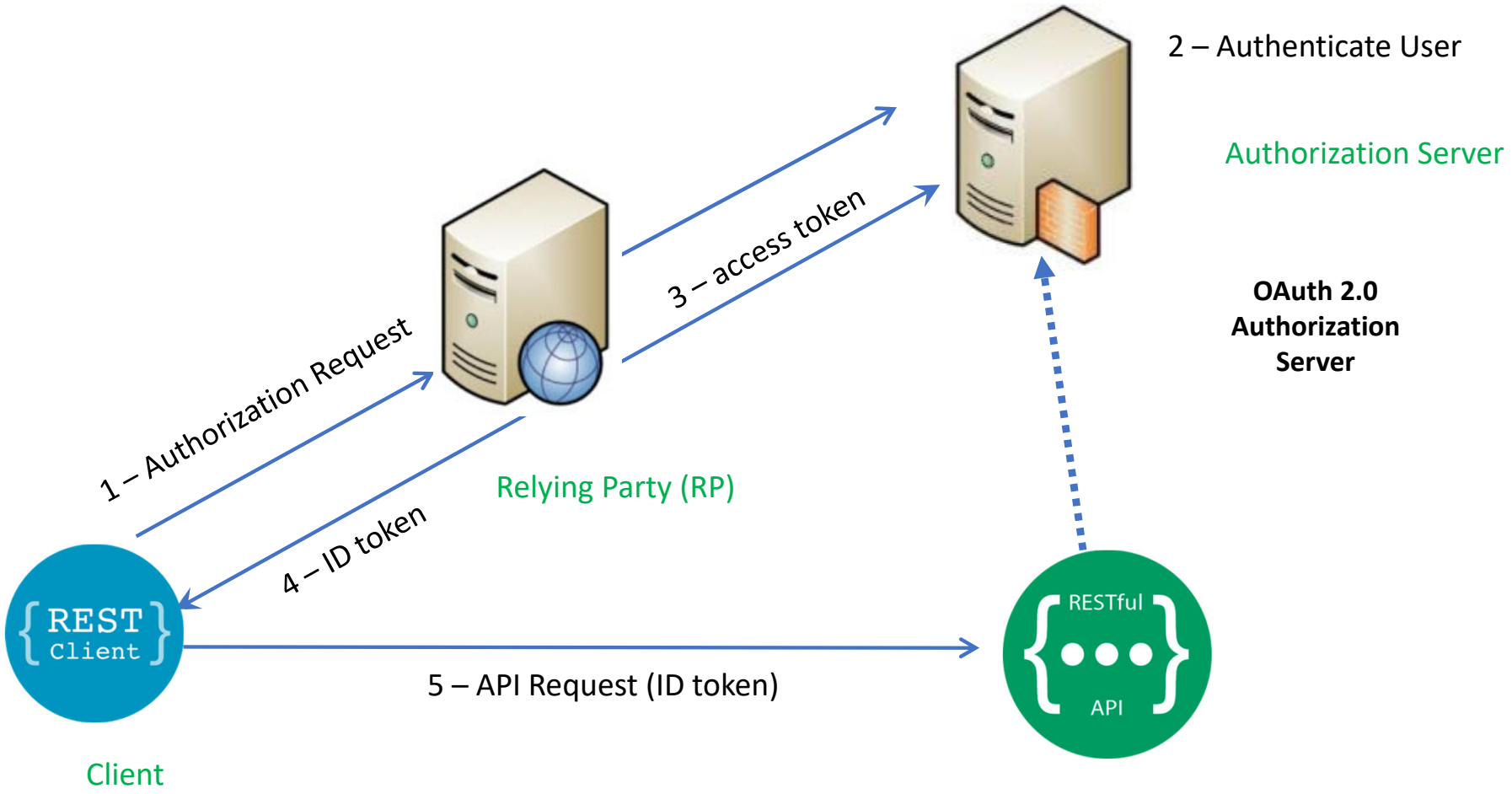
- ***Resource owner*** - An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end user. *In a z/OS Connect EE API requester scenario, the resource owner might be the user of the CICS, IMS, or z/OS application.*

- ***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.

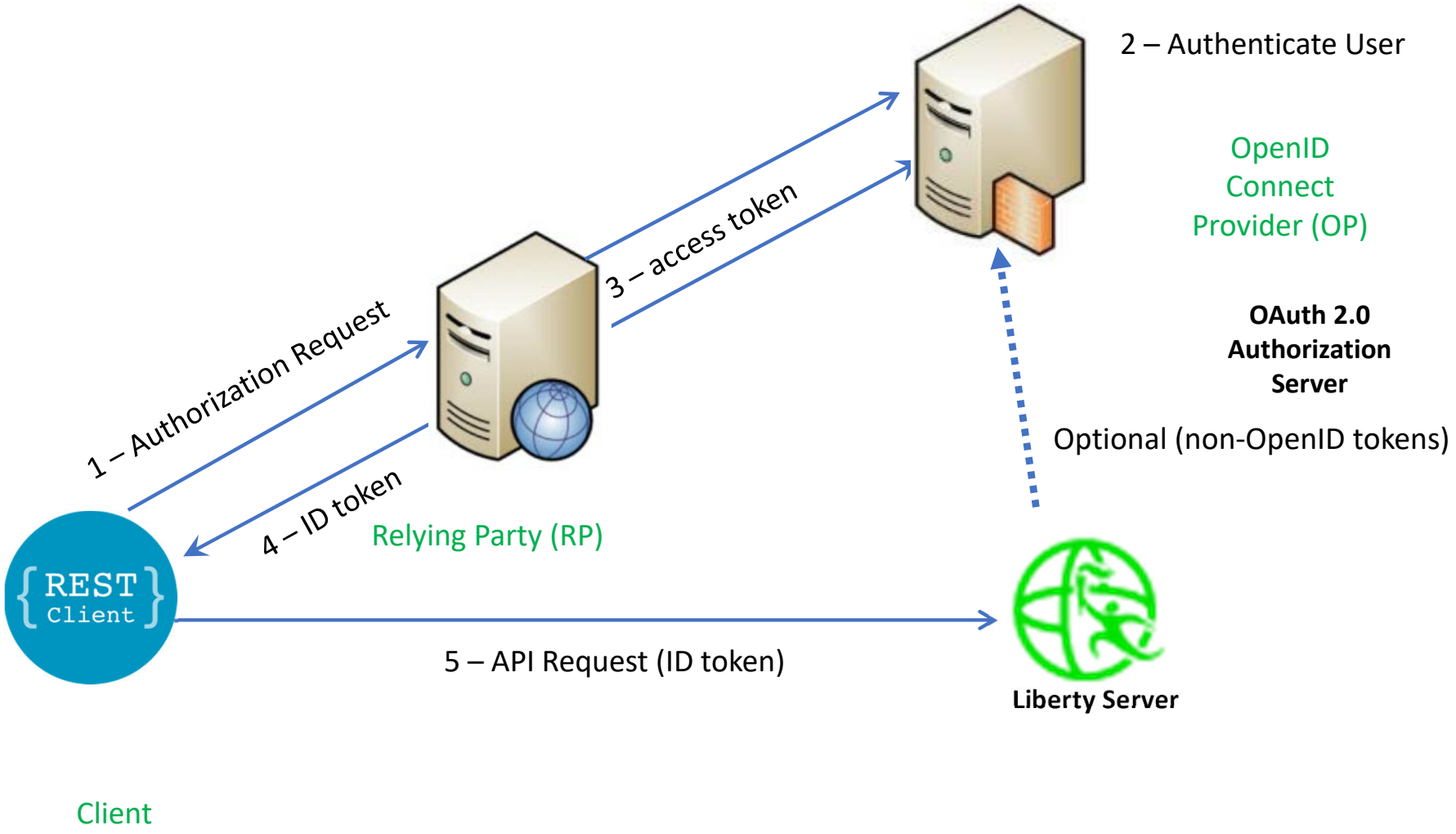
mitchj@us.ibm.com

Typical OAuth Authorization Flow



mitchj@us.ibm.com

Typical Authorization Flow for am OpenID Connect token to a z/OS Connect API Provider



mitchj@us.ibm.com

Tech/Tip: Let's explore a Liberty OpendID Provider flow

```
<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.

mitchj@us.ibm.com

Tech/Tip: Generating a JWT using Liberty as the OP

```
  <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')
```

mitchj@us.ibm.com

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:
  >distributedUser1<
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('*'))
```


mitchj@us.ibm.com

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.

mitchj@us.ibm.com

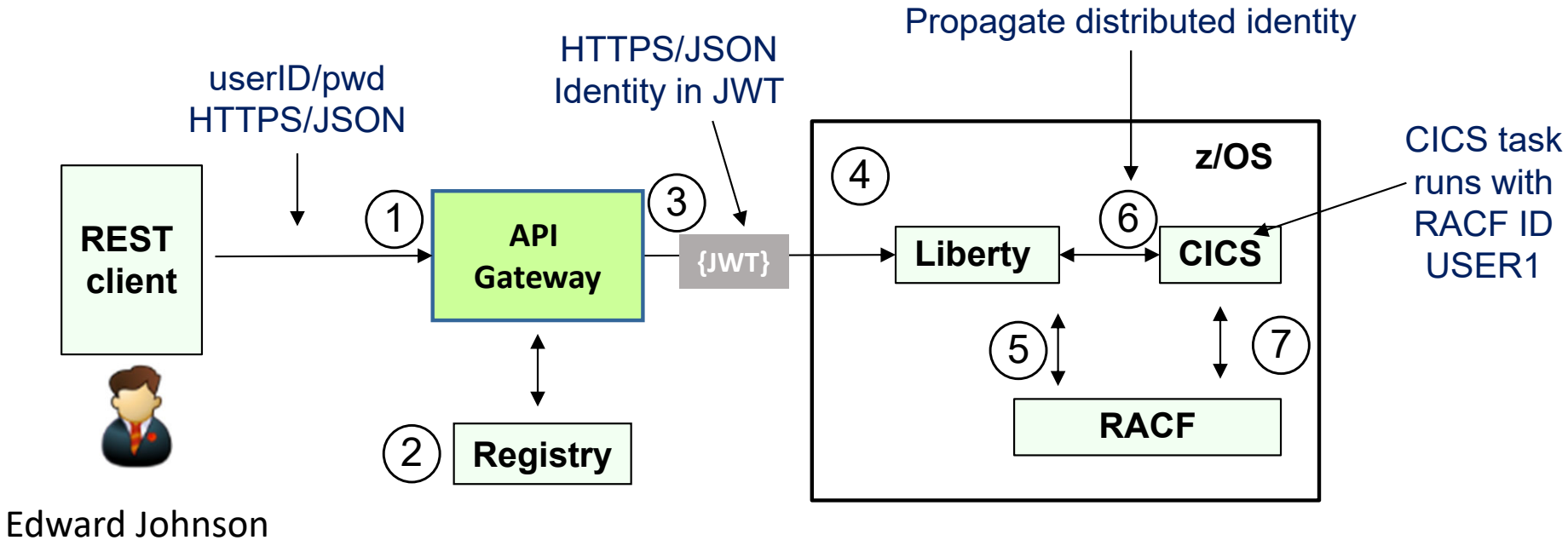
Liberty/zCEE OpenID Client identity mapping configuration attributes (JWK)

```
{
  "kid": "574eafad-fcb5-412e-97a3-8100a1c1fa5b",
  "alg": "RS256"
}
{
  "sub": "mitchj",
  "aud": "myZCEE",
  "iss": "https://wg31.washington.ibm.com:26213/oidc/endpoint/OP",
  "exp": 1610451176,
  "iat": 1610451876
}
```

```
<openidConnectClient
    id="ATSJWK"
    clientId="RS-JWT-ZCEE"
    authFilterRef="jwkAuthFilter"
    inboundPropagation="required"
    signatureAlgorithm="RS256"
    userIdentifier="sub"
    mapIdentityToRegistryUser="true"
    issuerIdentifier=https://wg31.washington.ibm.com:26213/oidc/endpoint/OP
    disableLtpaCookie="true"
    audiences="myZcee"
    jwkEndpointUrl=https://wg31.washington.ibm.com:26213/oidc/endpoint/OP/jwk
    jwkClientId="jwtClient"
    jwkSecret="jwtSecret"/>
  </openidConnectClient>
```


mitchj@us.ibm.com

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. Liberty validates JWT
5. Liberty calls RACF to map distributed ID to RACF user ID and authorizes access to API
6. 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

mitchj@us.ibm.com

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

mitchj@us.ibm.com

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

mitchj@us.ibm.com

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.

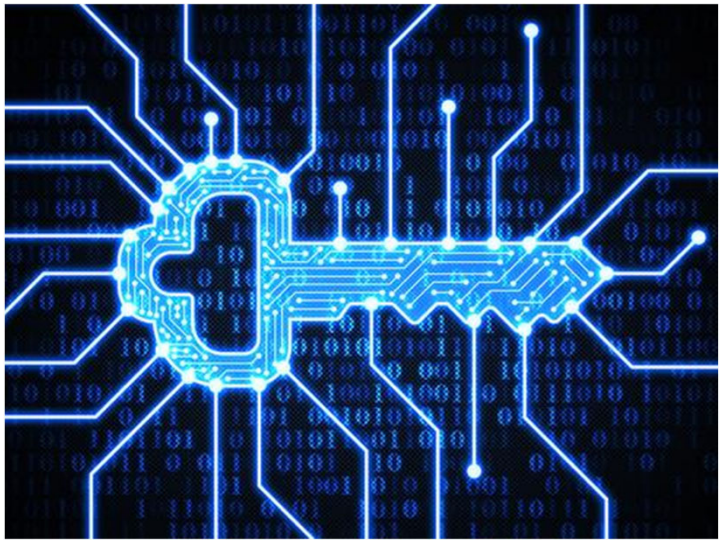
mitchj@us.ibm.com

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.



mitchj@us.ibm.com

z/OS Connect Security server XML Configuration

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

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

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

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

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

Globally, requires that inbound request using HTTPS in order to access APIs, services and API requesters, unless overridden on the specific resource definitions.

Requires that inbound request use HTTPS in order to access the API.

Requires that inbound request use HTTPS when directly accessing this service.

Requires that all inbound request for this API requester use HTTPS..

Requires that inbound request for this service use HTTPS when directly accessing this service.

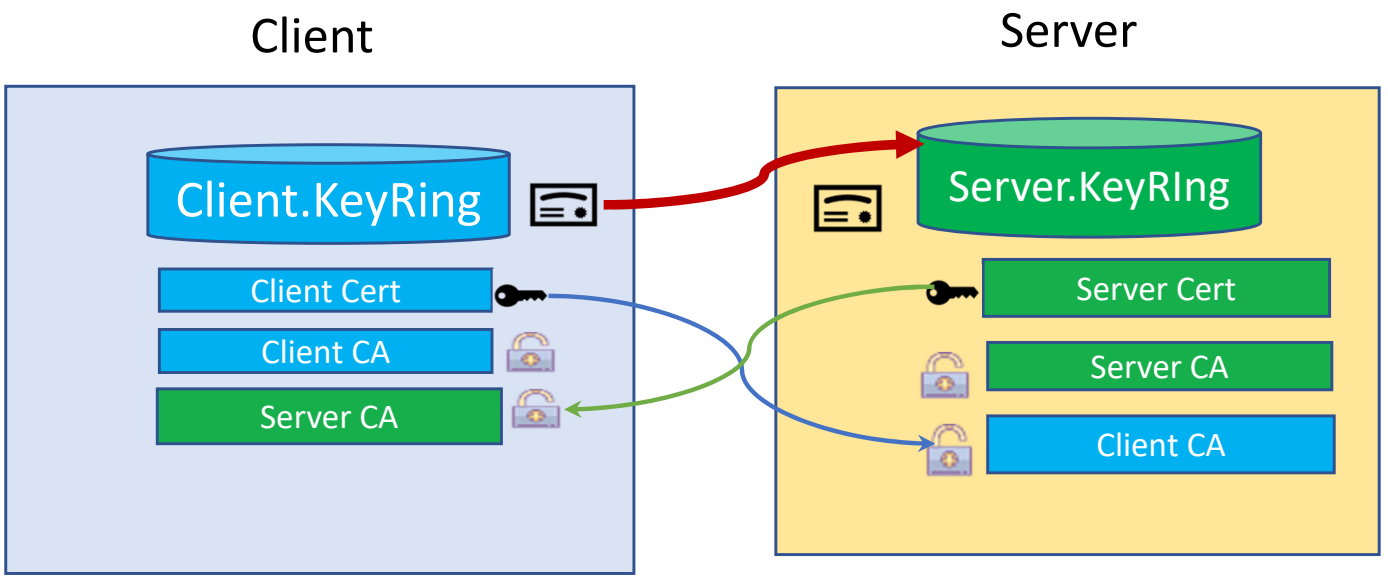
requireSecure controls inbound TLS connections

mitchj@us.ibm.com

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.

mitchj@us.ibm.com

Tech/Tip: cURL trace of a TLS Handshake

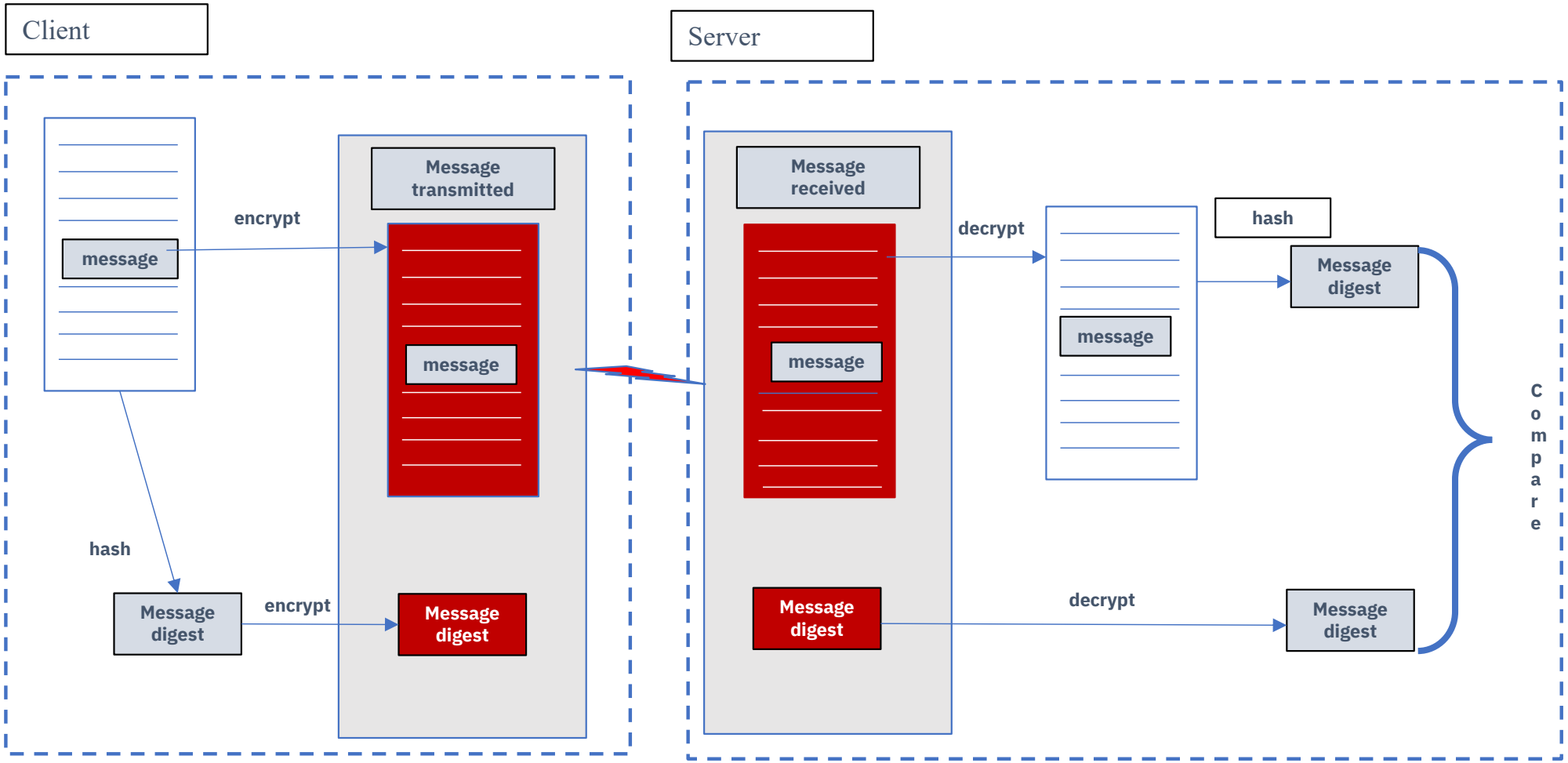
```
* successfully set certificate verify locations:
*   CAfile: certauth.pem
  CApath: none
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (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

mitchj@us.ibm.com

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.

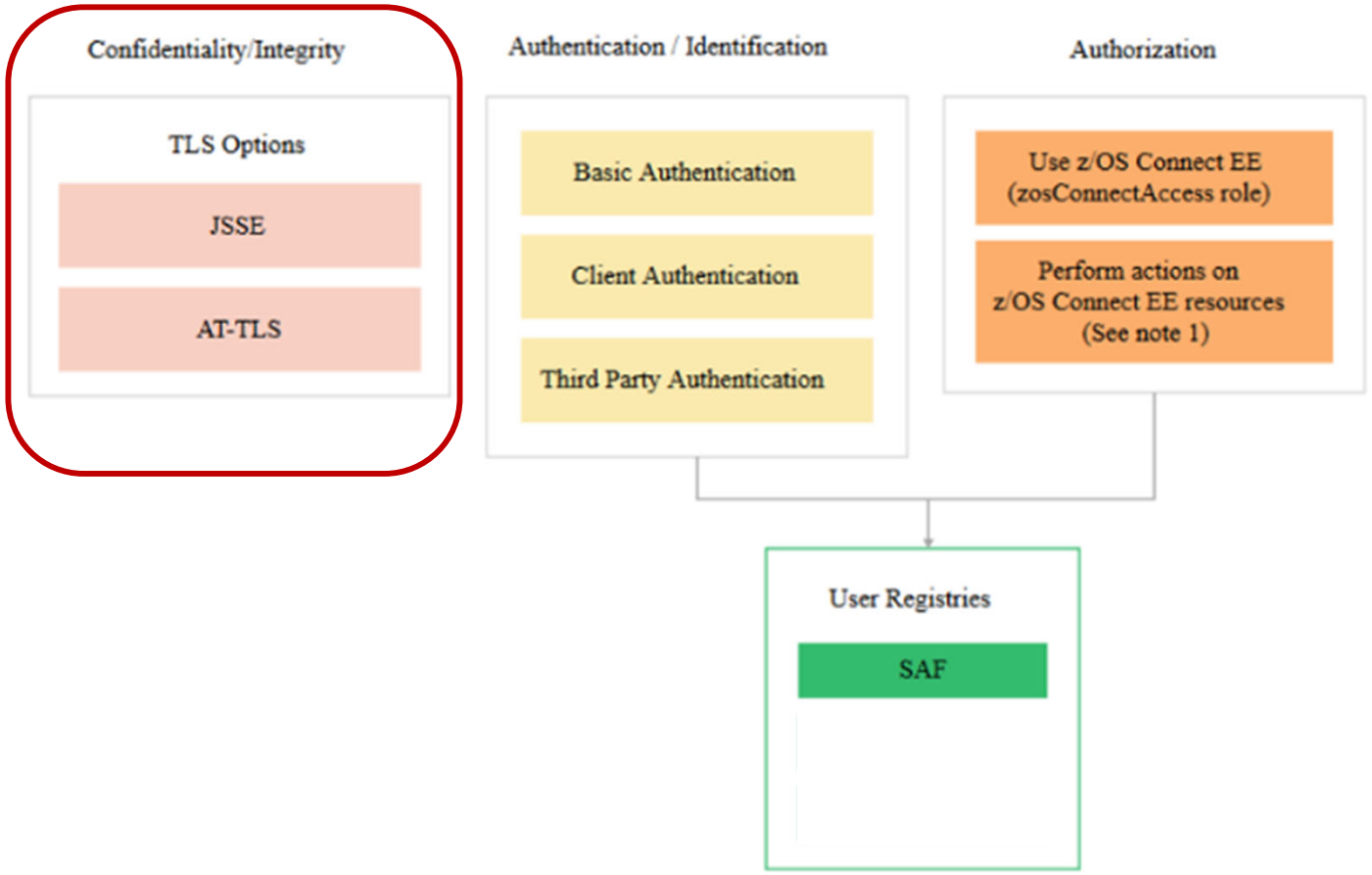
Table 1. CipherSpecs supported by IBM MQ and their equivalent CipherSuites

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

mitchj@us.ibm.com

Liberty and z/OS Connect EE security options

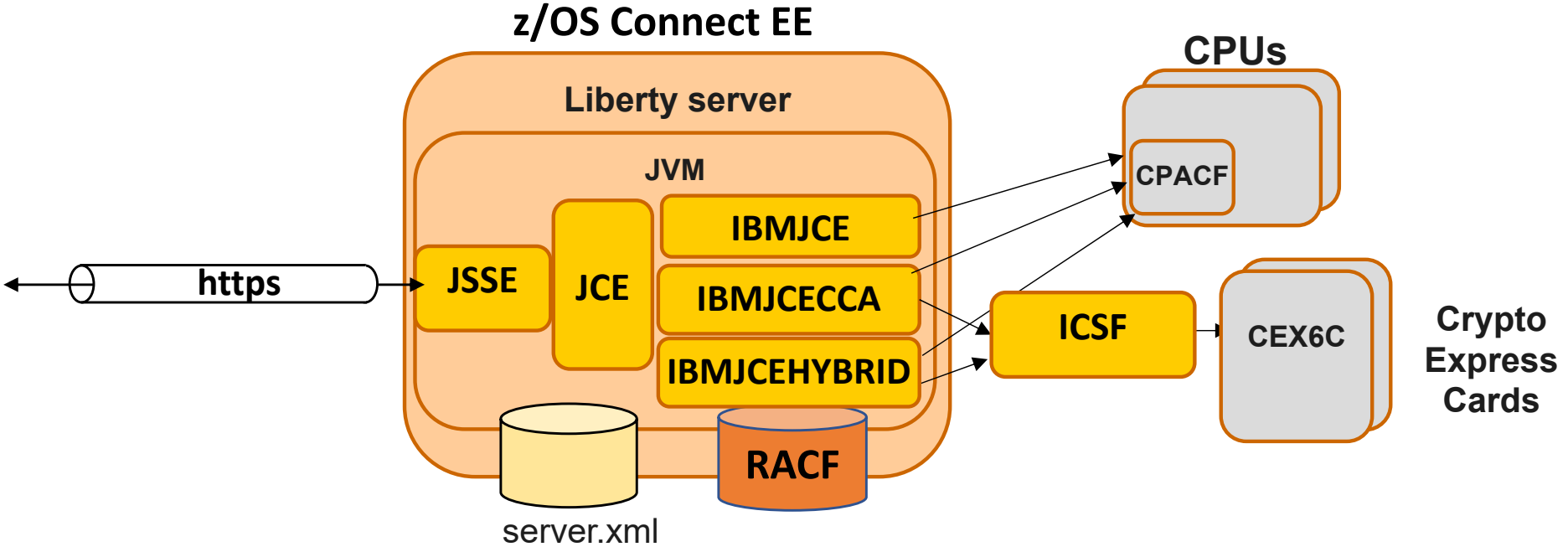


The actions which can be controlled by authorization are deploying, querying, updating, starting, stopping and deleting of APIs, services and API requesters.

mitchj@us.ibm.com

Using JSSE with Liberty

The server XML configuration defines the HTTPS ports, key rings, and other JSSE attributes

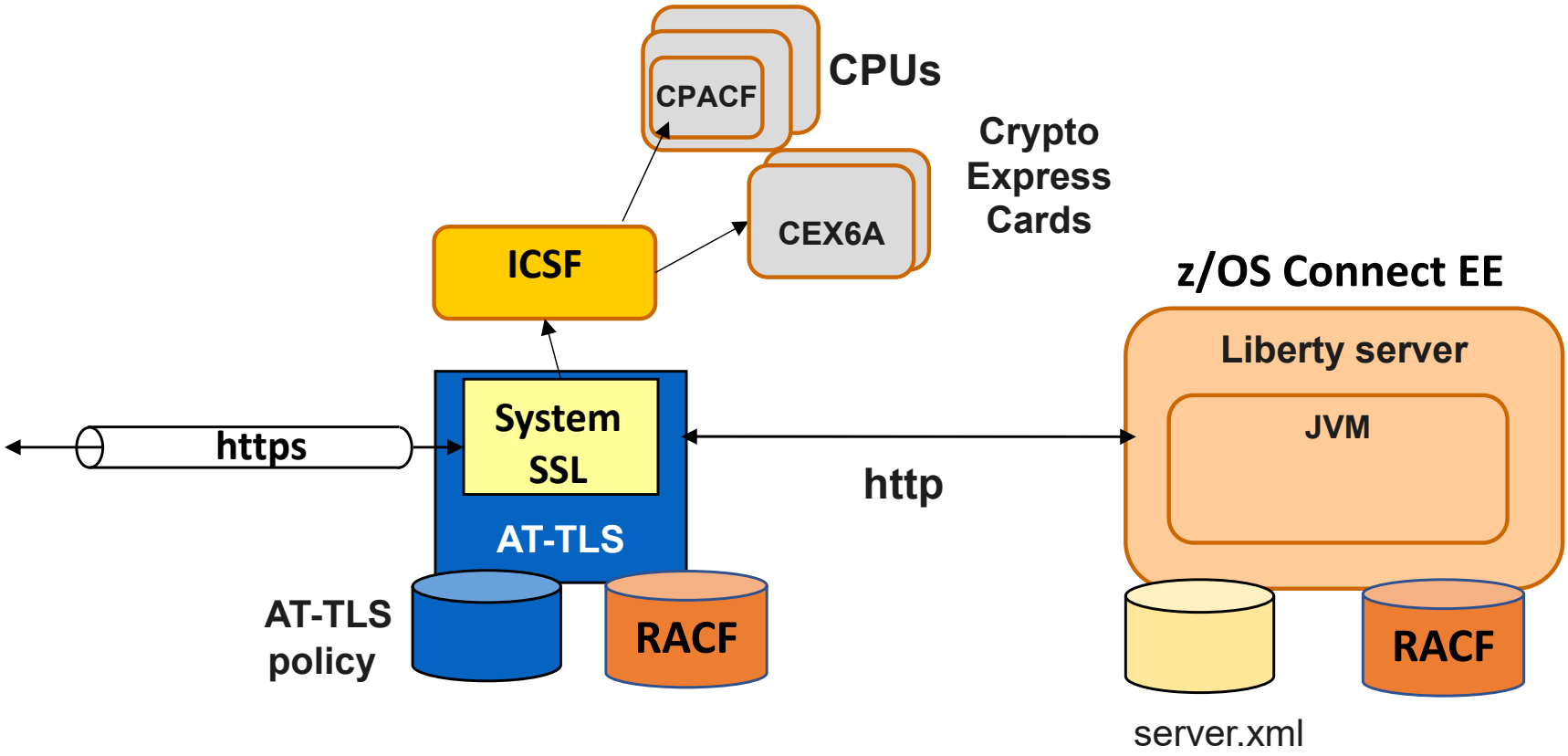


- z/OS Connect EE support for TLS is based on **Liberty** server support
- **Java Secure Socket Extension** (JSSE) API provides framework and Java implementation of TLS protocols used by Liberty HTTPS support
- **Java Cryptography Extension** (JCE) is standard extension to the Java Platform that provides implementation for cryptographic services
- **IBM Java SDK** for z/OS provides three different JCE providers, **IBMJCE, IBMJCECCA and IBMJCEHYBRID**.
- The JCE providers access CPACF (**CP Assist for Cryptographic Functions)** directly, therefore keep your Java service levels current.

mitchj@us.ibm.com

Using AT-TLS with Liberty

The server XML configuration defines no HTTPS ports, key rings or other 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

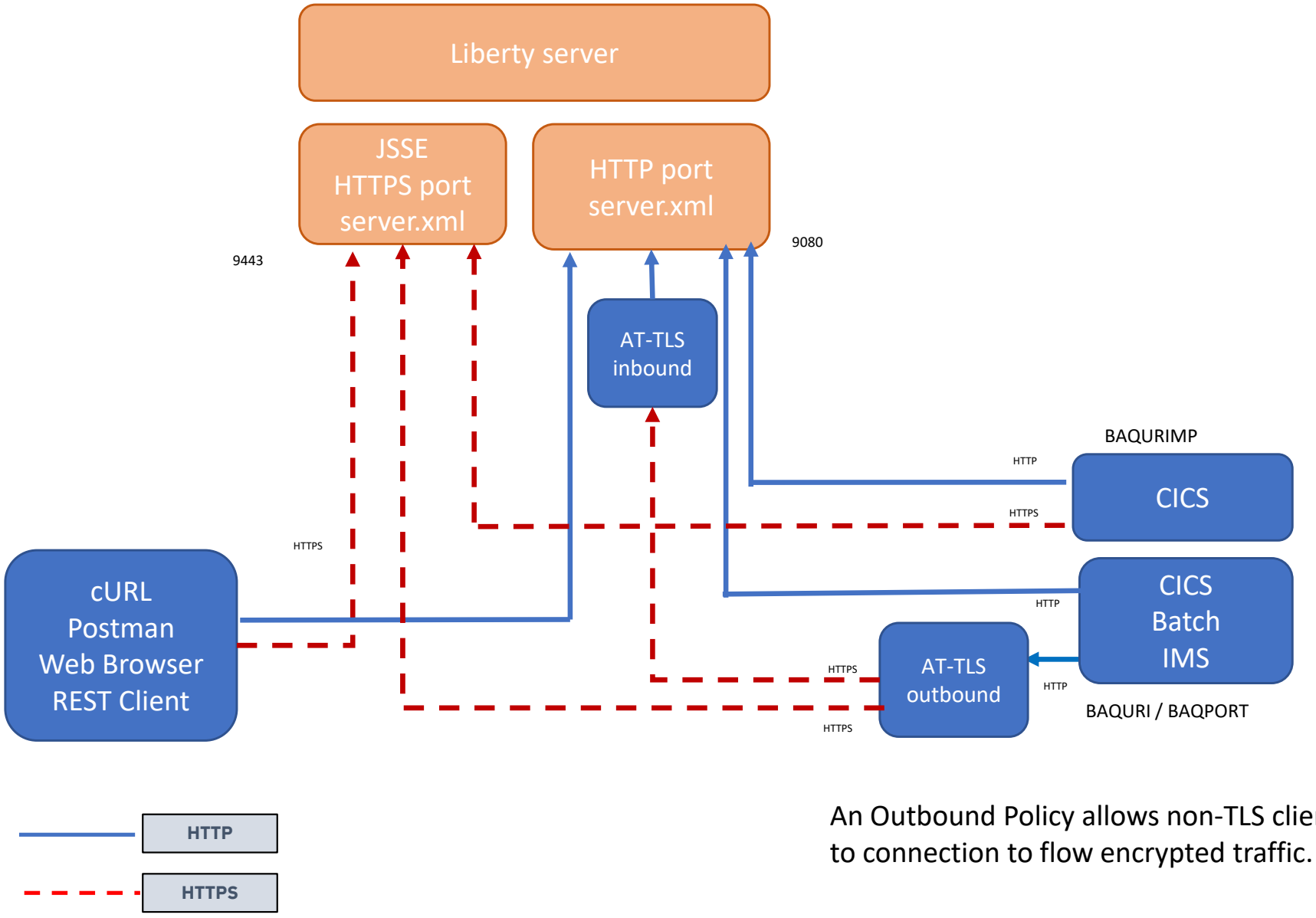
mitchj@us.ibm.com

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, etc.	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

mitchj@us.ibm.com

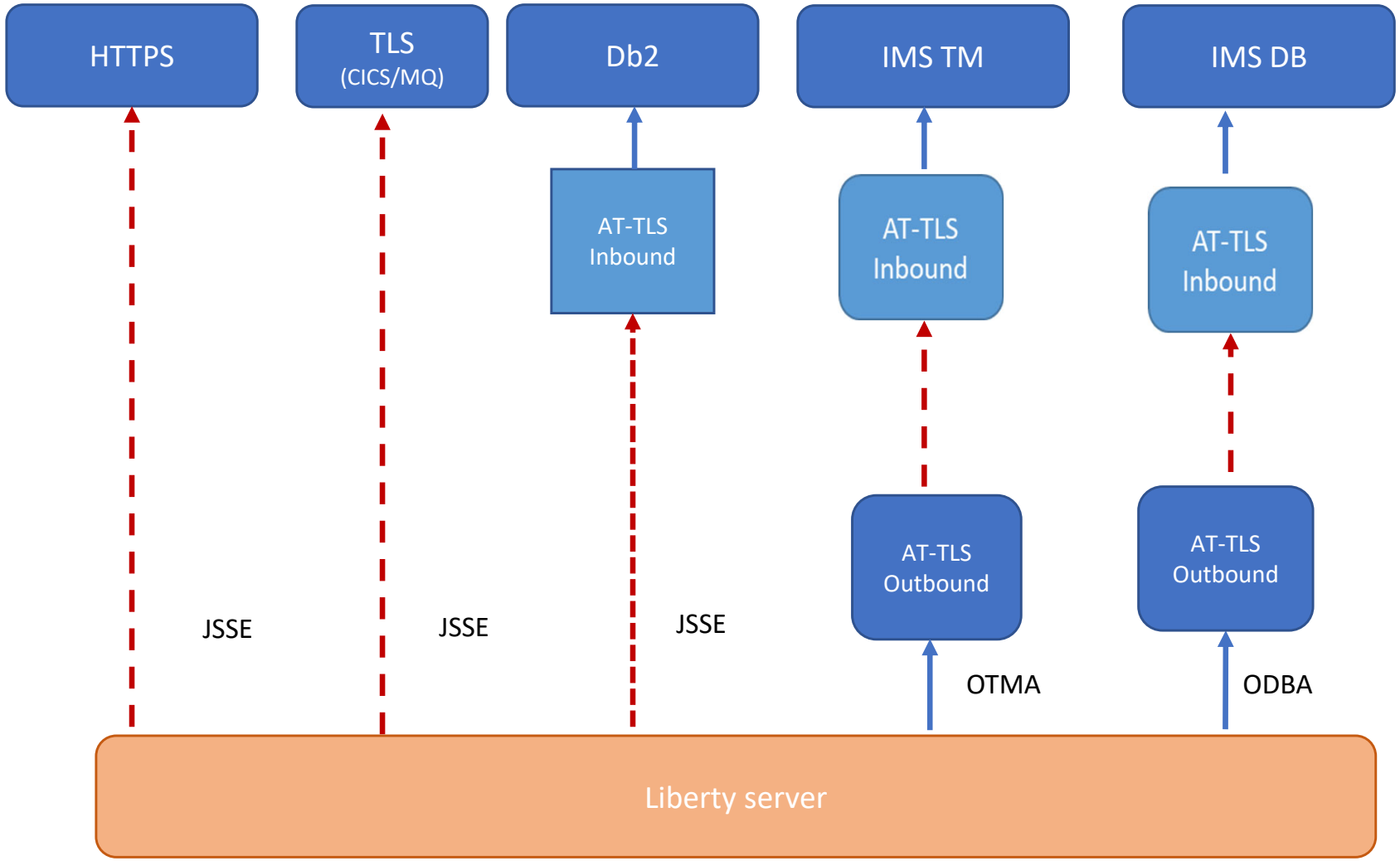
TLS client encryption to a Liberty server scenarios



An Outbound Policy allows non-TLS clients to connection to flow encrypted traffic.

mitchj@us.ibm.com

TLS encryptions from a Liberty 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 Liberty

mitchj@us.ibm.com

Let’s explore using TLS for encryption and data integrity using samples 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

Tech-Tip: 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.

mitchj@us.ibm.com

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" />
```

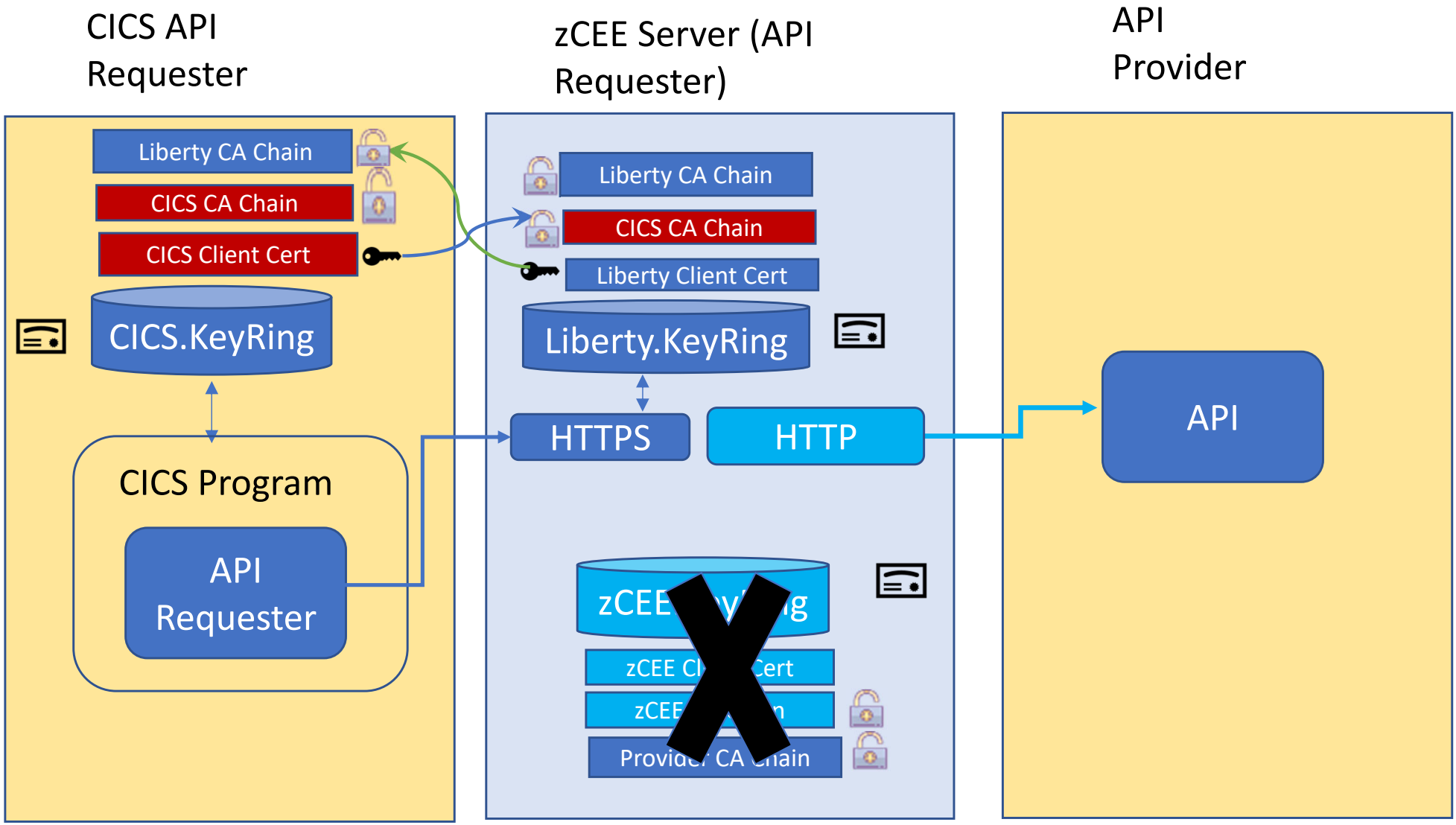
SSL repertoires

```
<zosconnect_authorizationServer sslCertsRef="SSL repertoire"/>
<zosconnect_cicsIpicConnection sslCertsRef="SSL repertoire"/>
<zosconnect_endpointConnect sslCertsRef="SSL repertoire"/>
<zosconnect_zosConnectRestClient sslCertsRef="SSL repertoire"/>
<zosconnect_zosConnectServiceRestClientConnection sslCertsRef="SSL repertoire"/>
```

F BAQSTRT,REFRESH,KEYSTORE

mitchj@us.ibm.com

TLS handshake scenario (CICS inbound handshake)

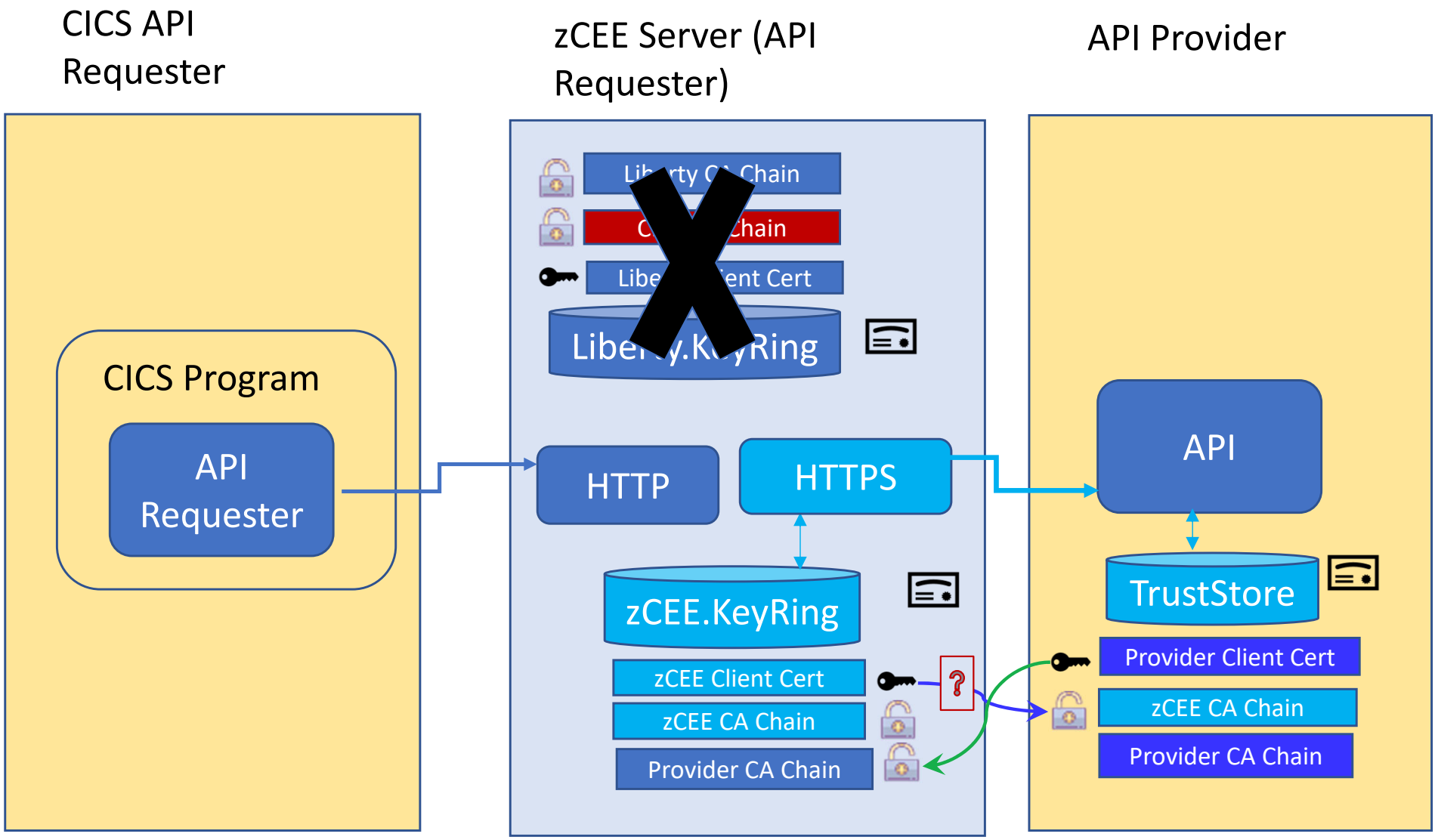



```
KEYRING=CICS.KeyRing,
ENCRYPTION=STRONG,
MAXSSLTCBS=8,
SSLDELAY=600
```

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

mitchj@us.ibm.com

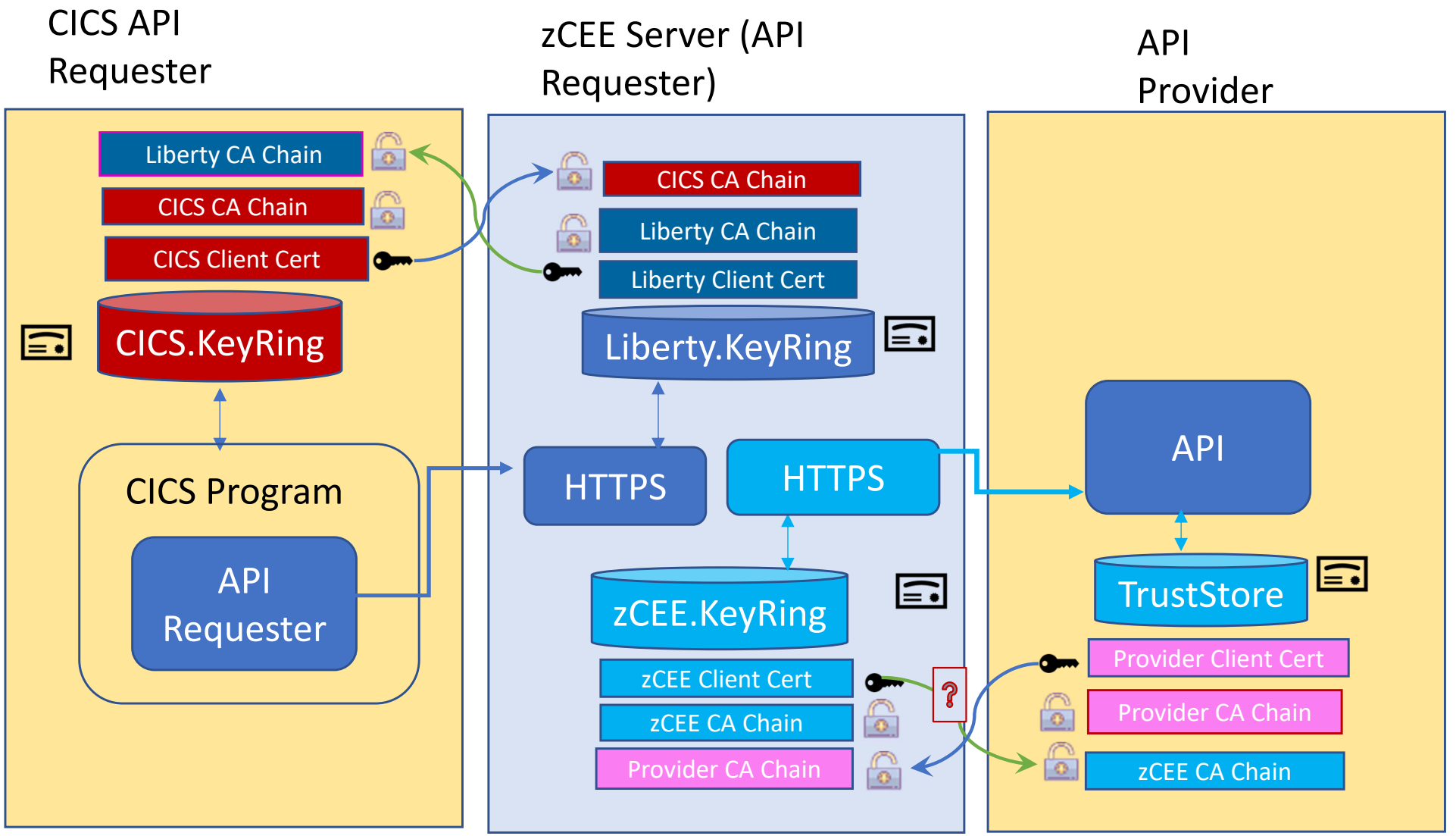
TLS handshake scenario (outbound handshake)



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

mitchj@us.ibm.com

TLS handshake scenario (multiple handshakes)



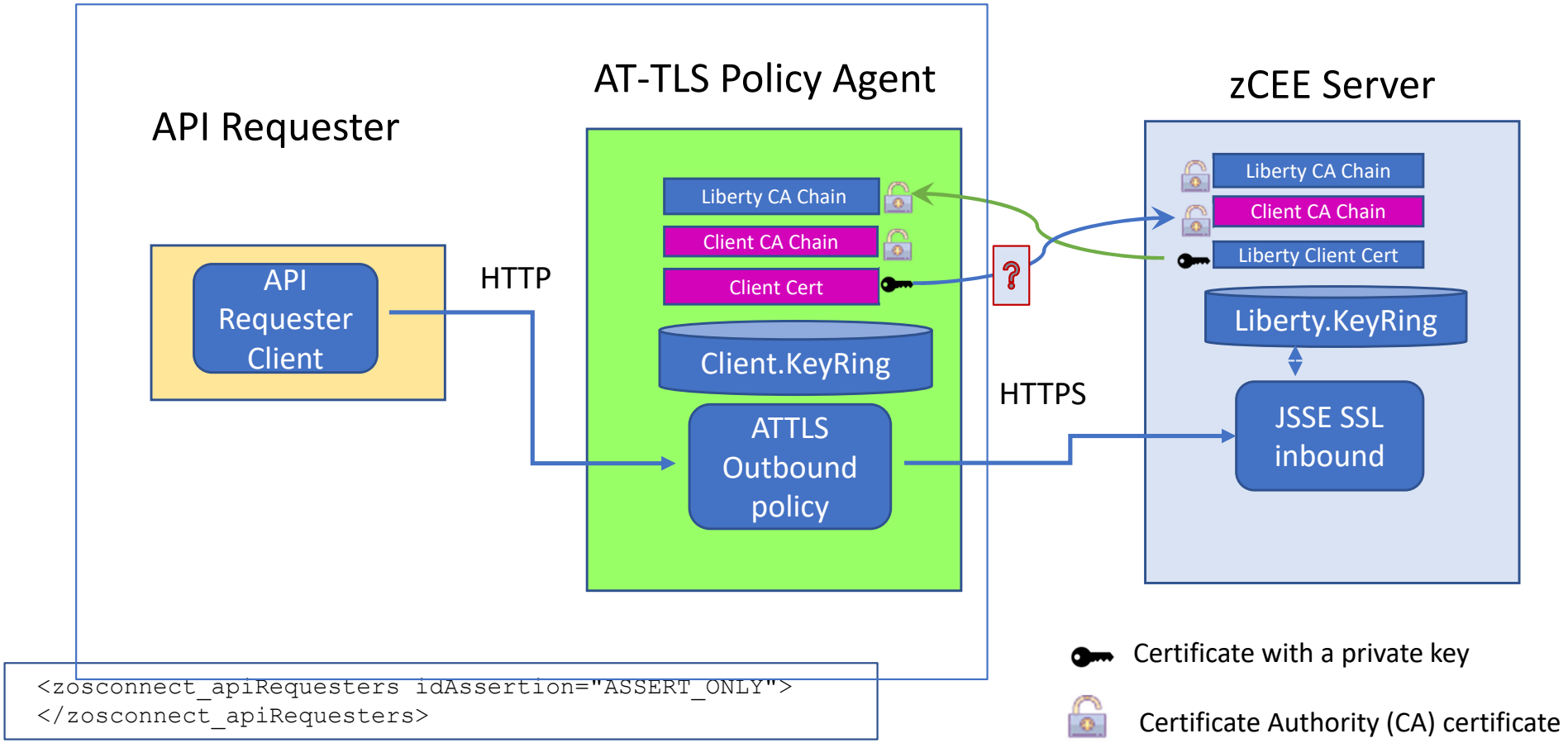

```
KEYRING=CICS.KeyRing,
ENCRYPTION=STRONG,
MAXSSLTCBS=8,
SSLDELAY=600
```

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

mitchj@us.ibm.com

AT-TLS - outbound policy handshake scenarios

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



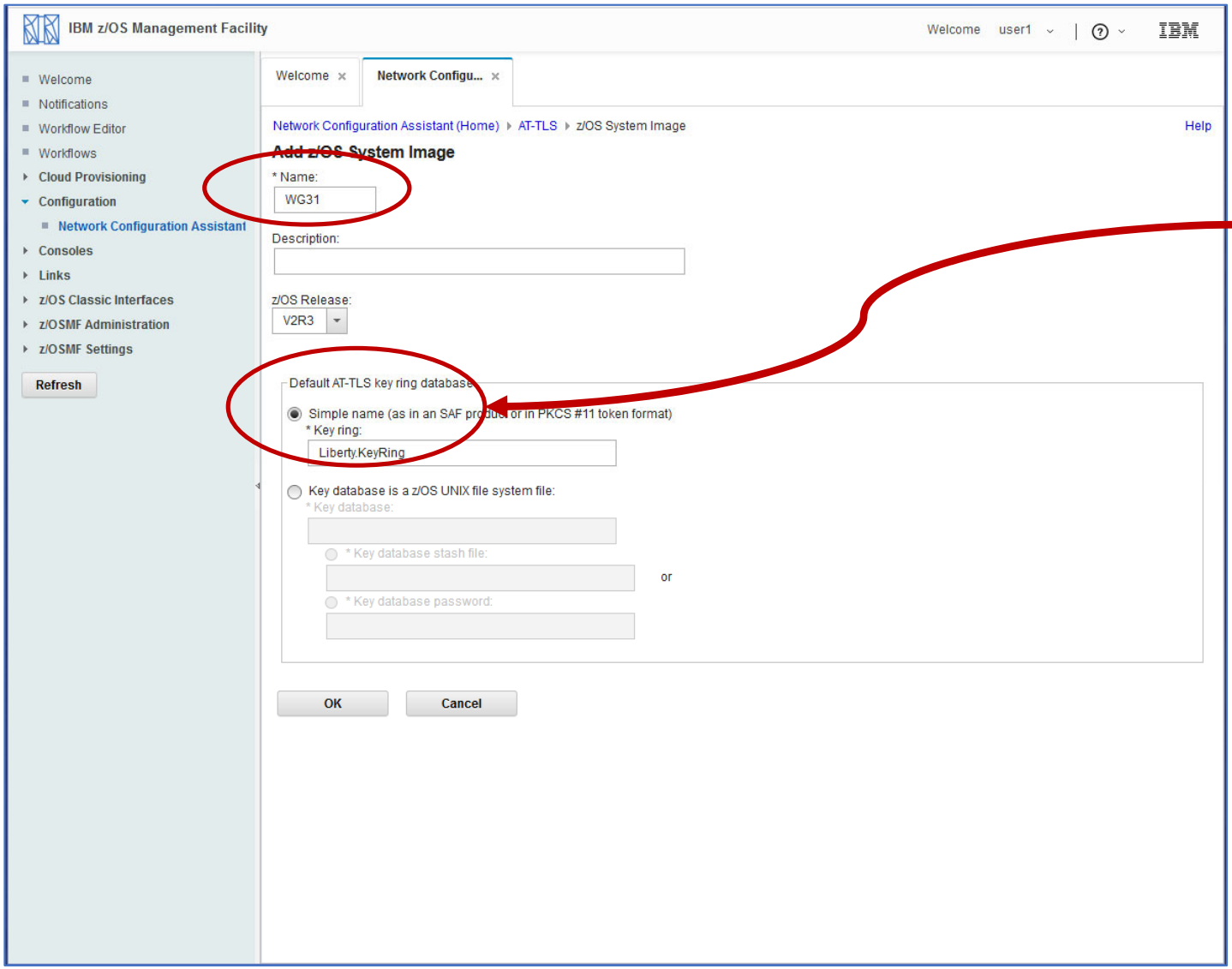

```
<zosconnect_apiRequesters idAssertion="ASSERT_ONLY">
</zosconnect_apiRequesters>
```

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

mitchj@us.ibm.com

AT-TLS – setting a default key ring

Using zOSMF Network Configuration Assistant to provide default Key Ring name

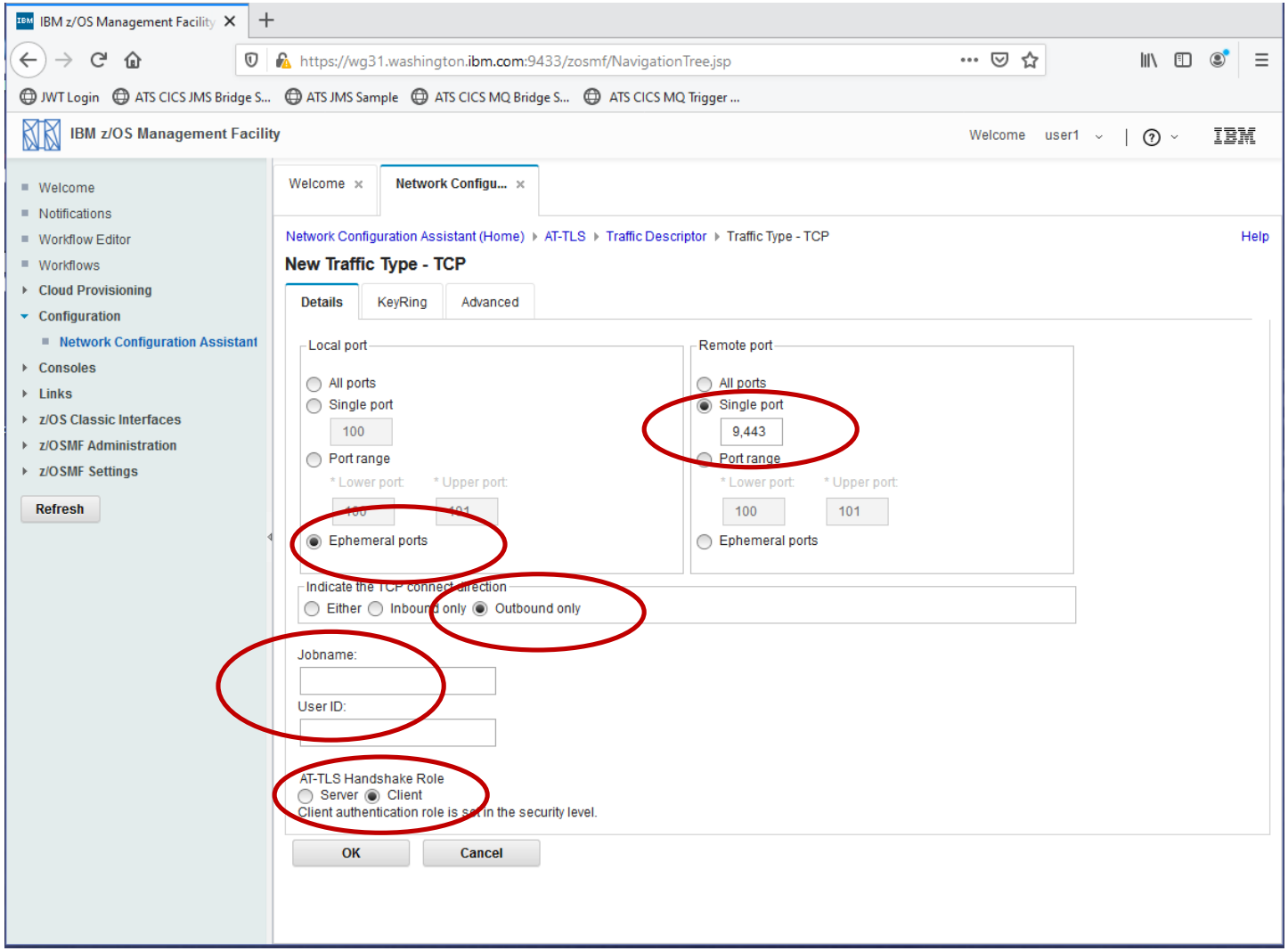


An SAF key ring name is specified as “identity/keyring". The current identity is used if the identity field is omitted.

mitchj@us.ibm.com

AT-TLS – configuring an outbound policy

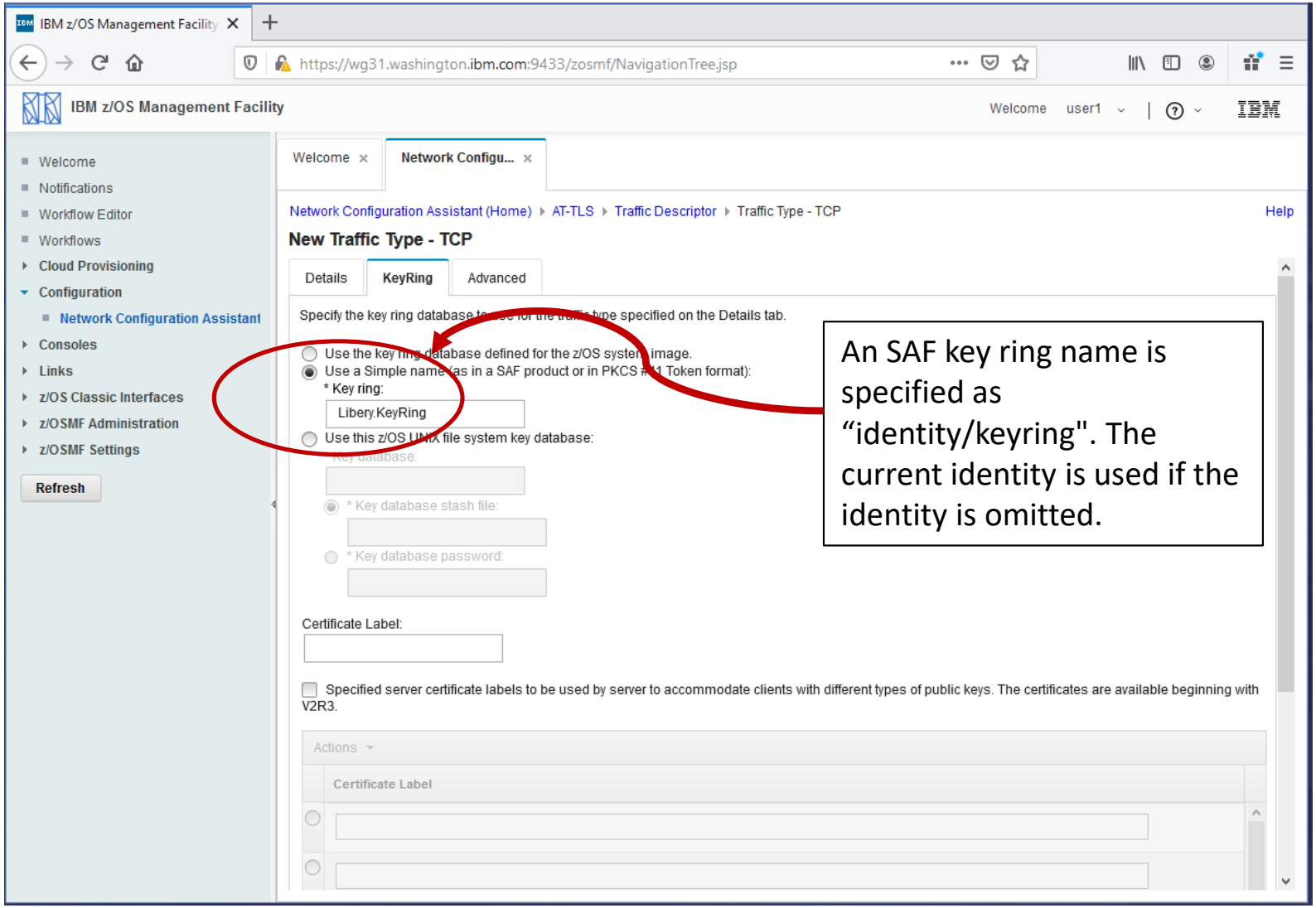
Using zOSMF Network Configuration Assistant to configure traffic descriptor



mitchj@us.ibm.com

AT-TLS - setting a policy key ring

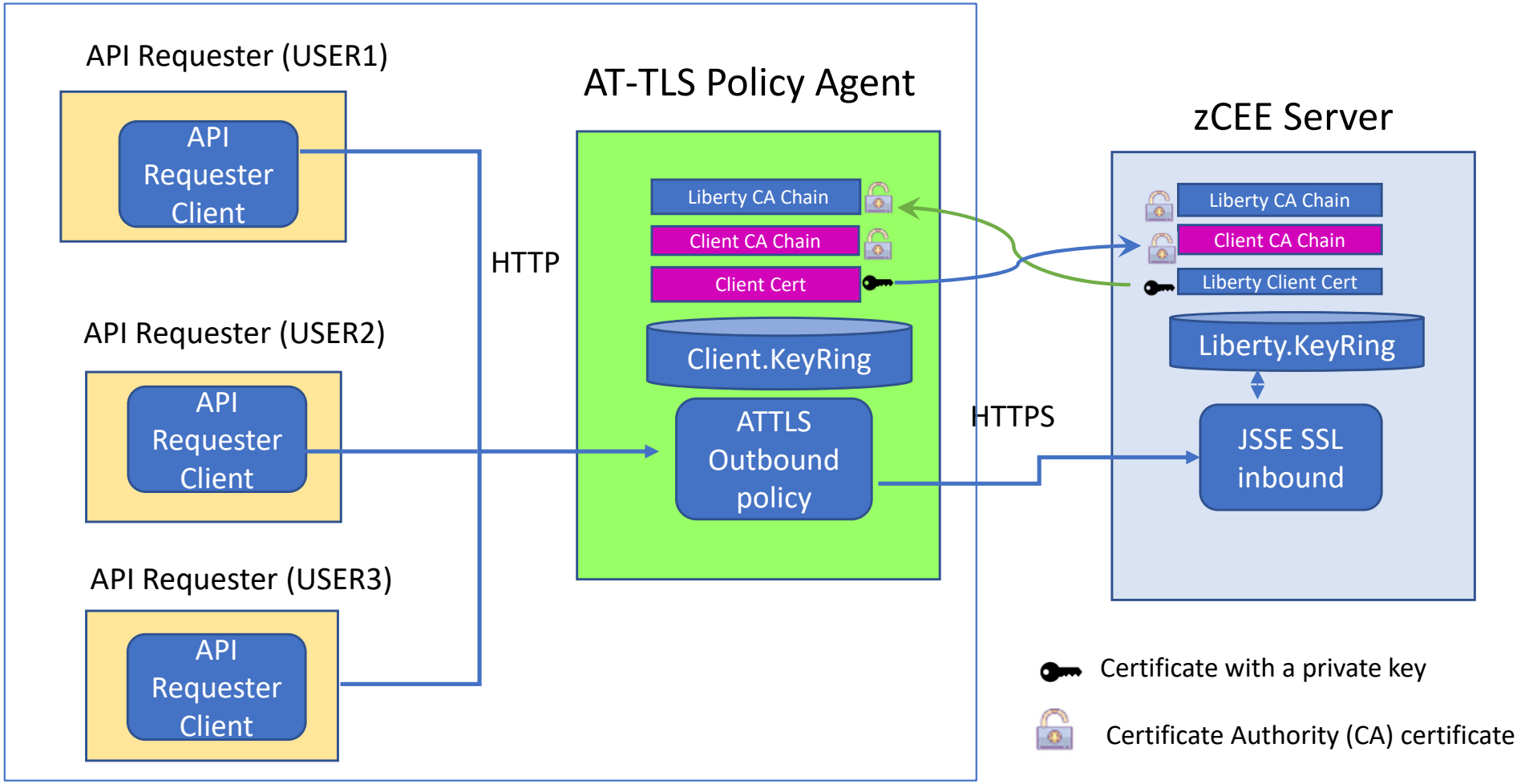
Using zOSMF Network Configuration Assistant to traffic descriptor Key Ring name



mitchj@us.ibm.com

AT-TLS - outbound policy handshake scenario

Use of a common key ring name with multiple client identities



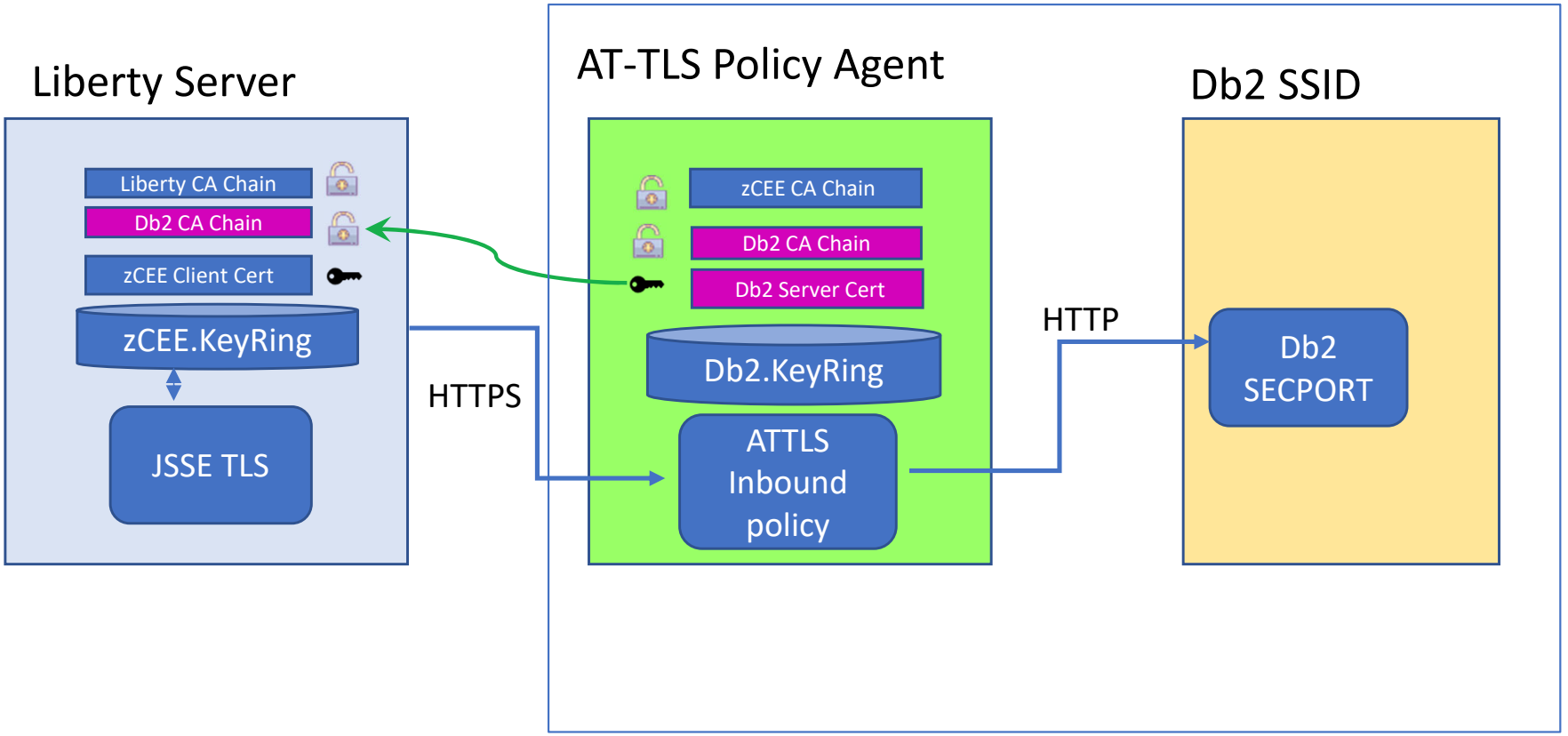
- Each user owns a keyring with the name Liberty.KeyRing.
- Each key ring has a different default client certificate for mutual authentication purposes.

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

mitchj@us.ibm.com

AT-TLS - inbound policy handshake scenario (Db2)

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

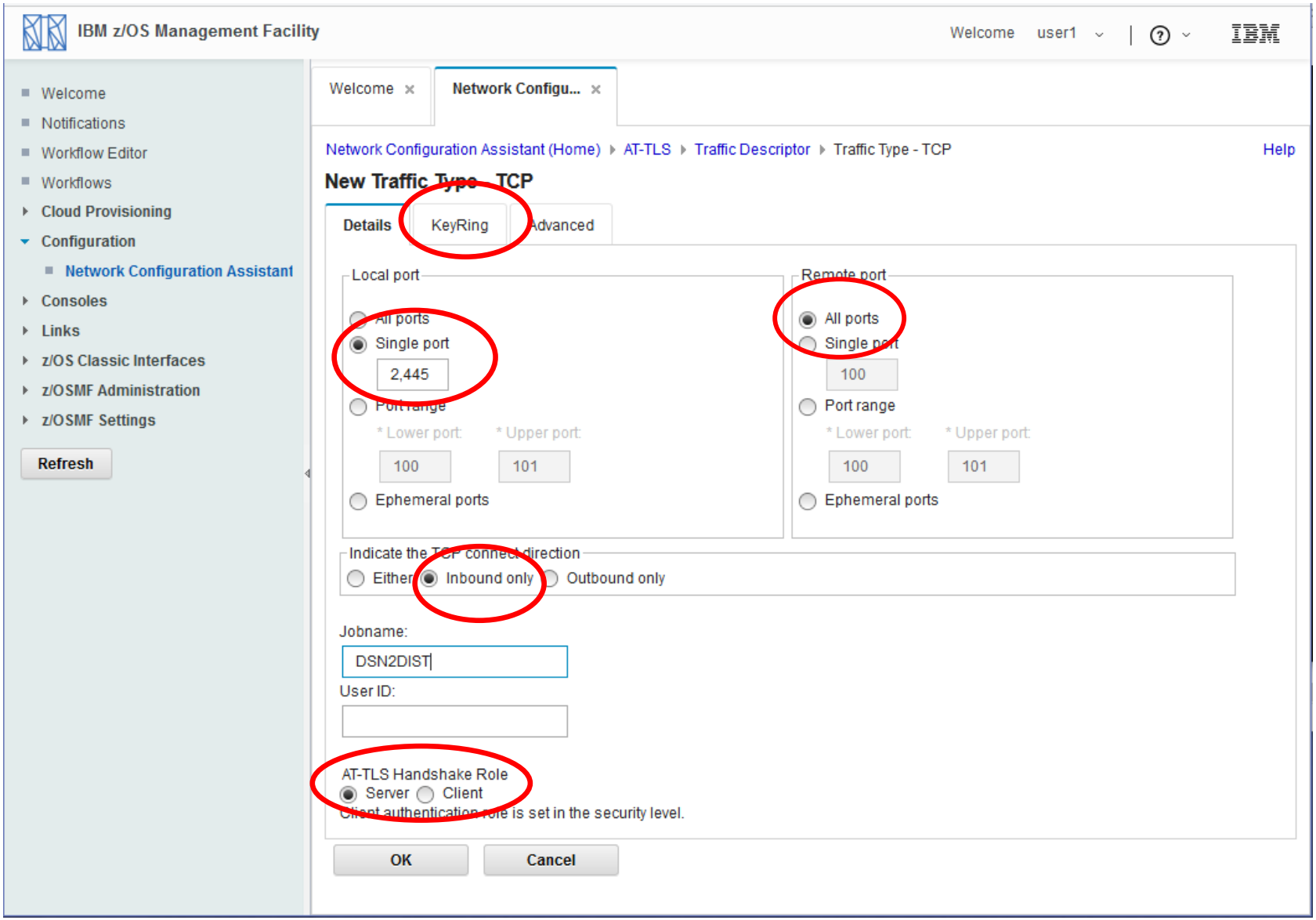


Note that DB2 is AT-TLS aware

mitchj@us.ibm.com

AT-TLS – configuring an inbound policy

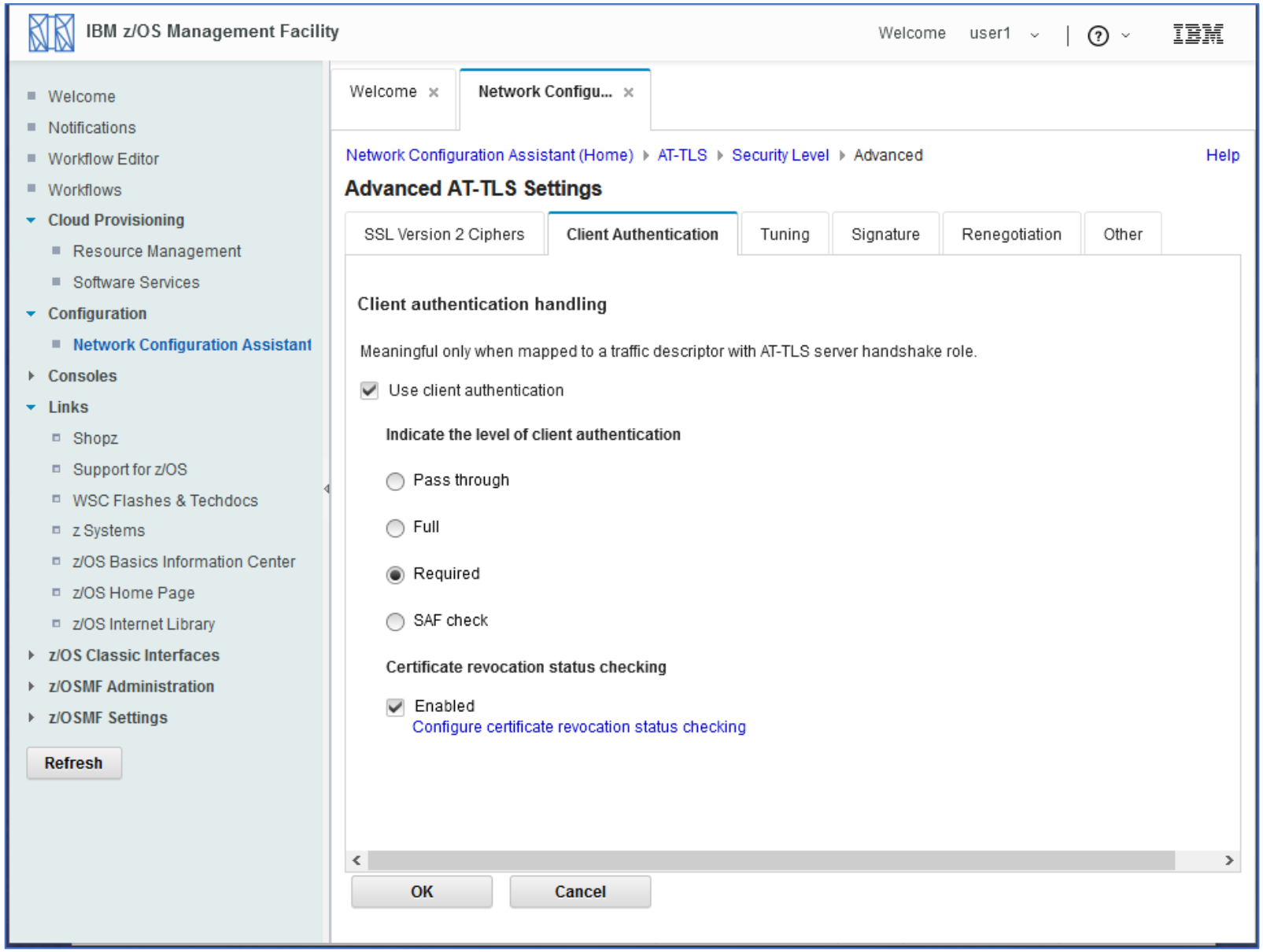
Using zOSMF Network Configuration Assistant



mitchj@us.ibm.com

AT-TLS – configuring client authentication

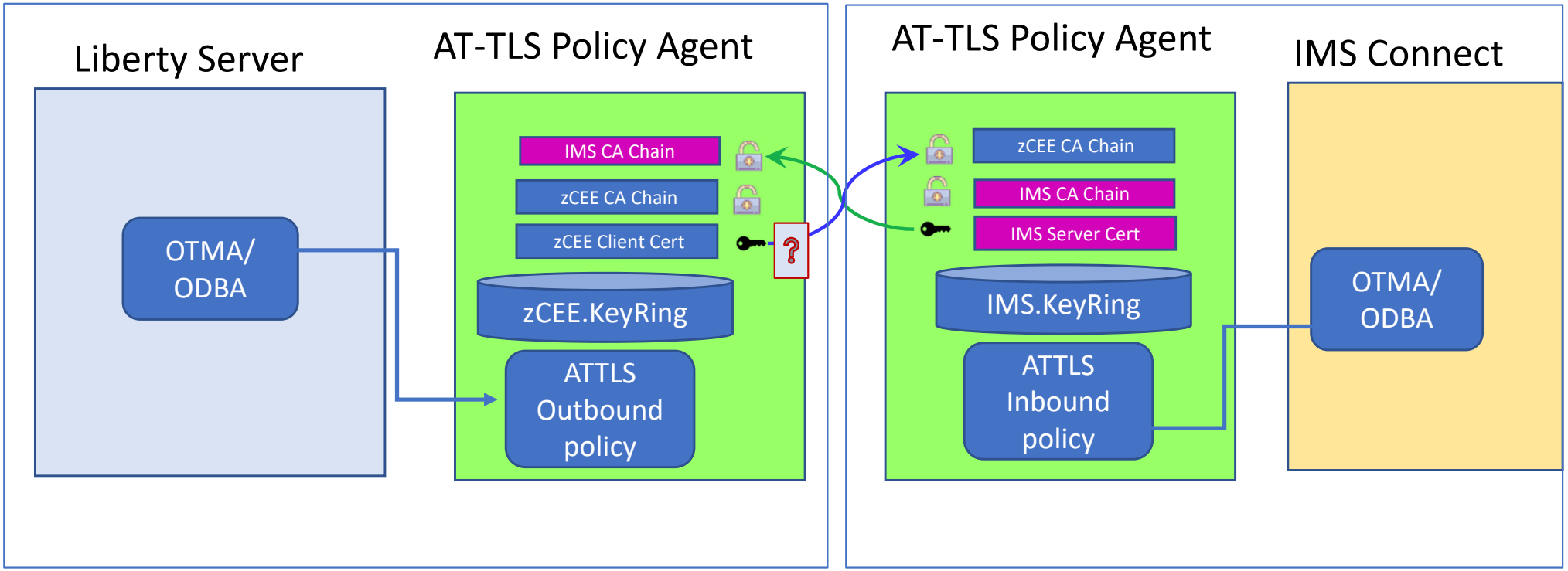
Using zOSMF Network Configuration Assistant (mutual authentication)



mitchj@us.ibm.com

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. Use PassTickets to assert identity.

mitchj@us.ibm.com

Configuring TLS Encryption with JSSE

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

mitchj@us.ibm.com

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

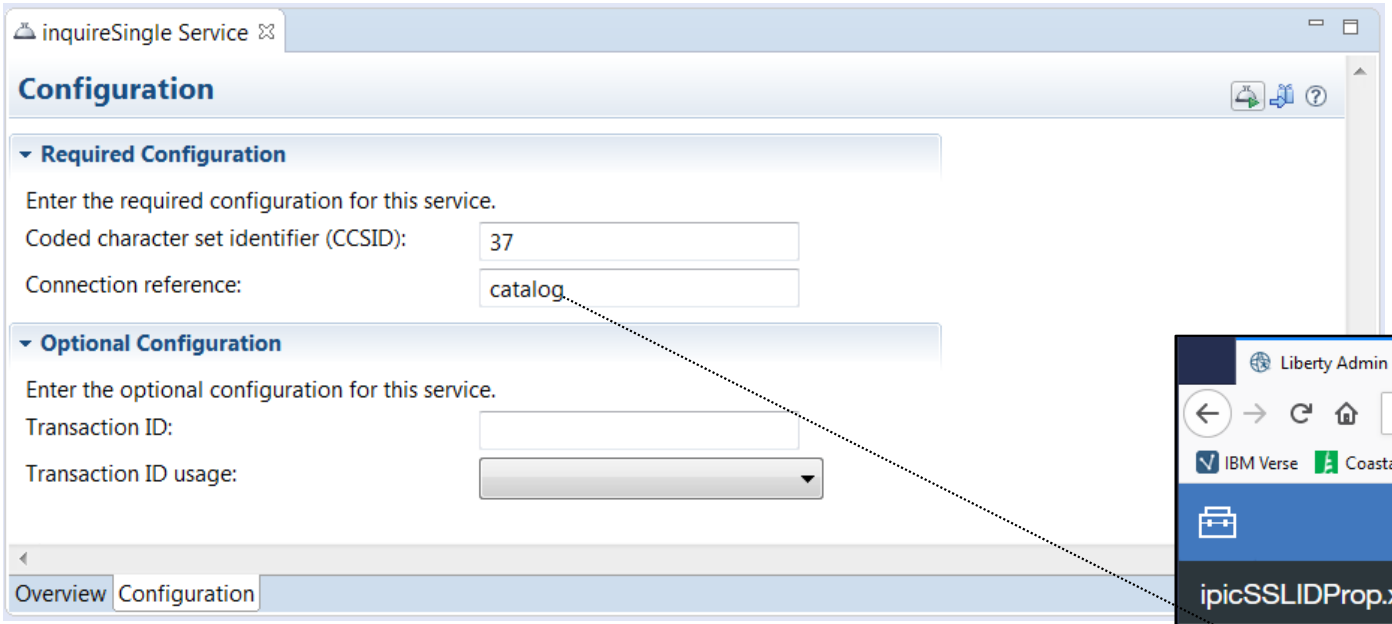
CipherSpec	Equivalent CipherSuite (IBM JRE)	Equivalent CipherSuite (Oracle JRE)	Protocol	FIPS 140-2 compatible
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	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

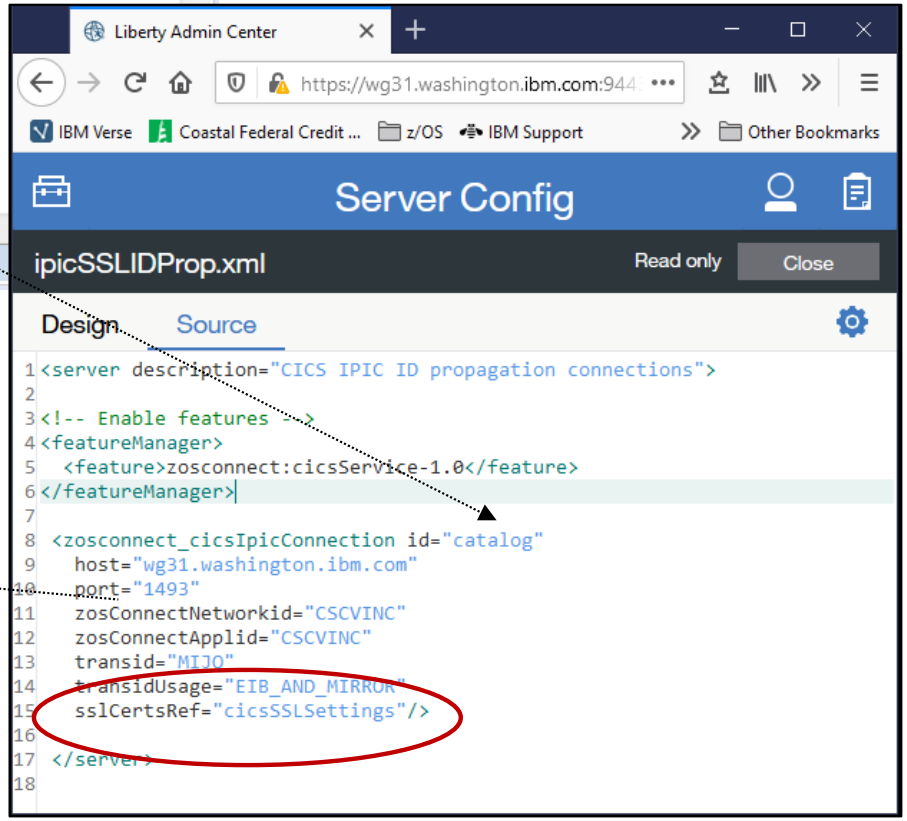
https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.dev.doc/q113210_.htm

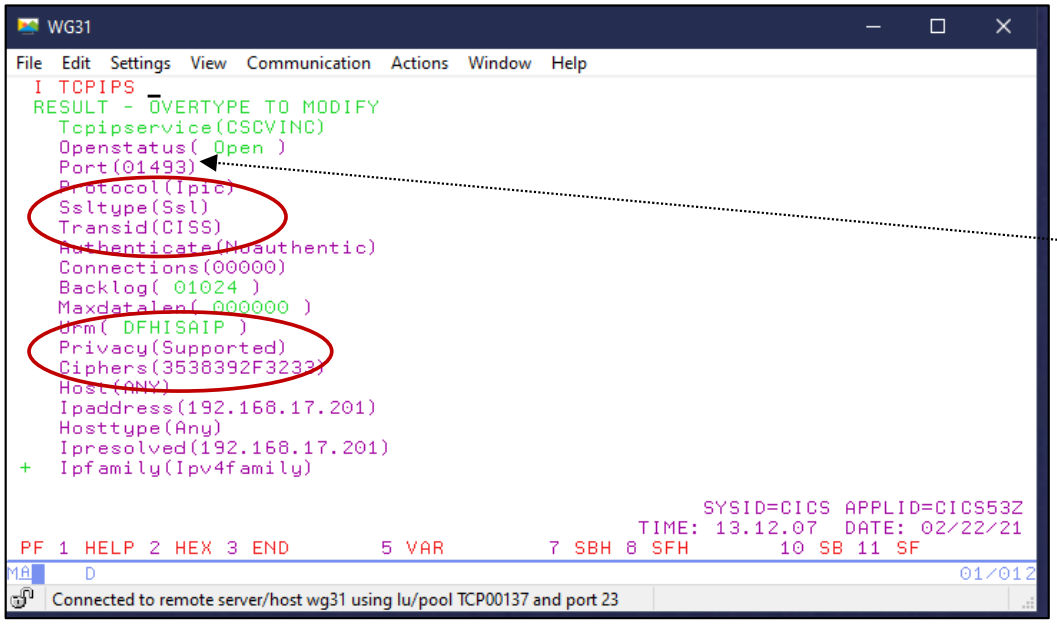
mitchj@us.ibm.com

CICS IPIC using TLS

The server.xml file is the key configuration file:







Define IPIC/TLS connections to CICS

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

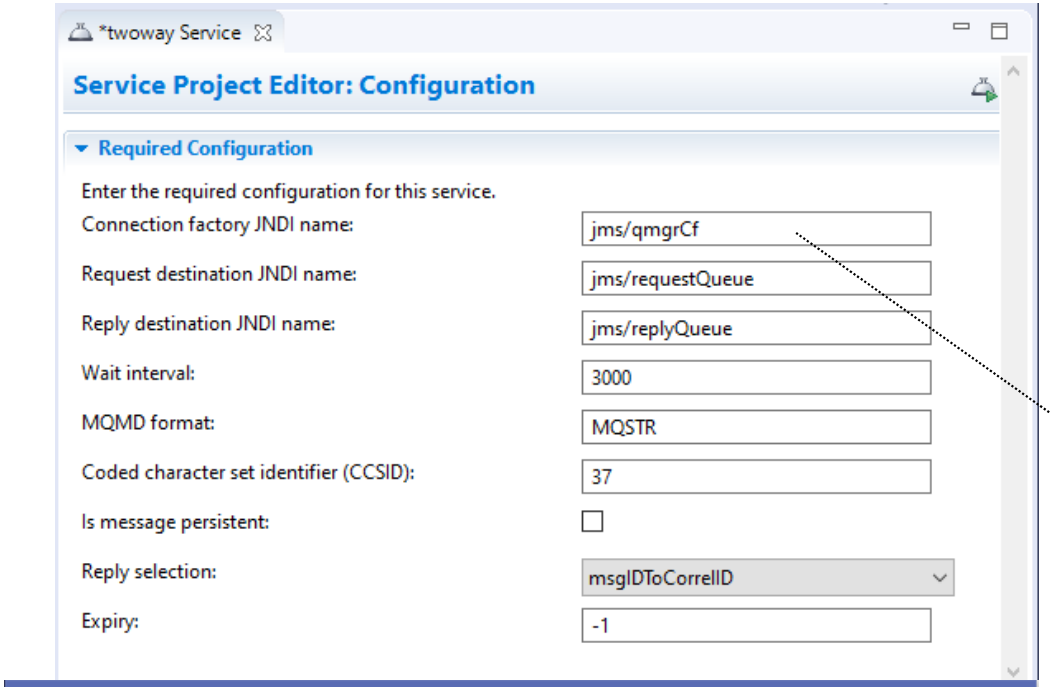
2-character cipher number	4-character cipher number	Short name	Description [1]	FIPS 140-2	Base security level FMID HCPT440	Security level 3 FMID JCPT441
00	0000	TLS_NULL_WITH_NULL_NULL	No encryption or message authentication and RSA key exchange		X	X
01	0001	TLS_RSA_WITH_NULL_MD5	No encryption with MD5 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

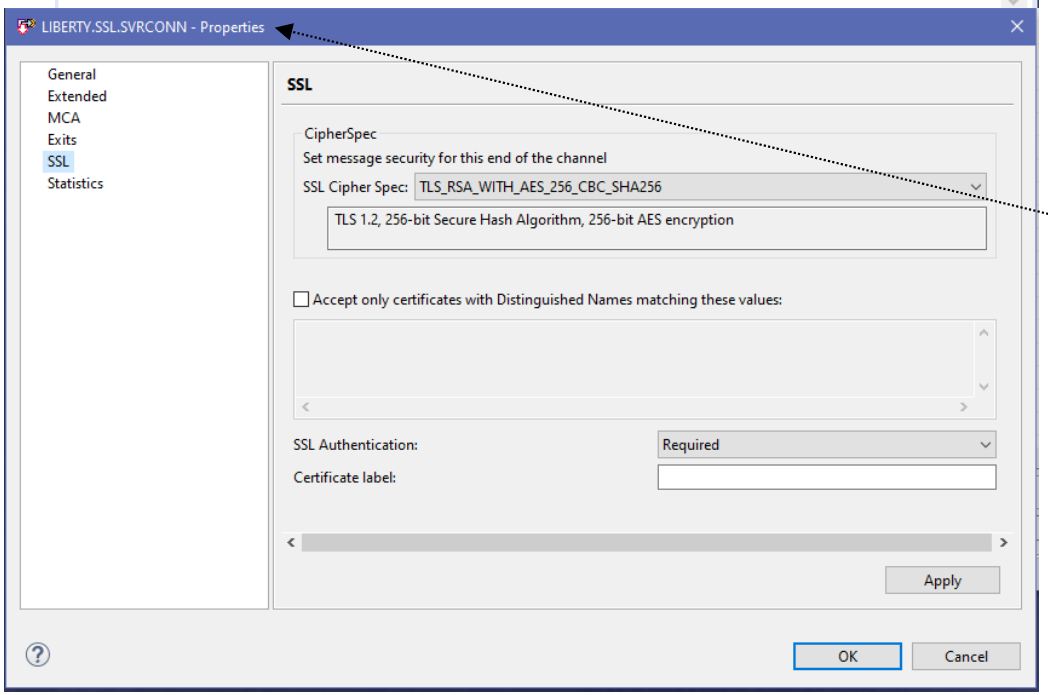
https://www.ibm.com/support/knowledgecenter/SSLTBW_2.4.0/com.ibm.zos.v2r4.gska100/csdcwh.htm

mitchj@us.ibm.com

MQ JMS using TLS

The server.xml file is the key configuration file:





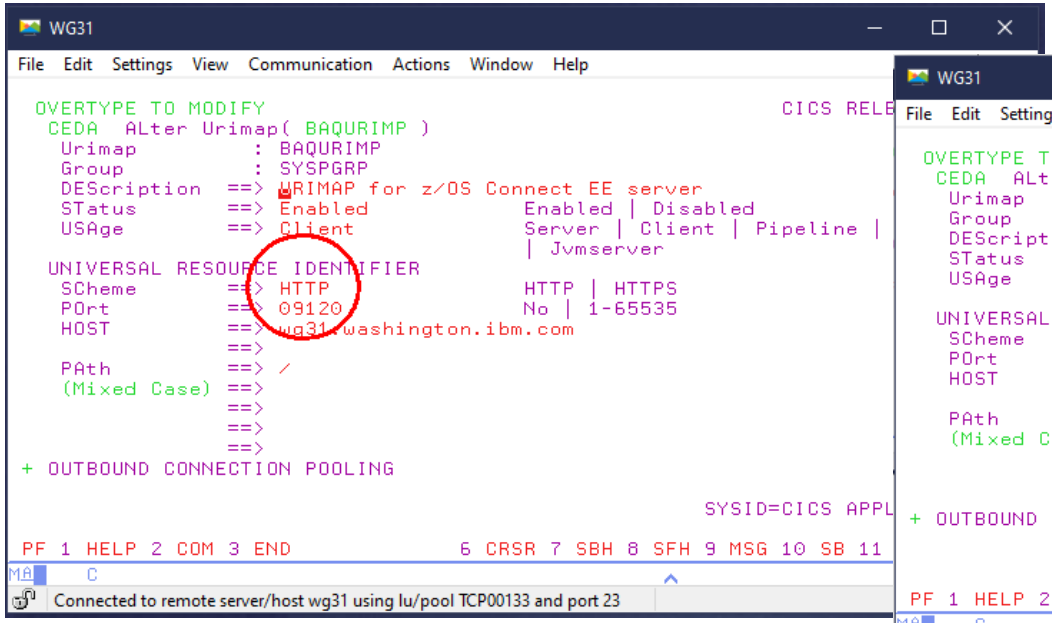
Tech/Tip: API Requester - HTTP v HTTPS

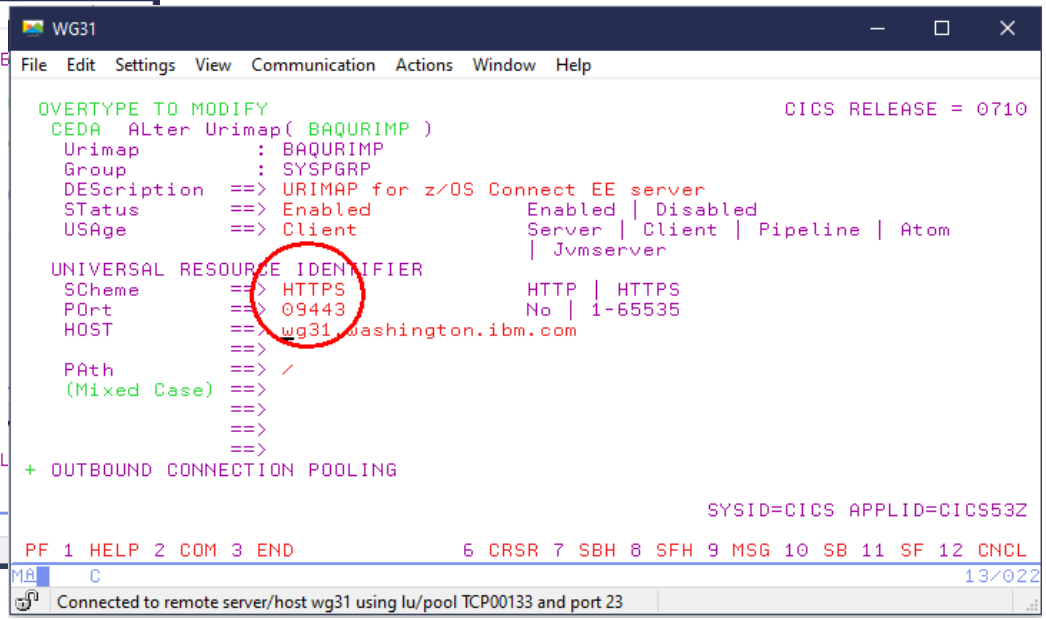
MVS Batch and IMS with AT-TLS

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

```
CEEOPTS DD *
 POSIX(ON),
 ENVAR("BAQURI=wg31.washington.ibm.com",
 "BAQPORT=9443")
```

CICS URIMAPs





Field BAQ-ZCON-SERVER-URI was added to BAQRINFO in V3.0.37.

MOVE "URIMAP01" TO BAQ-ZCON-SERVER-URI.

mitchj@us.ibm.com

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

mitchj@us.ibm.com

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

mitchj@us.ibm.com

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.

mitchj@us.ibm.com

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.

mitchj@us.ibm.com

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.

mitchj@us.ibm.com

Tech/Tip: Enabling hardware cryptography key rings

jvm.options

```
-Djava.security.properties=${server.config.dir}/java.security
```

java.security

```
security.provider.1=com.ibm.crypto.hdwrCCA.provider.IBMJCECCA
security.provider.2=com.ibm.crypto.provider.IBMJCE
security.provider.3=com.ibm.jsse2.IBMJSSEProvider2
security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
. . . . . .
```

Enabling the IBMJCECCA provider

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

Enabling the IBMJCEHYBRID provider

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

See URL https://www.ibm.com/support/pages/node/6209109 for details on implementing IBMJCECCA and IBMJCEHYBRID hardware encryption providers

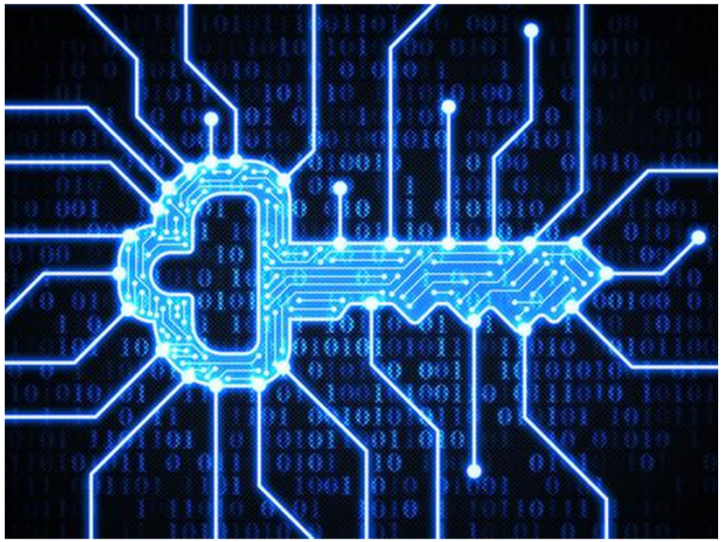
mitchj@us.ibm.com

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.

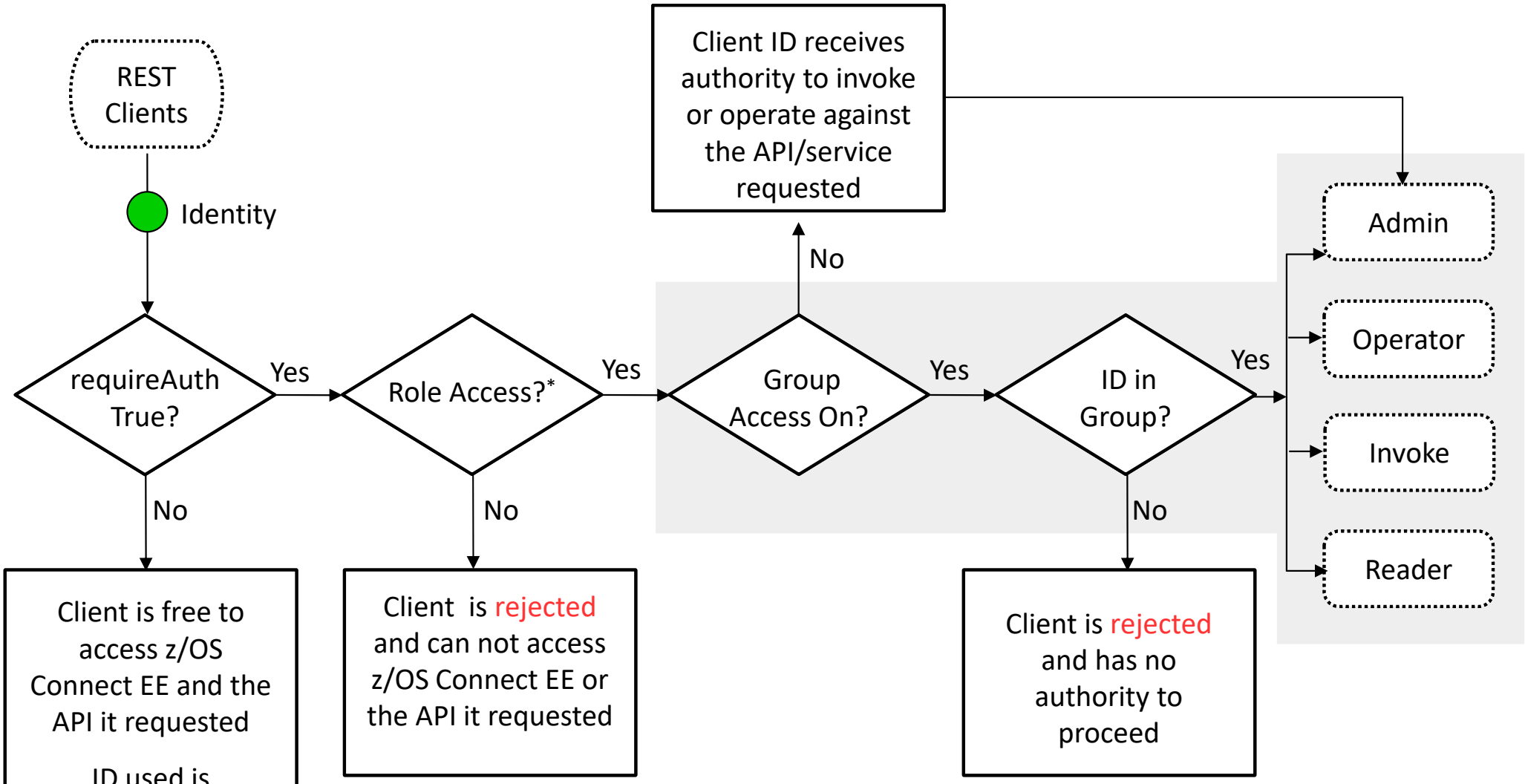


mitchj@us.ibm.com

Authorization

Once we have an identity, then what?

Security flow with z/OS Connect EE




```
*RDEFINE EJBROLE profilePrefix.zos.connect.access.roles.zosConnectAccess UACC(NONE)
 PERMIT profilePrefix.zos.connect.access.roles.zosConnectAccess[1]
       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

mitchj@us.ibm.com

z/OS Connect Authorization Functions

Operations - Ability to perform all z/OS Connect EE operations and actions except for function *Invoke*. The following operations/actions are allowed:

APIs:

- *To obtain a list of all APIs (GET).**
- For a specific API, get its details and API Swagger document (GET) and *deploy (POST)**, update (PUT), start(PUT), stop(PUT), and delete(DELETE) it.

Services:

- *To obtain a list of all services or statistics for all services (GET).**
- For a specific service, get its details, request and response schemas, statistics (GET) and *deploy(POST)**, update(PUT), start(PUT), stop(PUT), and delete(DELETE) it.

API Requesters:

- *To obtain a list of all API requesters* (GET).*
- For a specific API requester, get its details (GET) and *deploy (POST)**, update(PUT), start(PUT), stop(PUT), and delete(DELETE) it.

*These APIs use either the POST or GET method to invoke the REST APIs whose URIs have no path parameter. Therefore, the name of the API, or service or API Requester is not available. For authorization, only the default or global groups list can be used since no specific group list can be determined (for deployment, the name is embedded in the archive file).

mitchj@us.ibm.com

z/OS Connect Authorization Levels

Reader - Ability for:

APIs:

- *To obtain a list of all APIs* (GET) .*
- For a specific API, get its details and API Swagger document (GET).

Services:

- *To obtain a list of all services* (GET). *
- For a specific service, get its details and request and response schemas (GET).

API Requesters:

- *To obtain a list of all API requesters* (GET).*
- For a specific API requester, get its details (GET) .

Invoke - Ability to invoke user APIs, services and/or API requesters (POST,PUT,GET,DELETE,+).

Admin - All z/OS Connect EE actions are allowed, including all corresponding *Operations*, *Invoke*, and *Reader* actions configured for the same z/OS Connect resource.

*These APIs use either the POST or GET method to invoke the REST APIs whose URIs have no path parameter. Therefore, the name of the API, service or API Requester is not available. For authorization, only the default or global groups list since no specific group list can be determined (for deployment, the name is embedded in the archive file.

mitchj@us.ibm.com

z/OS Connect administration API

Interface providing meta-data and life-cycle operations for z/OS Connect services, APIs and API requesters.

APIs : Operations for working with APIs

Show/Hide | List Operations | Expand Operations

GET	/apis	Returns a list of all the deployed z/OS Connect APIs
POST	/apis	Deploys a new API into z/OS Connect
DELETE	/apis/{apiName}	Undeploys an API from z/OS Connect
GET	/apis/{apiName}	Returns detailed information about a z/OS Connect API
PUT	/apis/{apiName}	Updates an existing z/OS Connect API

Services : Operations for working with services

Show/Hide | List Operations | Expand Operations

GET	/services	Returns a list of all the deployed z/OS Connect services
POST	/services	Deploys a new service into z/OS Connect
DELETE	/services/{serviceName}	Undeploys a service from z/OS Connect
GET	/services/{serviceName}	Returns detailed information about a z/OS Connect service
PUT	/services/{serviceName}	Updates an existing z/OS Connect service
GET	/services/{serviceName}/schema/{schemaType}	Returns the request or response schema for a z/OS Connect service

API Requesters : Operations that work with API Requesters.

Show/Hide | List Operations | Expand Operations

GET	/apiRequesters	Returns a list of all the deployed z/OS Connect API Requesters
POST	/apiRequesters	Deploys a new API Requester into z/OS Connect and invoke an API Requester call
DELETE	/apiRequesters/{apiRequesterName}	Undeploys an API Requester from z/OS Connect
GET	/apiRequesters/{apiRequesterName}	Returns the detailed information about a z/OS Connect API Requester
PUT	/apiRequesters/{apiRequesterName}	Updates an existing z/OS Connect API Requester

mitchj@us.ibm.com

z/OS Connect RESTful Administrative APIs Security

z/OS Connect uses group security for controlling authorization for accessing APIs. There are sets of default global groups for functional roles are configured in a zosConnectManager configuration element as shown below:

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

There are four classes of groups available controlling z/OS Connect functions, administration, operations, invoking and reader in our server. An authenticated identity membership in one or more of these groups provides access to the corresponding function to that identity. There is also a way to provide an alternative set of groups for functional roles for specific APIs, services, and API requesters in subordinate configuration elements in our server.

```
<zosConnectAPI name="cscvinc"
        adminGroup="CSCADMIN"  operationsGroup="CSCOPERS"
        invokeGroup="CSCINVKE" readerGroup="CSCREADR"/>

<service name="cscvincSelectService"
        adminGroup="CSCADMIN"  operationsGroup="CSCOPERS"
        invokeGroup="CSCINVKE" readerGroup="CSCREADR"/>

<apiRequester name="cscvinc_1.0.0"
        adminGroup="CSCADMIN"  operationsGroup="CSCOPERS"
        invokeGroup="CSCINVKE" readerGroup="CSCREADR"/
```


mitchj@us.ibm.com

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 interceptor only

Alternative interceptor list – authorization and audit interceptors

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')
```

mitchj@us.ibm.com

Tech/Tip: Server XML example – combining TLS/AUTH interceptor

```
<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"
         invokeGroup="iaprgrp1"/>
  </zosconnect_apiRequesters>
```

Global TLS security and authentication are enabled.

TLS security is disabled for this API requester archive artifact. Avoiding the HTTP 302 REDIRECT error.

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

mitchj@us.ibm.com

Example of z/OS Connect Authorization Levels

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

<zosconnect_zosConnectAPIs>
    <zosConnectAPI name="cscvinc"
        adminGroup="CSCADMIN"  operationsGroup="CSCOPERS"
        invokeGroup="CSCINVKE" readerGroup="CSCREADR"/>
    <zosConnectAPI name="db2employee"
        adminGroup="DB2ADMIN"  operationsGroup="DB2OPERS"
        invokeGroup="DB2INVKE" readerGroup="DB2READR"/>
</zosconnect_zosConnectAPIs>

<zosconnect_services>
    <service name="cscvincSelectService"
        adminGroup="CSCADMIN"  operationsGroup="CSCOPERS"
        invokeGroup="CSCINVKE" readerGroup="CSCREADR"/>
    <service name="selectEmployee"
        adminGroup="DB2ADMIN"  operationsGroup="DB2OPERS"
        invokeGroup="DB2INVKE" readerGroup="DB2READR"/>
</zosconnect_services>

<zosconnect_apiRequesters>
    <apiRequester name="cscvincSelectService"
        adminGroup="CSCADMIN"  operationsGroup="CSCOPERS"
        invokeGroup="CSCINVKE" readerGroup="CSCREADR"/>
    <apiRequester name="selectEmployee"
        adminGroup="DB2ADMIN"  operationsGroup="DB2OPERS"
        invokeGroup="DB2INVKE" readerGroup="DB2READR"/>
</zosconnect_apiRequesters>
```

When groups are specified for zosConnectAPI, service, or apiRequester configuration elements, the global groups are ignored for certain functions. Other functions, e.g., deploy new artifact, get a list or service statistics, only use the global groups.

mitchj@us.ibm.com

z/OS Connect Authorization Summary

- Only members of groups GMADMIN, GMOPERS or GMREADER can connect to a z/OS server from the API toolkit.

- Only members of groups GMADMIN or GMOPERS can deploy new z/OS Connect API, service or API requester artifacts.

- Members of groups GMADMIN, GMOPERS, DB2ADMIN or DB2OPERS can not manage (e.g., change, stop or delete) z/OS Connect artifacts *managed* by group CSCOPERS or CSCADMIN.

- Members of groups GMADMIN, GMOPERS, CSCADMIN or CSCOPERS can not manage (e.g., change, stop or delete) z/OS Connect artifacts *managed* by group DB2OPERS or DB2ADMIN.

- Only members of group CSCADMIN, CSCINV, DB2ADMIN or DB2INVKE can invoke the artifacts defined in the subordinate element:
  - Members of group CSCADMIN or CSCVINKE can invoke artifacts managed by CSCINVKE
  - Members of group DB2ADMIN or DB2INVKE can invoke artifacts managed by DB2INVKE
  - Members of groups GMADMIN or GMINVOKE can not invoke any artifacts protected these specific subordinate groups.

mitchj@us.ibm.com

Tech-Tips: z/OS Connect Authorization Levels

Identity	Group
USER11	GMADMIN
USER12	GMOPERS
USER13	GMINVOKE
USER14	GMREADER
USER21	CSCADMIN
USER22	CSCOPERS
USER23	CSCINVKE
USER24	CSCREADR
USER31	DB2ADMIN
USER32	DB2OPERS
USER33	DB2INVKE
USER34	DB2READR

mitchj@us.ibm.com

Tech-Tip: z/OS Connect Authorization Deploy Sample

Deploy a new service as USER11

```
c:\z\passticket>curl -X POST --user user11:Z4GG7V4H  --data-binary @cscvincSelectService.sar  --header
"Content-Type: application/zip" --insecure https://wg31.washington.ibm.com:9443/zosConnect/services
{"zosConnect":{"serviceName":"cscvincSelectService","serviceDescription":"","serviceProvider":"CICS-
1.0","version":"1.0.0","serviceURL":"https://wg31.washington.ibm.com:9443
/zosConnect/services/cscvincSelectService","serviceInvokeURL":"https://wg31.washington.ibm.com:9443/zosConnect/
services/cscvincSelectService?action=invoke","dataXformProvider":"zosConnectWVXform-
1.0","serviceStatus":"Started"},"cscvincSelectService":{
"transidUsage":"EIB_ONLY","transid":"CSMI","program":"CSCVINC","connectionRef":"cscvinc","ccsid":"37"}}
```

This service is now managed by members of CSCADMIN and CSCOPERS

```
c:\z\passticket>curl -X PUT --user user11:V4BTUHV5  --insecure
https://wg31.washington.ibm.com:9443/zosConnect/services/cscvincSelectService?status=stopped
{"errorMessage":"BAQR0435W: The zosConnectAuthorization interceptor encountered an error while processing a
request.","errorDetails":"BAQR0409W: User user11 is not authorized to perform the request."}

c:\z\passticket>curl -X DELETE --user user11:V4BTUHV5  --insecure
https://wg31.washington.ibm.com:9443/zosConnect/services/cscvincSelectService
{"errorMessage":"BAQR0434W: The zosConnectAuthorization interceptor encountered an error while processing a
request for service cscvincSelectService.","erro                                                          rm
the request."}

c:\z\passticket>curl -X PUT --user user21:QLTNYH
https://wg31.washington.ibm.com:9443/zosConnect/
{"zosConnect":{"serviceName":"cscvincSelectServi
1.0","version":"1.0.0","serviceURL":"https://wg3                                                          erv
ice","serviceInvokeURL":"https://wg31.washington.ibm.com:9443/zosConnect/services/cscvincSelectService?action=i
nvoke","dataXformProvider":"zosConnectWVXform-1.0","serviceStatus":"Stopped"}}

c:\z\passticket>curl -X DELETE --user user21:QLTNYHS7  --insecure
https://wg31.washington.ibm.com:9443/zosConnect/services/cscvincSelectService
{"name":"cscvincSelectService"}
```

```
<zosconnect_services>
   <service name="cscvincSelectService"
      adminGroup="CSCADMIN"  operationsGroup="CSCOPERS"
      invokeGroup="CSCINVKE" readerGroup="CSCREADR"/>
</zosconnect_services>
```


mitchj@us.ibm.com

Tech-Tip: z/OS Connect Authorization Invoke Sample

```
c:\z\passticket>curl -X GET --user user11:MPSIJ0F2 --header "Content-Type: application/json"
--insecure https://wg31.washington.ibm.com:9443/zosConnect/apis/cscvinc
{"errorMessage":"BAQR0436W: The zosconnectAuthorization interceptor encountered an error
while processing a request for API cscvinc.","errorDetails":"BAQR0409W: User user11 is not
authorized to perform the request."}

c:\z\passticket>curl -X GET --user user13:LSHWBUAK --header "Content-Type: application/json"
--insecure  https://wg31.washington.ibm.com:9443/cscvinc/employee/000100
{"errorMessage":"BAQR0436W: The zosConnectAuthorization interceptor encountered an error
while processing a request for API cscvinc.","errorDetails":
"BAQR0409W: User user13 is not authorized to perform the request."}

c:\z\passticket>curl -X GET --user user21:C8CGF4KA --header "Content-Type: application/json"
--insecure  https://wg31.washington.ibm.com:9443/cscvinc/employee/000100
{"cscvincSelectServiceOperationResponse":{"cscvincContainer":{"response":{"CEIBRESP2":0,"USE
RID":"CICSUSER","filea":{"date":"26 11 81","amount":"$0100.11","address":"SURREY,
ENGLAND","phoneNumber":"32156778","name":"S. D.
BORMAN","employeeNumber":"000100"},"CEIBRESP":0}}}}

c:\z\passticket>curl -X GET --user user31:VWIJW74F --header "Content-Type: application/json"
--insecure  https://wg31.washington.ibm.com:9443/cscvinc/employee/000100
{"errorMessage":"BAQR0436W: The zosConnectAuthorization interceptor encountered an error
while processing a request for API cscvinc.","errorDetails":
"BAQR0409W: User user33 is not authorized to perform the request."}
```

```
<zosconnect_zosConnectAPIs>
    <zosConnectAPI name="cscvinc"
        adminGroup="CSCADMIN"  operationsGroup="CSCOPERS"
        invokeGroup="CSCINVKE" readerGroup="CSCREADR"/>
</zosconnect_zosConnectAPIs>
```


mitchj@us.ibm.com

Tech-Tip: Solution for z/OS Connect Authorization Levels

```
<zosconnect_zosConnectManager
         globalInterceptorsRef="interceptorList_g"
         globalAdminGroup="GMADMIN" globalOperationsGroup="GMOPERS,CSCOPERS,DB2OPERS"
         globalInvokeGroup="GMINVOKE" globalReaderGroup="GMREADER"/>

<zosconnect_zosConnectAPIs>
    <zosConnectAPI name="cscvinc" operationsGroup="CSCOPERS" invokeGroup="CSCINV"/>
    <zosConnectAPI name="db2employee" operationsGroup="DB2OPERS" invokeGroup="DB2INVKE"/>
</zosconnect_zosConnectAPIs>

<zosconnect_services>
    <service name="cscvincSelectService" operationsGroup="CSCOPERS" invokeGroup="CSCINV"/>
    <service name="selectEmployee" operationsGroup="DB2OPERS" invokeGroup="DB2INVKE"/>
</zosconnect_services>

<zosconnect_apiRequesters>
    <apiRequester name="cscvincSelectService" operationsGroup="CSCOPERS" invokeGroup="CSCINV"/>
    <apiRequester name="selectEmployee" operationsGroup="DB2OPERS" invokeGroup="DB2INVKE"/>
</zosconnect_apiRequesters>
```

- Now members of groups GMADMIN, GMOPERS,CSCOPERS, DB2OPERS and GMREADER can connect to a z/OS server from the API toolkit.

When a partial list of subordinate groups are provided, the corresponding default global groups for the absence groups are used.

mitchj@us.ibm.com

Tech-Tip: z/OS Connect Authorization Get Sample

Deploy a new API Requester

```
c:\z\passticket>curl -X POST --user user12:ID1V0B4C  --data-binary @cscvinc.ara  --header "Content-Type:
application/zip" --insecure https://wg31.washington.ibm.com:9443/zosConnect/apiRequesters
{"name":"cscvincapi_1.0.0","version":"1.0.0","description":"","status":"Started","apiRequesterUrl":"https:\/\/w
g31.washington.ibm.com:9443\/zosConnect\/apiRequesters\/cscvincapi_1.0.0","connection":"cscvincAPI”}
```

This API requester details is now accessible by members of GMREADER but not CSCREADR

```
c:\z\passticket>curl -X GET --user user14:XQIC6GB3 --header "Content-Type: application/json" --insecure
https://wg31.washington.ibm.com:9443/zosConnect/apiRequesters/cscvincapi_1.0.0
{"name":"cscvincapi_1.0.0","version":"1.0.0","description":"","status":"Started","apiRequesterUrl":"https:\/\/w
g31.washington.ibm.com:9443\/zosConnect\/apiRequesters\/cscvincapi_1.0.0","connection":"cscvincAPI"}

c:\z\passticket>curl -X GET --user user24:UFX1XUKB --header "Content-Type: application/json" --insecure
https://wg31.washington.ibm.com:9443/zosConnect/apiRequesters/cscvincapi_1.0.0
{"errorMessage":"BAQR1176W: The zosConnectAuthorization interceptor encountered an error while processing a
request for API requester cscvincapi_1.0.0."} *
```

This API requester is now managed by members of CSCOPERS and GMADMIN

```
c:\z\passticket>curl -X PUT --user user12:N2AHSG71  --insecure
https://wg31.washington.ibm.com:9443/zosConnect/apiRequesters/cscvincapi_1.0.0?status=stopped
{"errorMessage":"BAQR1176W: The zosConnectAuthorization interceptor encountered an error while processing a
request for API requester cscvincapi_1.0.0."}

c:\z\passticket>curl -X PUT --user user22:RVWKLNZV  --insecure
https://wg31.washington.ibm.com:9443/zosConnect/apiRequesters/cscvincapi_1.0.0?status=stopped
{"name":"cscvincapi_1.0.0","version":"1.0.0","description":"","status":"Stopped","apiRequesterUrl":"https:\/\/w
g31.washington.ibm.com:9443\/zosConnect\/apiRequesters\/cscvincapi_1.0.0","connection":"cscvincAPI"}

c:\z\passticket>curl -X DELETE --user user22:WAHOKIGF  --insecure
https://wg31.washington.ibm.com:9443/zosConnect/apiRequesters/cscvincapi_1.0.0
{"name":"cscvincapi_1.0.0"}q
```

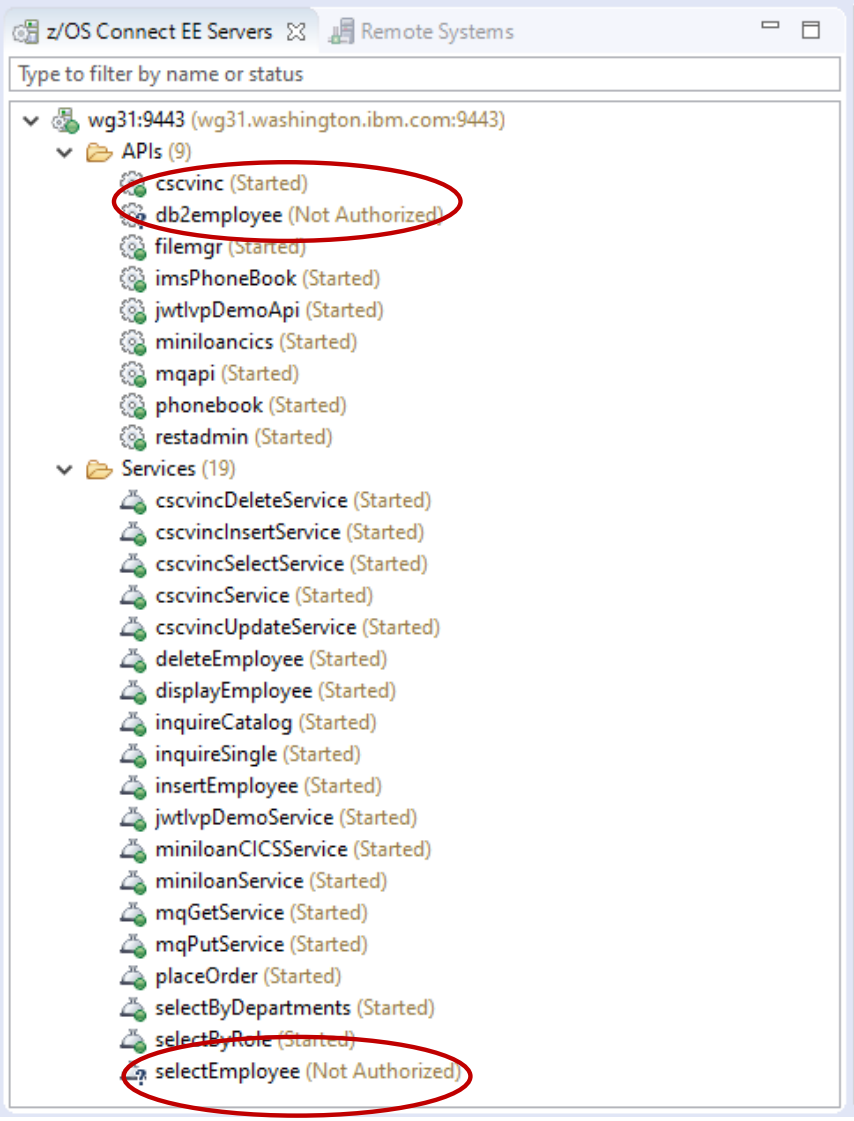
*BAQR0430W: User user24 is not authorized to perform the request on resource /zosConnect/apiRequesters/cscvincapi_1.0.0

mitchj@us.ibm.com

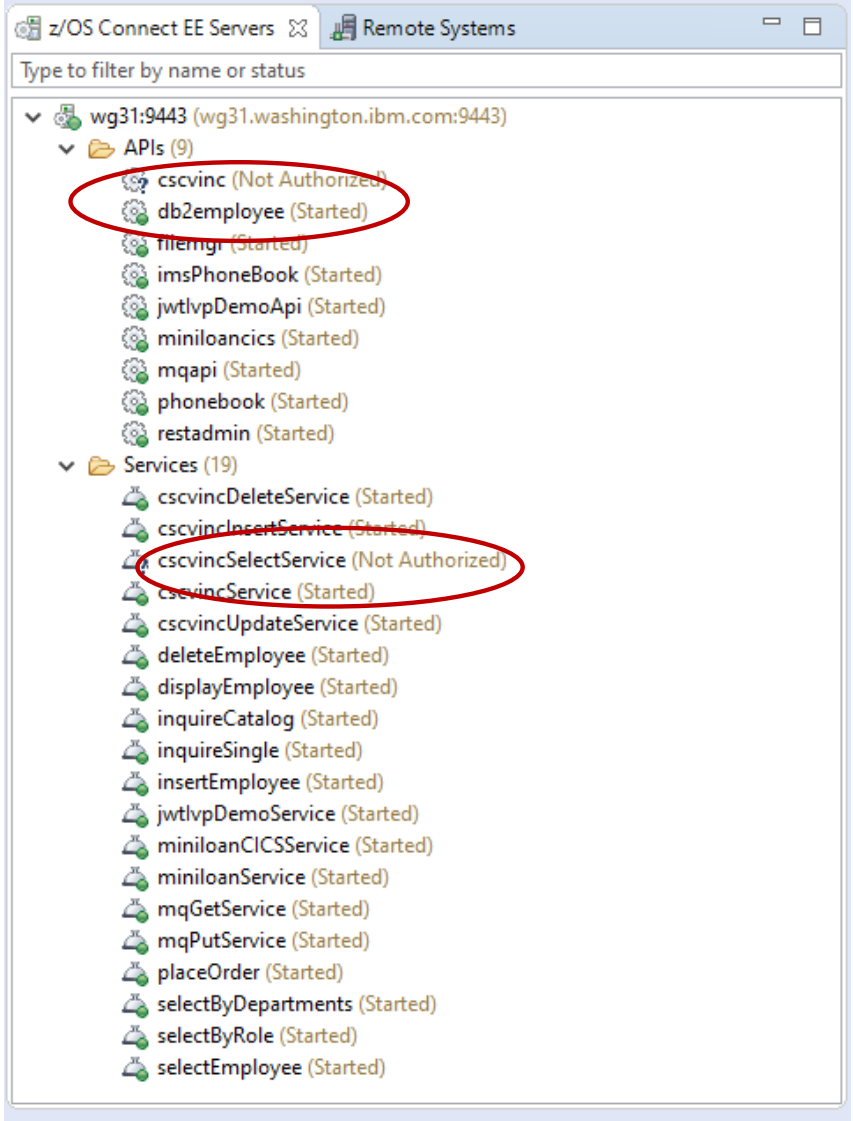
Tech-Tip: z/OS Toolkit and authorization status

Now members of CSCOPERS and DB2OPERS can connect to a server from the API Toolkit

CSCOPERS



DB2OPERS



mitchj@us.ibm.com

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=“IMScredentials">
<authData id=“IMScredentials" user= "identity" password= "password"/>

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

<zosconnect_cicsIpicConnection id="CICS"  authDataRef=“CICScredentials"/>
<zosconnect_authData id=“CICScredentials" 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.

mitchj@us.ibm.com

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

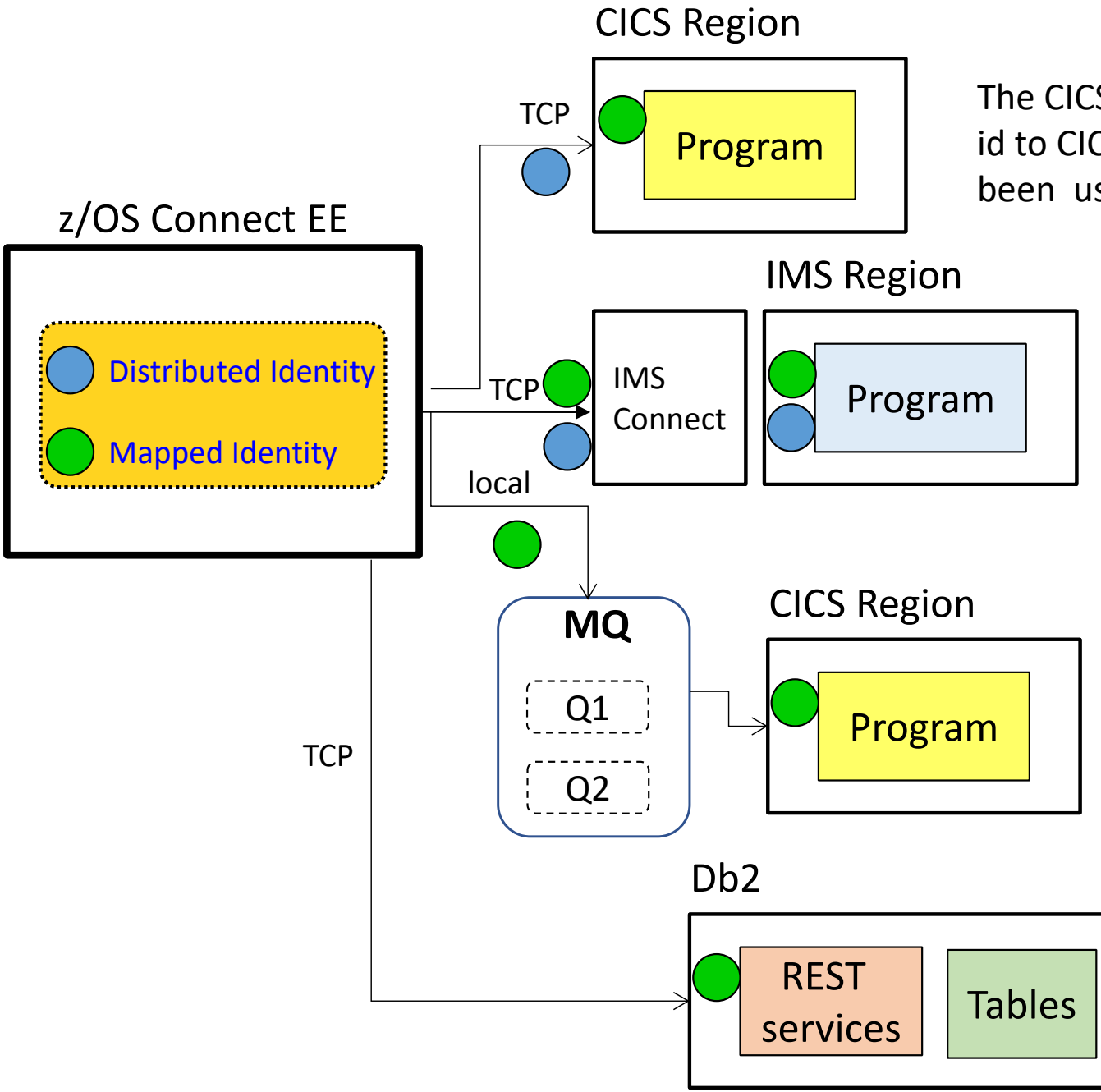
    {hash}ATAAAAAIcqTmHn5qZahAAAAAIMjzy+hP8YFaIO6LiCreVe4etRLUS9a25eVuYtx6WKiv

See the WebSphere Application Server for z/OS Liberty at URL:

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

mitchj@us.ibm.com

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)

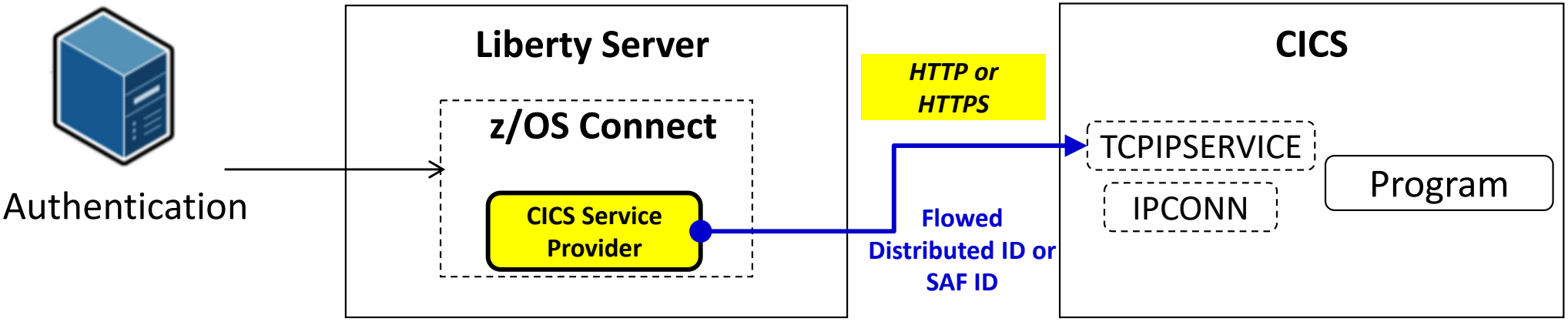
The IMS SP can request a PassTicket and send the PassTicket to IMS Connect (available in V3.0.33)

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)

mitchj@us.ibm.com

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.

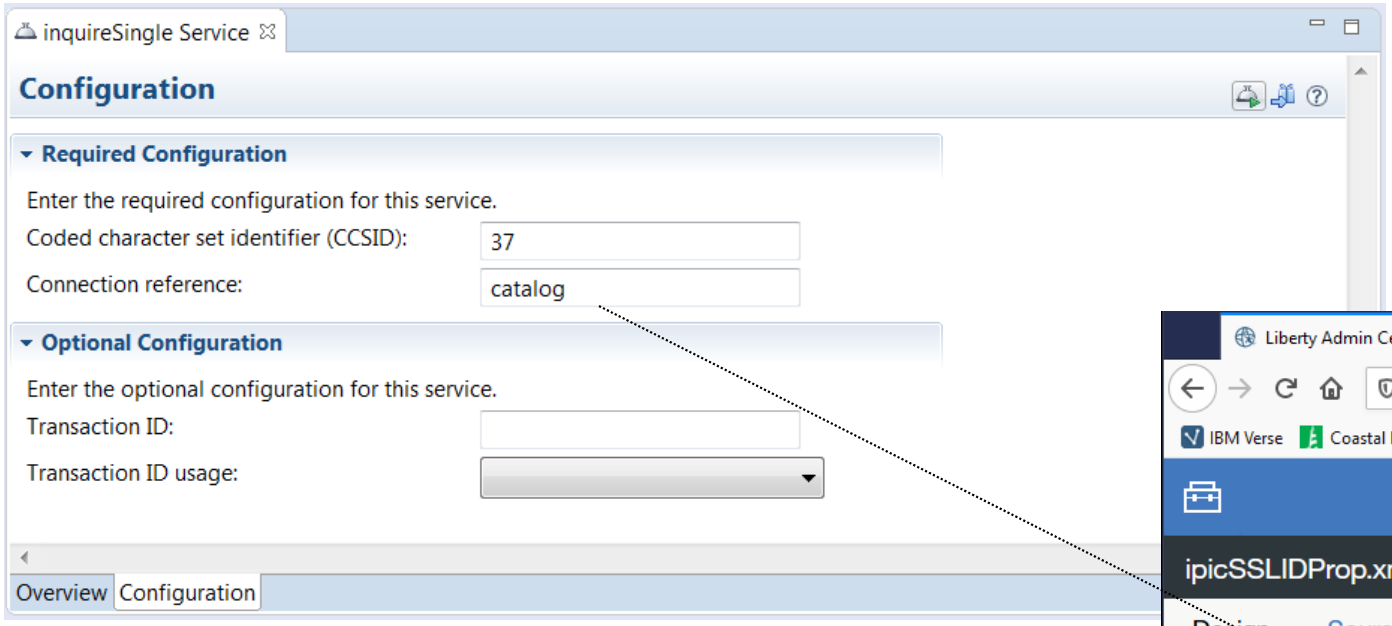
If a SAF ID is used for authentication (e.g., basic authentication with a SAF registry) then the SAF ID is passed to CICS.

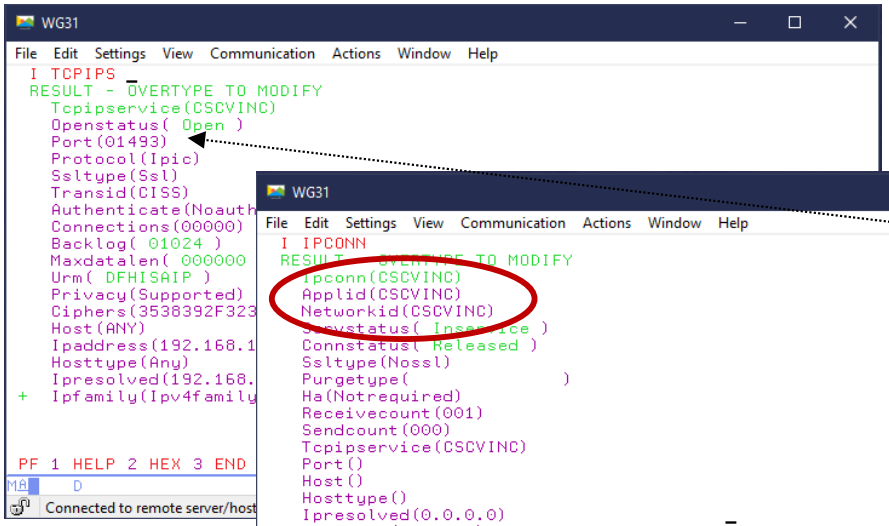
mitchj@us.ibm.com

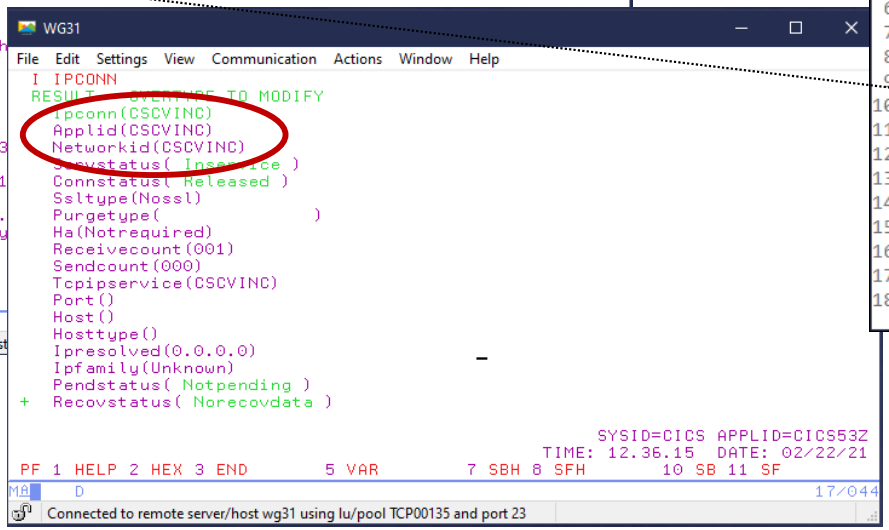
CICS IPIC (server.xml)

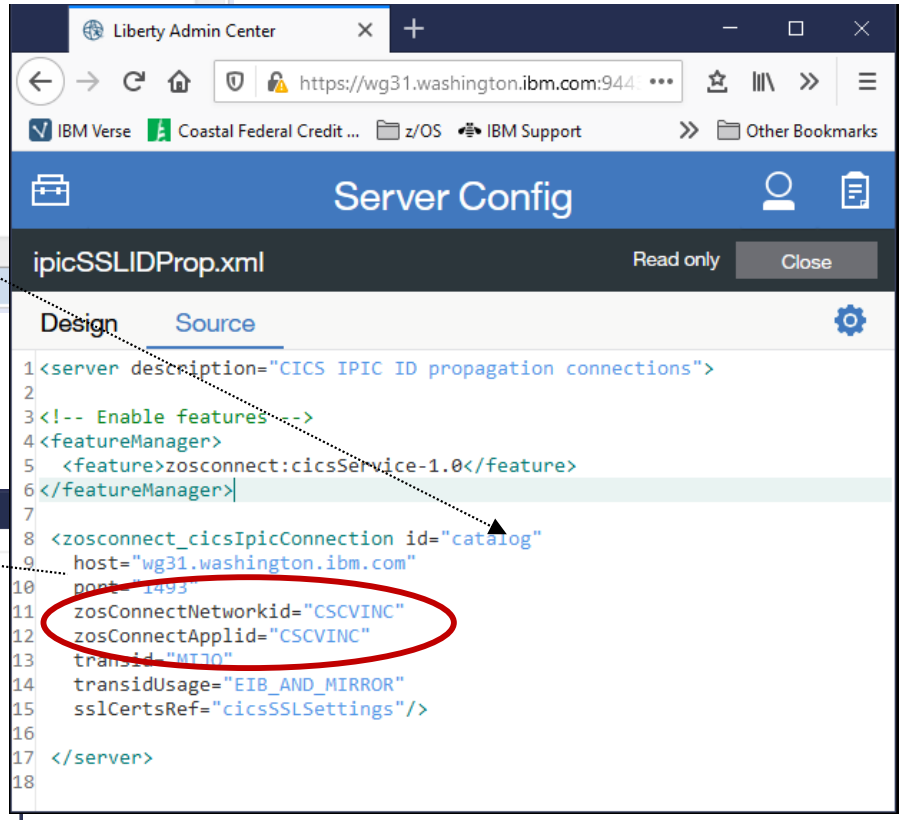
z/OS Connect EE

The server.xml file is the key configuration file:









Define identity propagation to CICS

mitchj@us.ibm.com

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"/>
```


mitchj@us.ibm.com

Identity Propagation and CICS High Availability

The service archive files use the following *Connection reference* values:

- cscvinc
- catalog
- miniloan

The CICS IPCONN resources that correspond to the zosconnect_cicsIpicConnection configuration elements must be dedicated to a z/OS Connect server and can not be reused.

baqsvr1's bootstrap.properties

```
ipicPort=1491
cicsHost=dvipa.washington.ibm.com
serverPrefix=baqsvr1
```

baqsvr2's bootstrap.properties

```
cicsHost=dvipa.washington.ibm.com
ipicPort=1491
serverPrefix=baqsvr2
```

ipicIDProp.xml

```
<zosconnect_cicsIpicConnection id="cscvinc"
  host="${cicsHost}"
  zosConnectNetworkid="${wlp.server.name}"        → baqsvr1 or baqsvr2
  zosConnectApplid="${wlp.server.name}"
  sharedPort="true" port="${ipicPort}"/>
<zosconnect_cicsIpicConnection id="catalog"
  host="${cicsHost}"
  zosConnectNetworkid="${serverPrefix}C"          → baqsvr1C or baqsvr2C
  zosConnectApplid="${serverPrefix}C"
  sharedPort="true" port="${ipicPort}"/>
<zosconnect_cicsIpicConnection id="miniloan"
  host="${cicsHost}"
  zosConnectNetworkid="${serverPrefix}M"          → baqsvr1M or baqsvr2M
  zosConnectApplid="${serverPrefix}M"
  sharedPort="true" port="${ipicPort}"/>
```


mitchj@us.ibm.com

CICS IPCONN and TCPIPService resources for HA

CICS Specific TCPIPService - IPIC

```
TCpipservice    : IPIC[1]
GROup           : SYSPGRP
Urm          ==> DFHISAIP
POrtnumber   ==> 01492
STatus       ==> Open
PROtocol     ==> IPic
TRansaction  ==> CISS
Host         ==> ANY
Ipaddress    ==> ANY
SPeciftcps   ==>
```

CICS Generic TCPIPService - IPICG

```
TCpipservice    : IPICG[1]
GROup           : SYSPGRP
Urm          ==> DFHISAIP
POrtnumber   ==> 01491
STatus       ==> Open
PROtocol     ==> IPic
TRansaction  ==> CISS
Host         ==> ANY
Ipaddress    ==> ANY
SPeciftcps   ==> IPIC
```

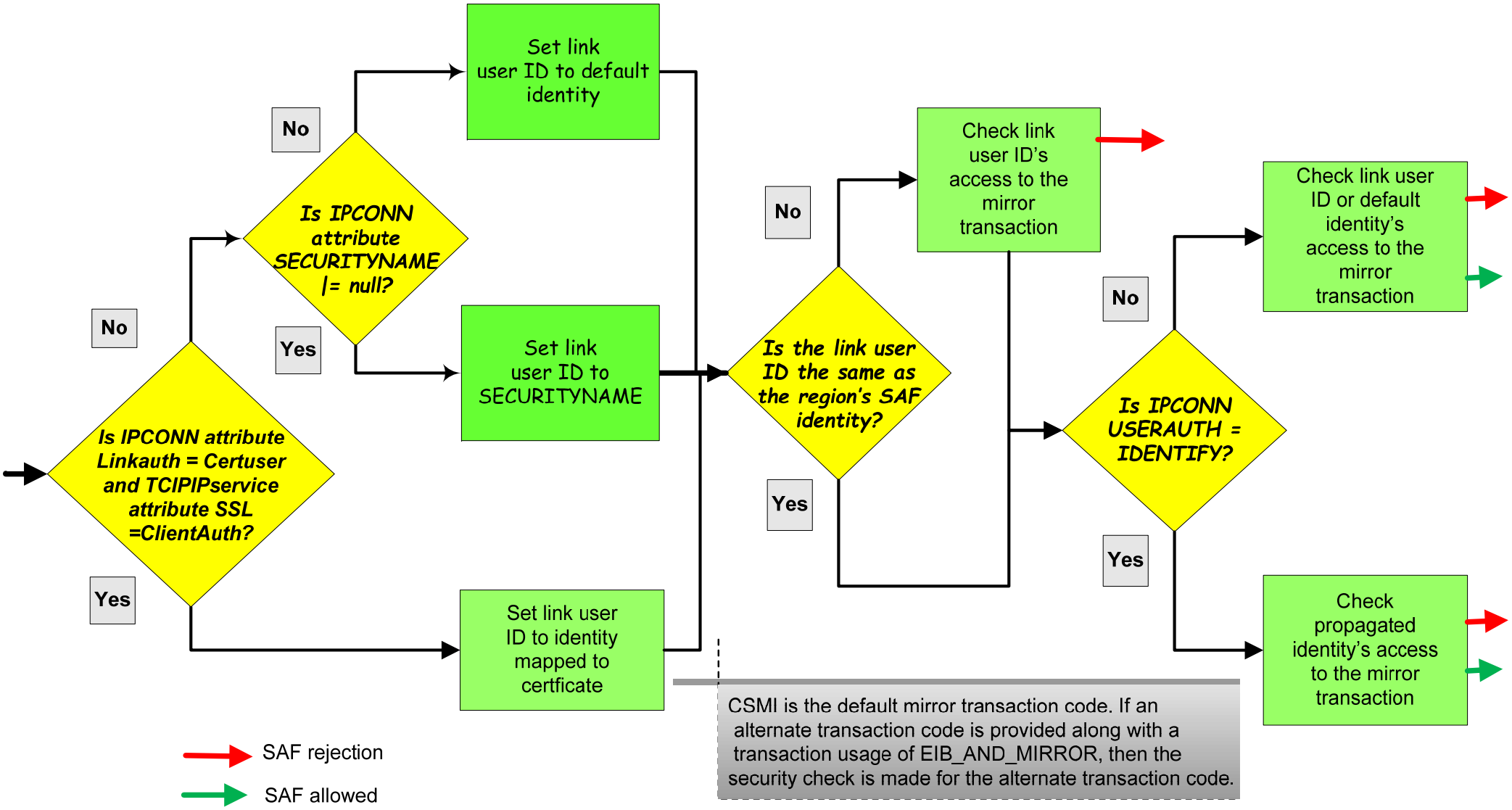
A client connects first to the CICS region's generic port (1491) and then the CICS region redirects the client to the region's specific port (1492).

```
I IPCONN ACQ
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Ipc(BAQSVR1 ) App(BAQSVR1 ) Net(BAQSVR1 ) Ins Acq Nos
        Rece(001) Sen(000) Tcp(IPIC     )
 Ipc(BAQSVR1C) App(BAQSVR1C) Net(BAQSVR1C) Ins Acq Nos
        Rece(001) Sen(000) Tcp(IPIC     )
 Ipc(BAQSVR1M) App(BAQSVR1M) Net(BAQSVR1M) Ins Acq Nos
        Rece(001) Sen(000) Tcp(IPIC     )
 Ipc(BAQSVR2 ) App(BAQSVR2 ) Net(BAQSVR2 ) Ins Acq Nos
        Rece(001) Sen(000) Tcp(IPIC     )
 Ipc(BAQSVR2C) App(BAQSVR2C) Net(BAQSVR2C) Ins Acq Nos
        Rece(001) Sen(000) Tcp(IPIC     )
 Ipc(BAQSVR2M) App(BAQSVR2M) Net(BAQSVR2M) Ins Acq Nos
        Rece(001) Sen(000) Tcp(IPIC     )
```

[1]CICS requires the specific TCPIPService be installed before the corresponding generic TCPIPService resource. TCPIPServices are installed in alphabetically order, so the name of specific service must be alphabetically prior to the name of the generic TCPIPService.

mitchj@us.ibm.com

Tech/Tip: CICS IPIC Security



PassTickets and IMS

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

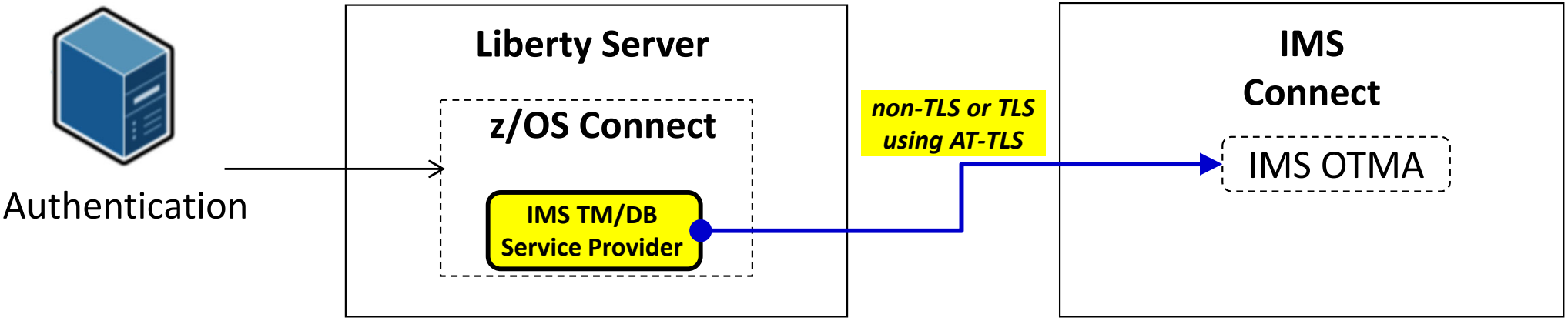
```
HWS=(ID=IMS15HWS,XIBAREA=100,RACF=Y,RRS=Y)
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,TIMEOUT=5000)
DATASTORE=(GROUP=OTMAGRP,ID=IVP1,MEMBER=HWSMEM,DRU=HWSYDRU0,
TMEMBER=OTMAMEM,APPL=IMSTMAPL)
ODACCESS=(ODBMAUTOCONN=Y,IMSPLEX=(MEMBER=IMS15HWS,TMEMBER=PLEX1),
     DRDAPORT=(ID=5555,PORTTMOT=6000),ODBMTMOT=6000,APPL=IMSDBAPL)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

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)

mitchj@us.ibm.com

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="/>
```

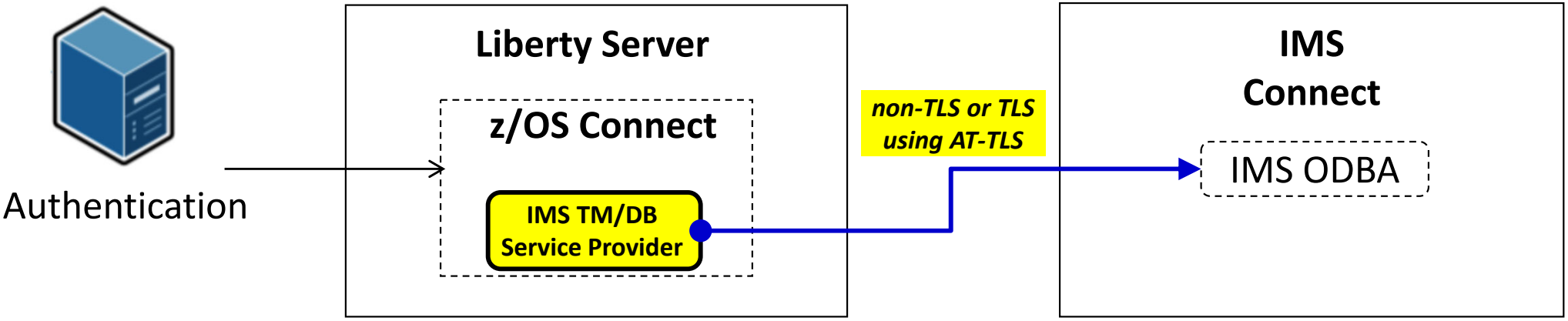
Specify a user identity and password to be used in the request to IMS Connect

```
<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

mitchj@us.ibm.com

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>
```

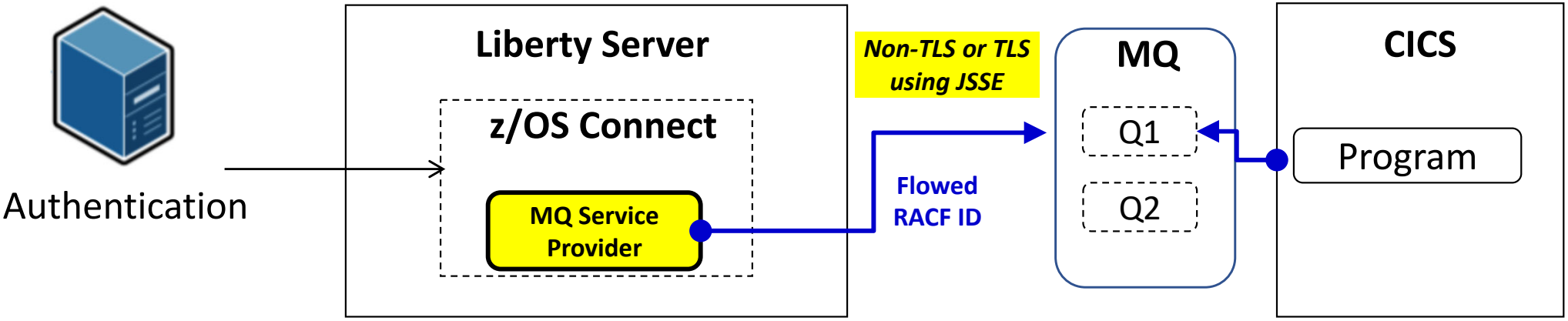
Specify a user identity and password to be used in the request to IMS Connect

```
<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

mitchj@us.ibm.com

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>
```

Define identity propagation to MQ

mitchj@us.ibm.com

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
```

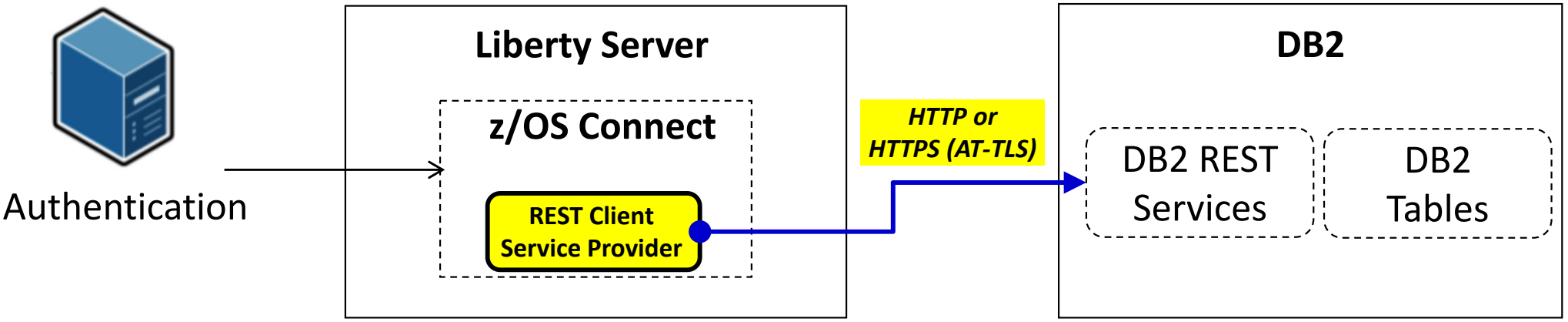
Which value should be used for applName is determine for use in RACF resources is determine as shown below.

- 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)
```


mitchj@us.ibm.com

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

mitchj@us.ibm.com

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

mitchj@us.ibm.com

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"/>
```

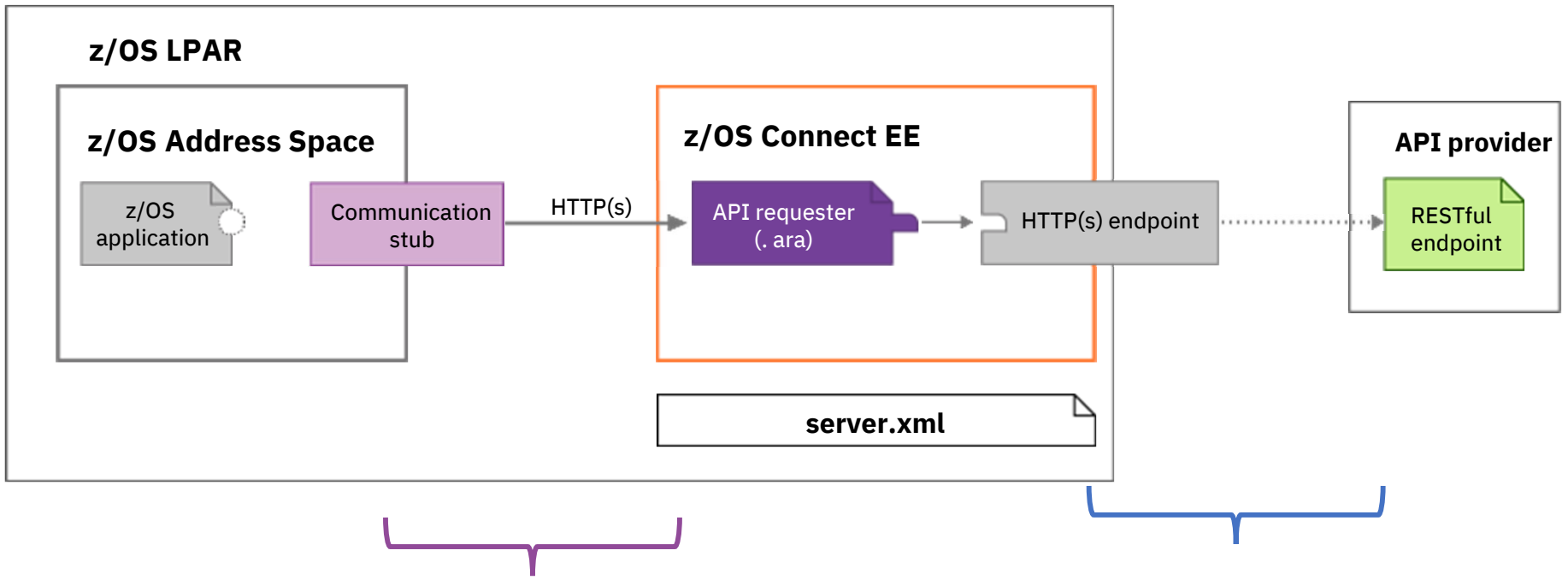
```
RDEFINE CBIND BBG.WOLA.ZCEEDVM.** UACC(NONE)
PERMIT BBG.WOLA.ZCEEDVM.** CLASS(CBIND)  ID(LIBSERV) ACC(READ)
SETROPTS RACLIST(CBIND) REFRESH
```


mitchj@us.ibm.com

z/OS Connect API Requester Security

Details

Authentication



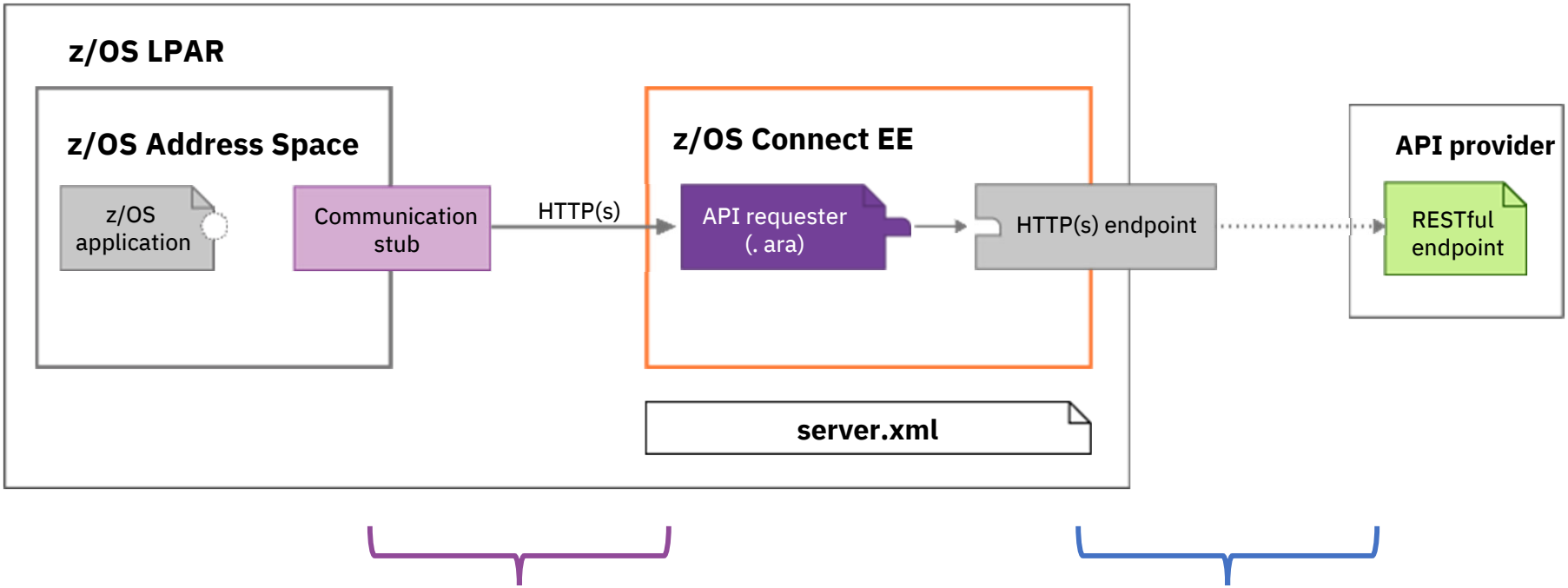
Options:

1. Basic Authentication
2. TLS Client/Server

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

mitchj@us.ibm.com

Encryption



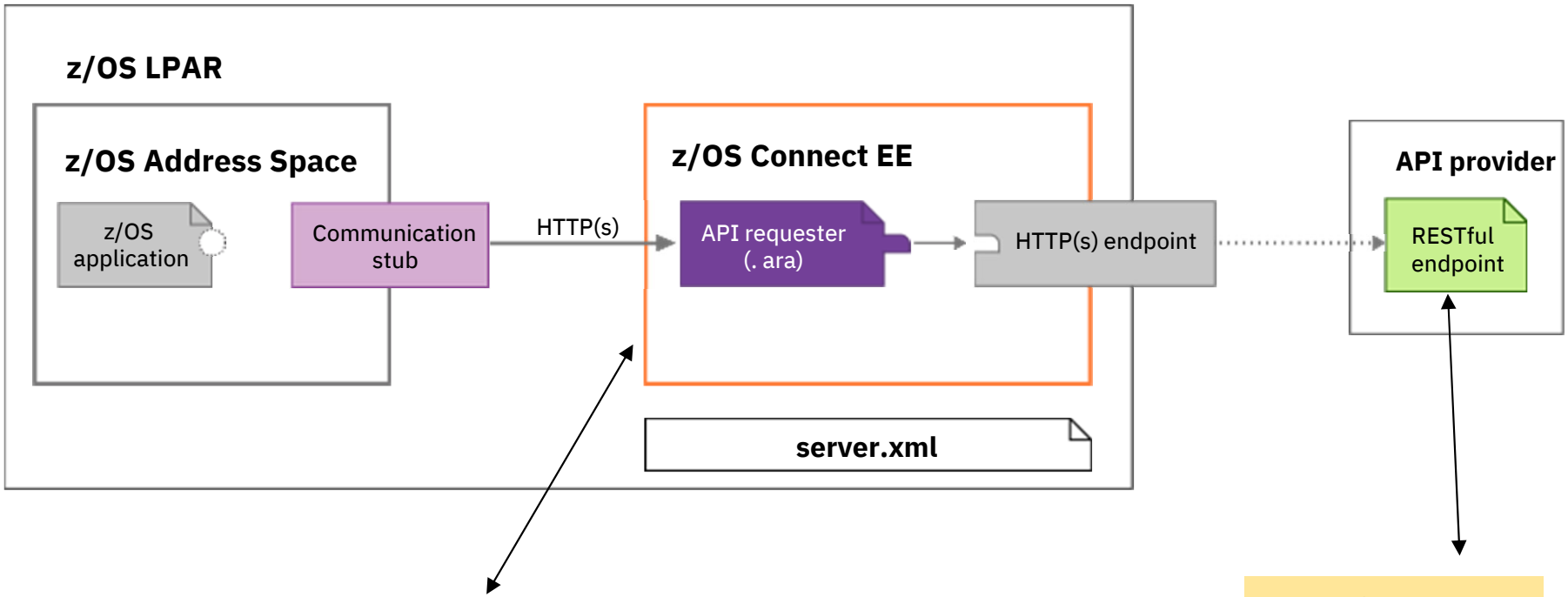
Options:

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

1. JSSE
2. AT-TLS

mitchj@us.ibm.com

Authorization



Authorization interceptor options:

- Global level
- API requester level

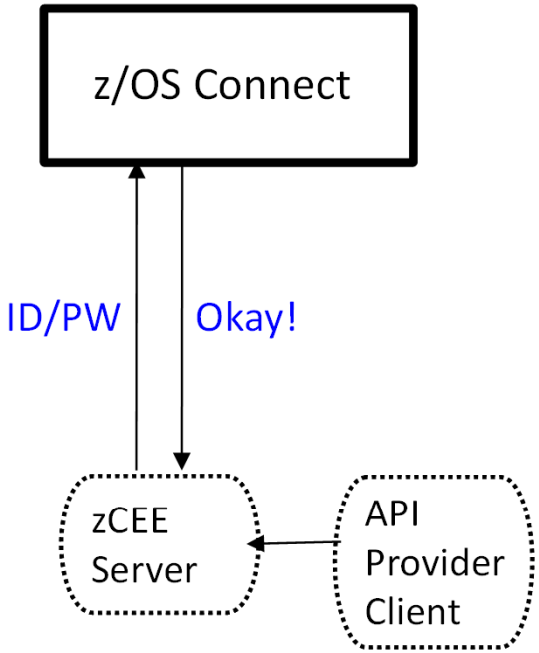
OAuth or JWT access token

mitchj@us.ibm.com

API Requester- API Provider Authentication

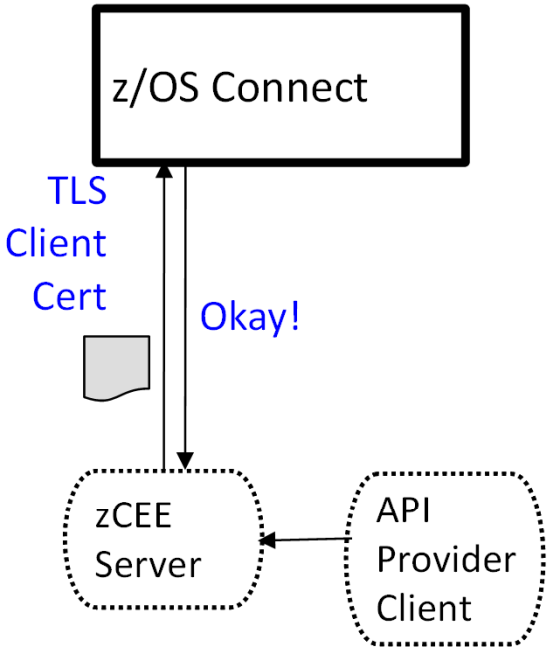
Several different ways this can be accomplished:

Basic Authentication



zCEE server supplies ID/PW or ID/PassTicket

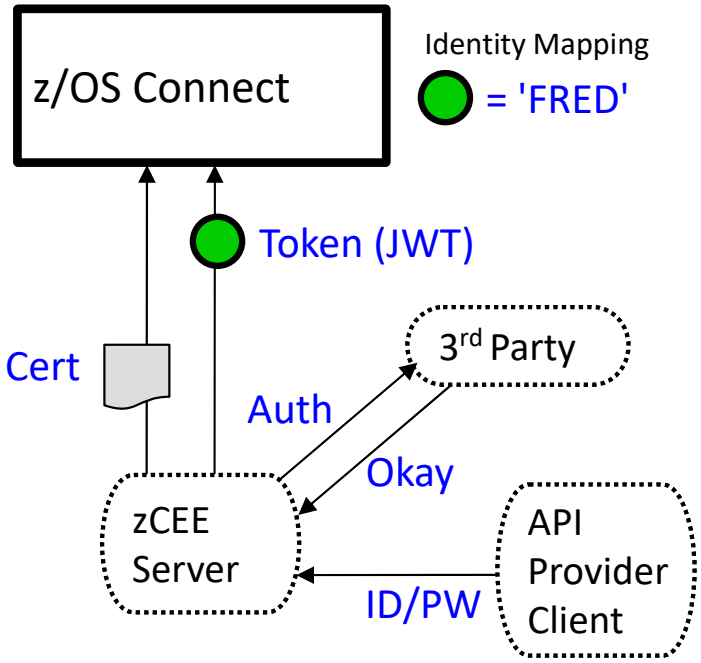
Client Certificate



Server requests a client certificate

zCEE supplies a client certificate

Third Party Authentication



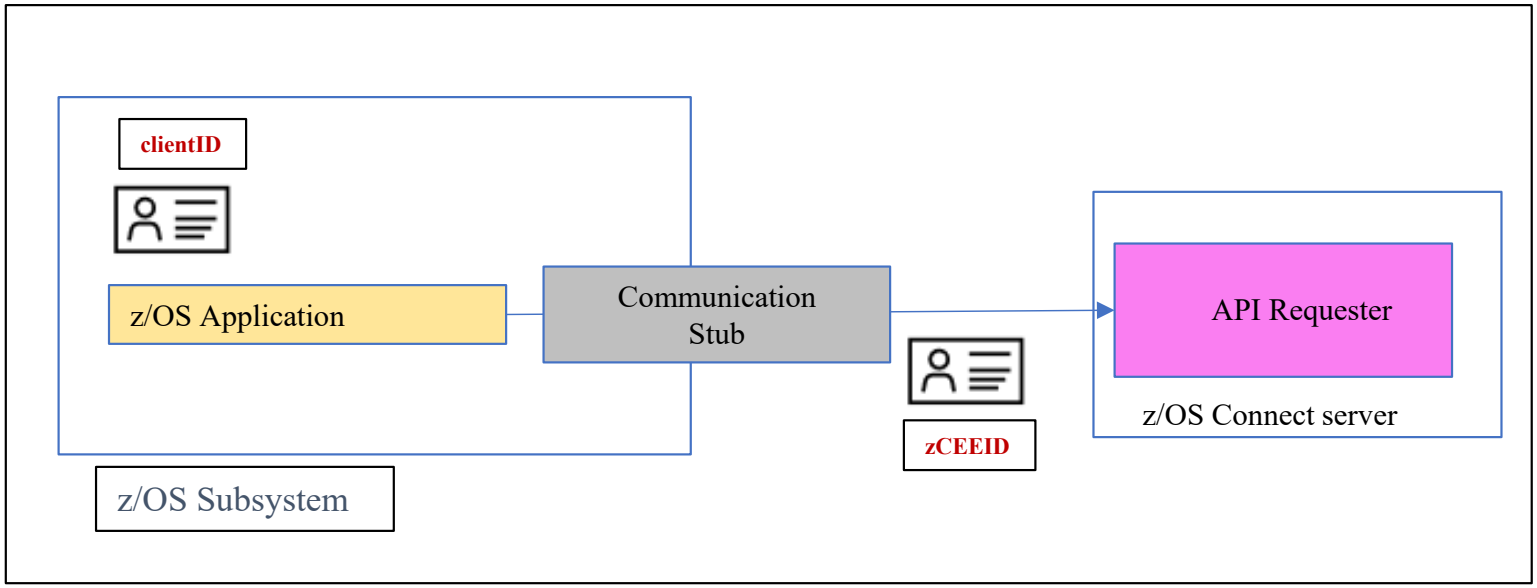
zCEE Server authenticates to 3rd party server

zCEE Server receives a trusted 3rd party token

Token flows to API Provider

mitchj@us.ibm.com

API Requester - basic authentication and 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>
```

mitchj@us.ibm.com

SAF SURROGAT Resources for identity assertion

- ***Enable the program control bit for Java shared object ifaedjreg64***

  su
  cd /usr/lib/java_runtime
  extattr +p libifaedjreg64.so

- ***Permit the server identity to the required FACILITY resource***

  PERMIT BPX.SERVER CLASS(FACILITY) ID(LIBSERV) ACCESS(READ)
  SETROPTS RACLIST(FACILITY) REFRESH

- ***Define a SURROGAT profile for the asserted identity and permit access to connection identity***

  RDEFINE SURROGAT clientID.BAQASSRT UACC(NONE) OWNER(SYS1)
  PERMIT clientID.BAQASSRT CLASS(SURROGAT) ACCESS(READ) ID(zCEEID)

  OR

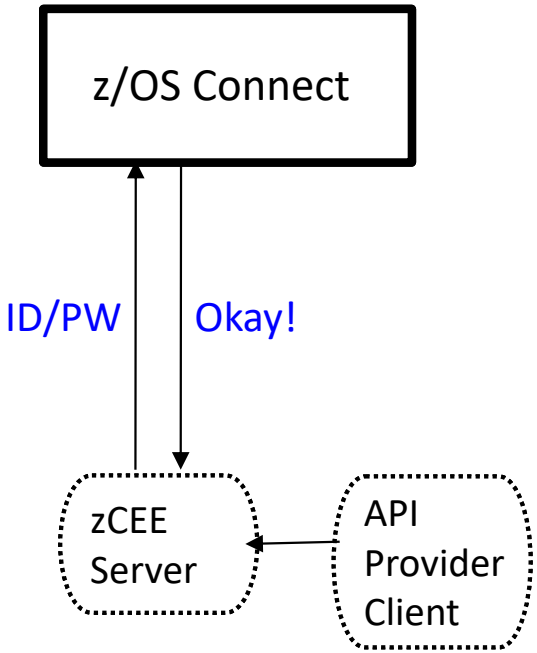
  RDEFINE SURROGAT *.BAQASSRT UACC(NONE) OWNER(SYS1)
  PERMIT *.BAQASSRT CLASS(SURROGAT) ACCESS(READ) ID(zCEEID)
  SETROPTS RACLIST(SURROGAT) REFRESH

mitchj@us.ibm.com

API Requester- API Provider Authentication

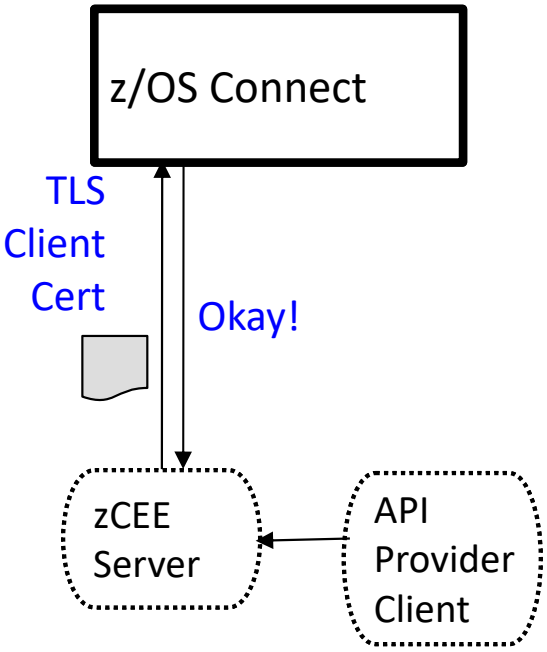
Several different ways this can be accomplished:

Basic Authentication



zCEE server supplies ID/PW or ID/PassTicket

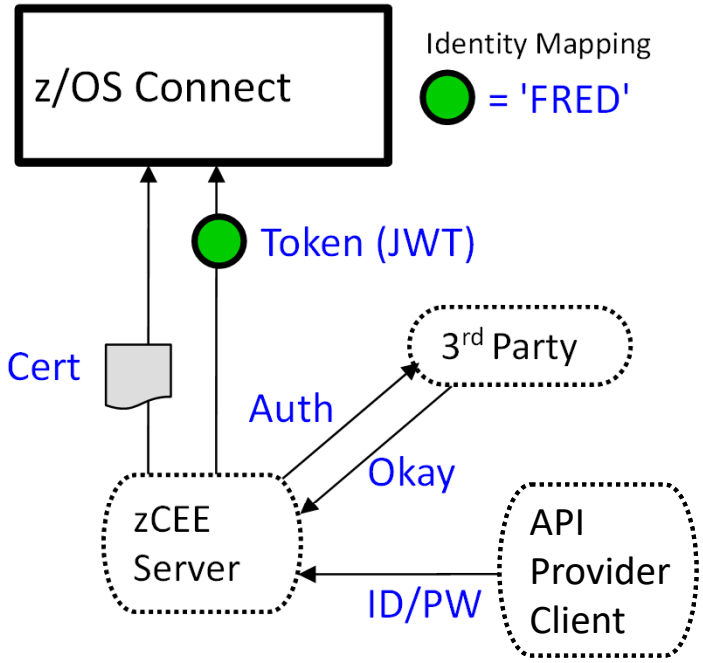
Client Certificate



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

mitchj@us.ibm.com

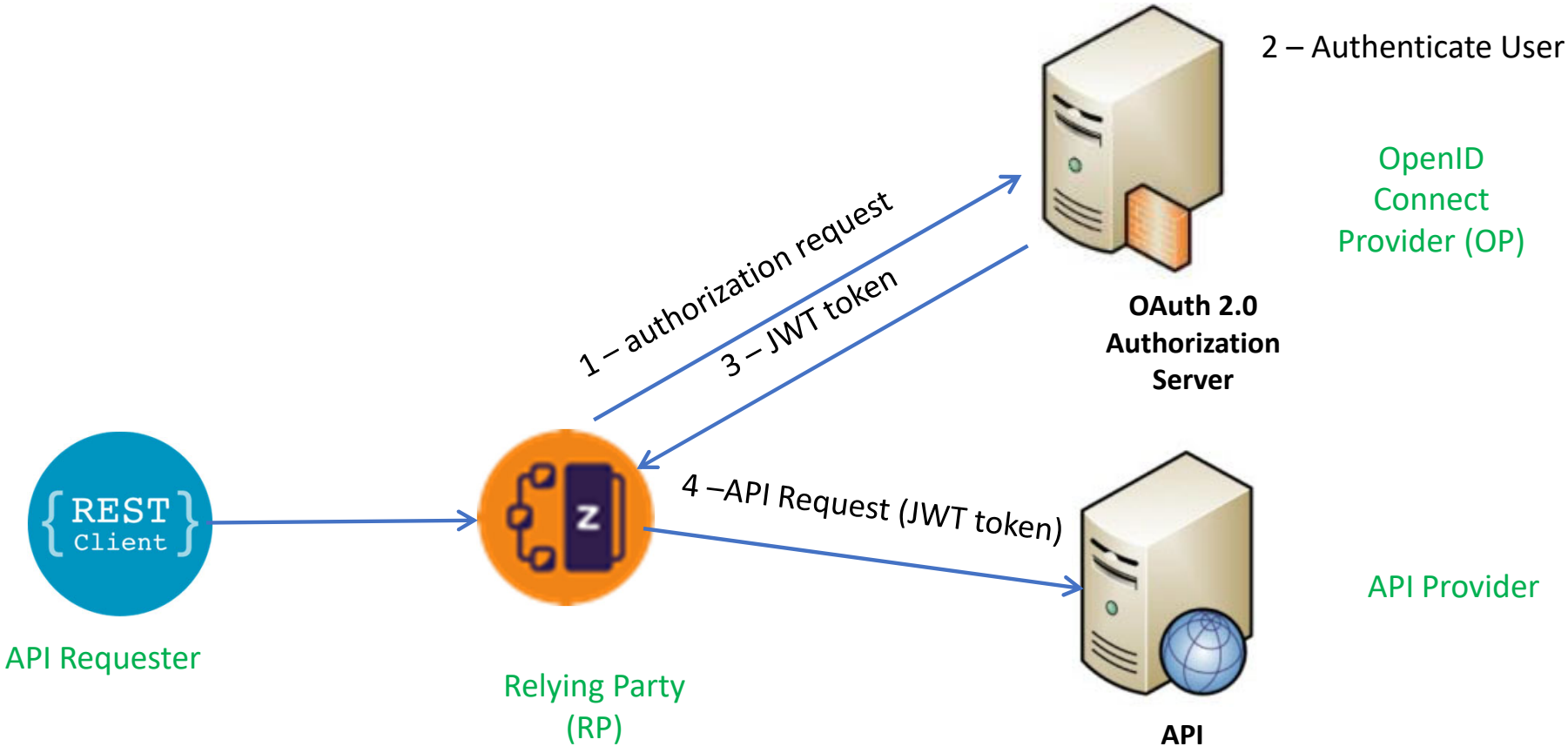
z/OS Connect API Requester - Token Support

z/OS Connect EE provides ***three*** 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 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.
3. Use the locally generated JWT support when you need to send a JWT that is generated by the z/OS Connect EE server.

mitchj@us.ibm.com

z/OS Connect OAuth Flow for API requester

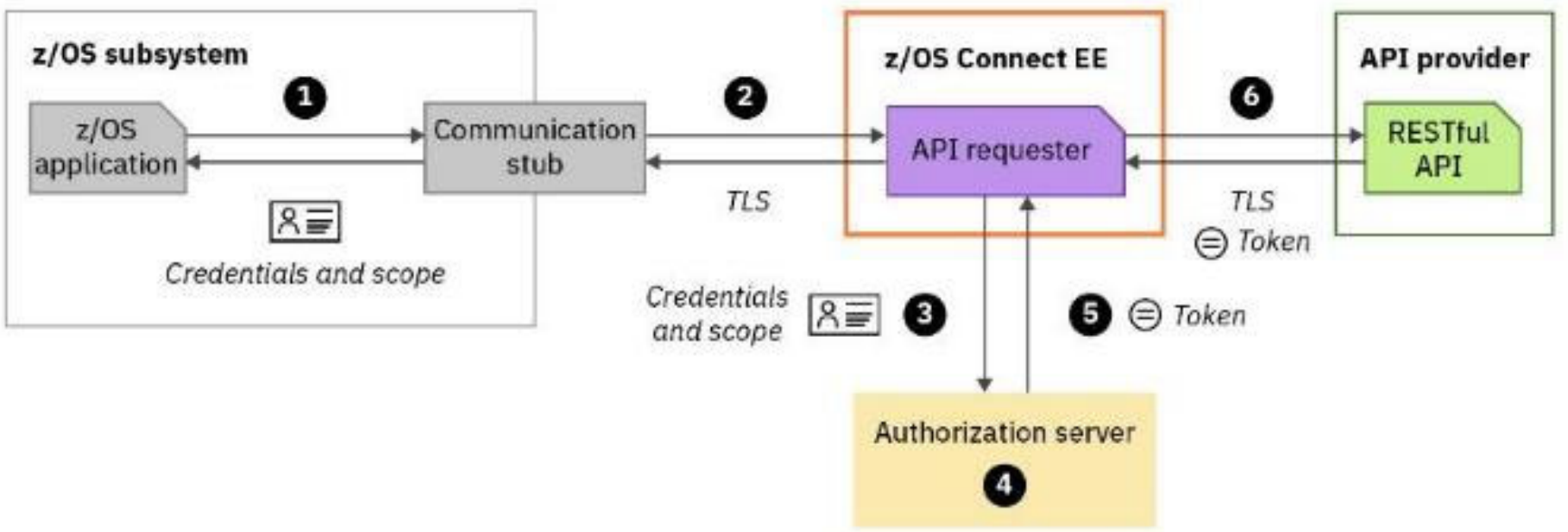


Grant Types:

- client_credentials
- password

mitchj@us.ibm.com

Calling an API with OAuth 2.0 support



mitchj@us.ibm.com

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.*

mitchj@us.ibm.com

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"/>
```


mitchj@us.ibm.com

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

mitchj@us.ibm.com

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

mitchj@us.ibm.com

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.

mitchj@us.ibm.com

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   )
```

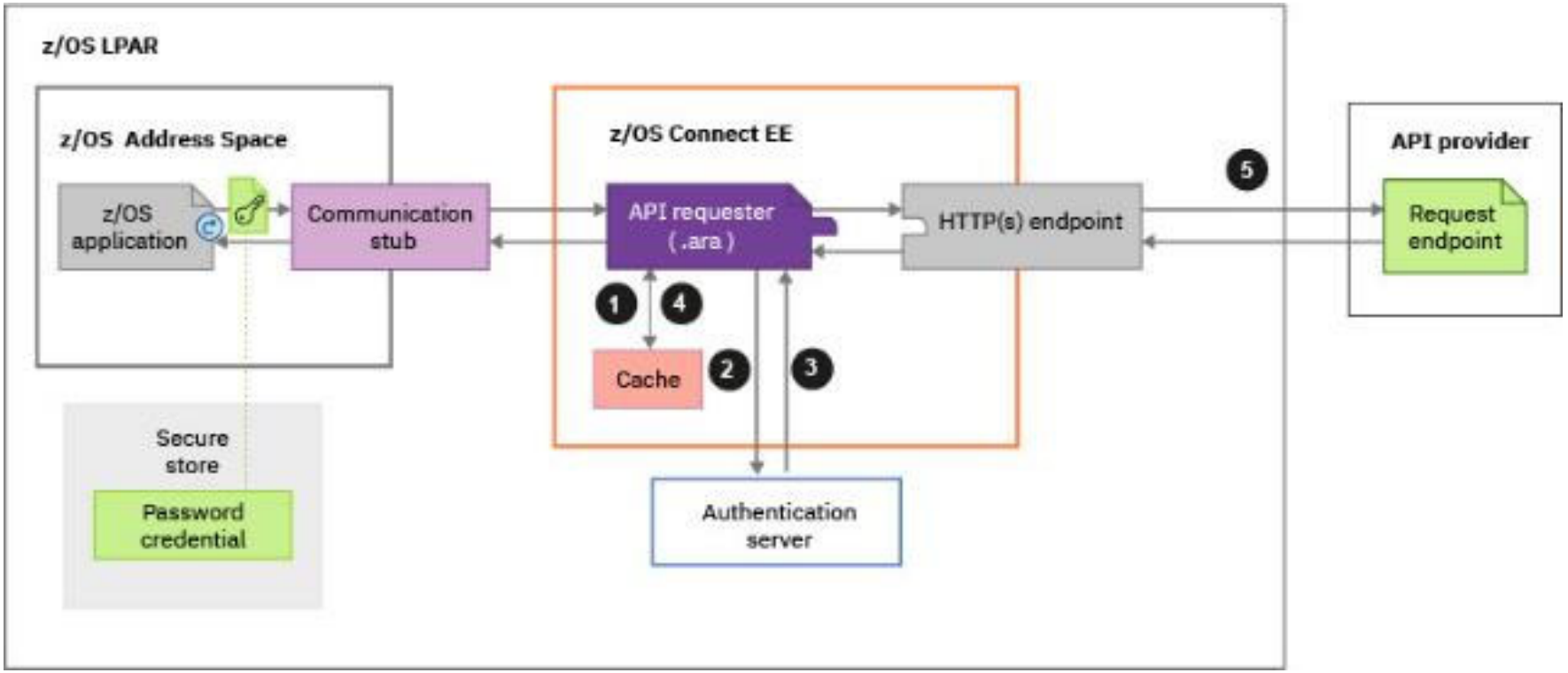
distuser2 is mapped to RACF identity USER2 which has no access to the EJBRole protecting z/OS Connect

mitchj@us.ibm.com

Calling an API with using a JWT custom flow

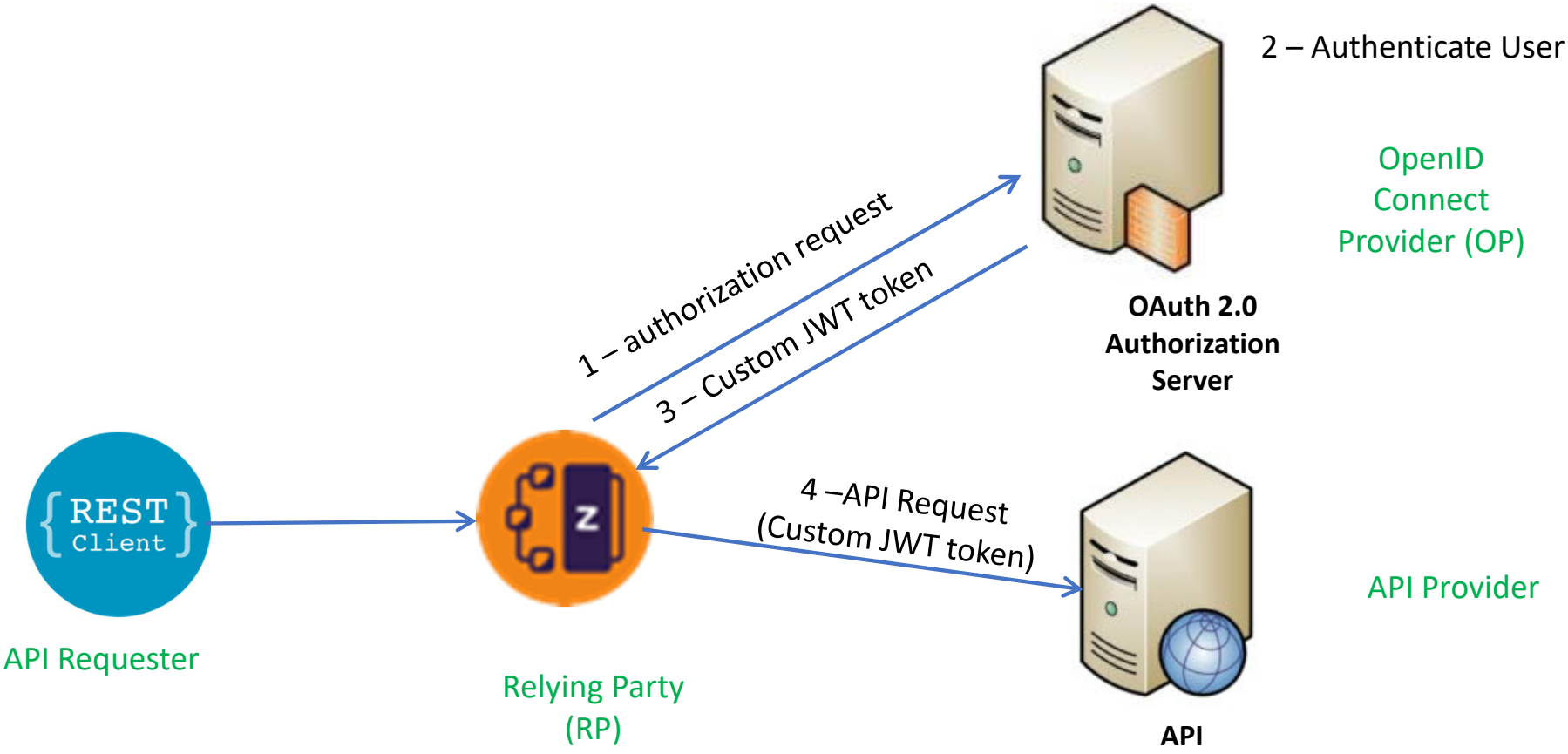
z/OS Connect provides two ways of calling an API secured with a JWT

- 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:
  - 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.



mitchj@us.ibm.com

z/OS Connect OAuth Customer Flow



mitchj@us.ibm.com

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.

mitchj@us.ibm.com

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

mitchj@us.ibm.com

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

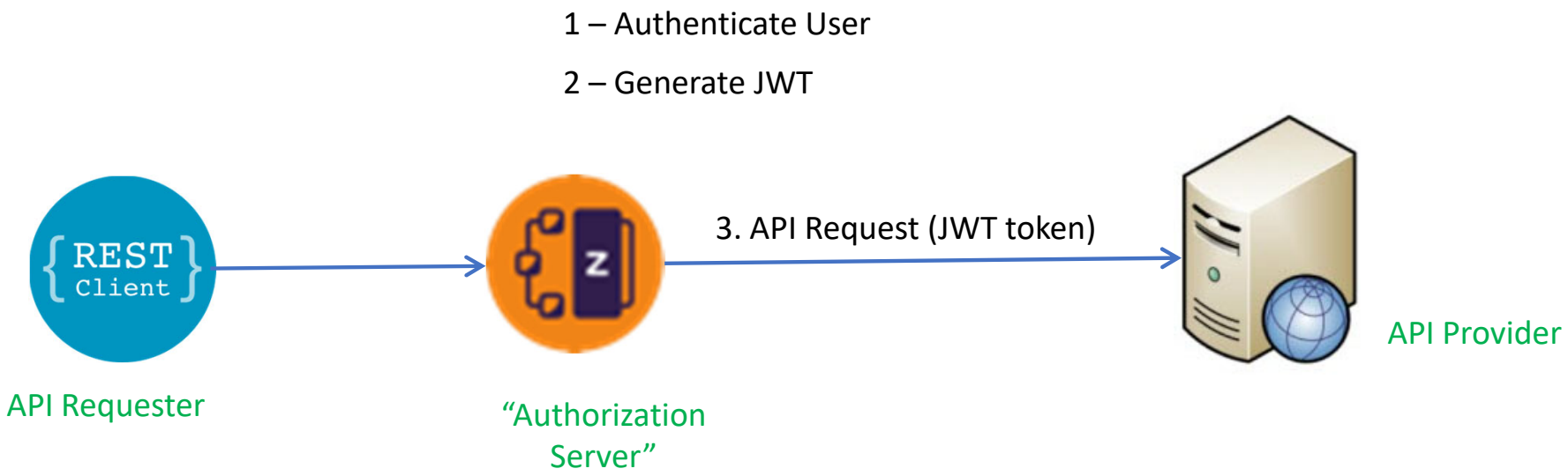
```
<tokenRequest credentialLocation="body"
   requestMethod="POST"
  // Use XML escaped characters in requestBody
  requestBody="{&amp;quot;apiuser&amp;quot;:&amp;quot;${userid}&amp;quot;,&amp;quot;apipassword&amp;quot;:&amp;quot;${password}&amp;quot;}" />
```

Response Token

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

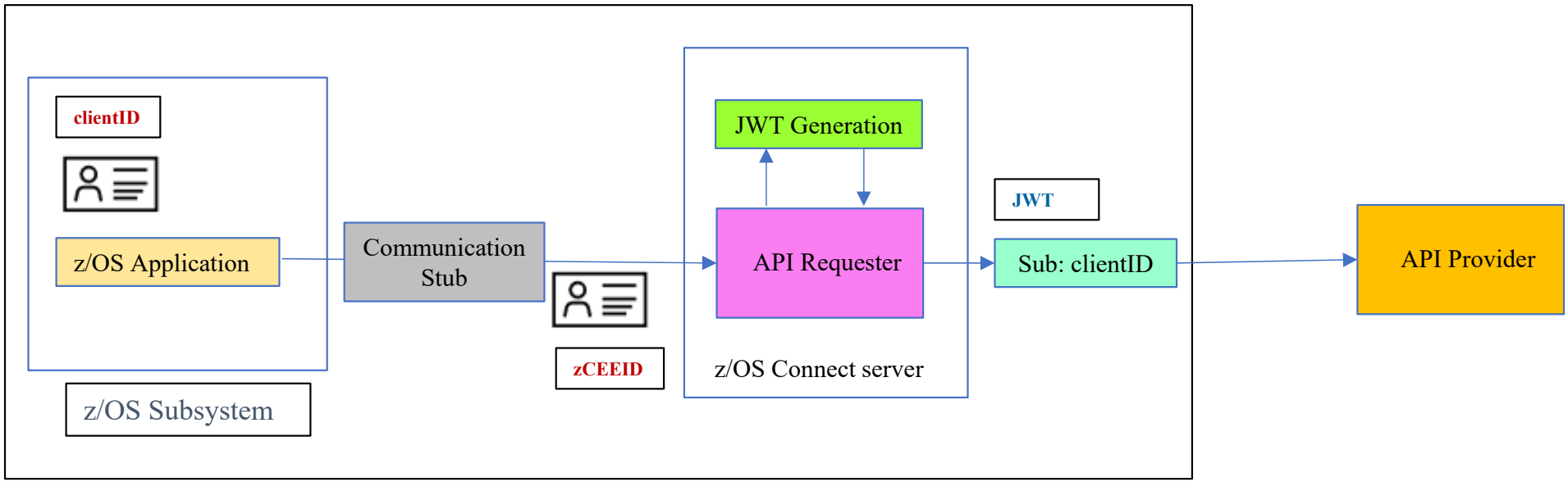

mitchj@us.ibm.com

z/OS Connect JWT Generation – V3.0.43



mitchj@us.ibm.com

API Requester – JWT Generation



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	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	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

mitchj@us.ibm.com

SAF SURROGAT Resources for identity assertion

- ***Enable the program control bit for shared object file ifaedjreg64***
  - ***su***
  - ***cd /usr/lib/java_runtime***
  - ***extattr +p libifaedjreg64.so***

- ***Permit the server's SAF identity to the required FACILITY resource***

  PERMIT BPX.SERVER CLASS(FACILITY) ID(LIBSERV) ACCESS(READ)
  SETROPTS RACLIST(FACILITY) REFRESH

- ***Define a SURROGAT profile for the asserted identity and permit access to the connection identity***

  RDEFINE SURROGAT clientID.BAQTOKEN UACC(NONE) OWNER(SYS1)
  PERMIT clientID.BAQTOKEN CLASS(SURROGAT) ACCESS(READ) ID(zCEEID)

  RDEFINE SURROGAT *.BAQTOKEN UACC(NONE) OWNER(SYS1)
  PERMIT *.BAQTOKEN CLASS(SURROGAT) ACCESS(READ) ID(zCEEID)
  SETROPTS RACLIST(SURROGAT) REFRESH

mitchj@us.ibm.com

Configuring JWT Generation support

```
<zosconnect_endpointConnection id="conn"
    host="http://api.server.com" port="8080"
    authenticationConfigRef="jwtConfig" />

<zosconnect_authTokenLocal id="jwtConfig"
   tokenGeneratorRef="jwtBuilder"
   header="Authorization" >
   <claims>{"name":"JohnSmith,
            "ID":"1234567890"}
   </claims>

<jwtBuilder id="jwtBuilder"
    scope="scope1"
    audiences="myApp1"
    jti="true"
    signatureAlgorithm="RS256"
    keyStoreRef="myKeyStore"
    keyAlias="jwtsigner"
    issuer="z/OS Connect EE Default"/>
```

One or more Public claim (e.g., *aud,exp,nbf,iat,jti*) or one or more private claims

The "sub" claim value will be application asserted user ID.

mitchj@us.ibm.com

Configuring JWT Generation support

```
 <zosconnect_endpointConnection id="conn1"
    host="http://api.server.com" port="8080"
    authenticationConfigRef="jwtConfig" />
<zosconnect_endpointConnection id="conn2"
    host="http://api.server.com" port="8080"
    authenticationConfigRef="jwtConfig" />
<zosconnect_authTokenLocal id="jwtConfig"
   tokenGeneratorRef="jwtBuilder"
   header="Authorization" >
   <claims>{"scope":"Scope1"}</claims>
<zosconnect_authTokenLocal id="jwtConfig"
   tokenGeneratorRef="jwtBuilder"
   header="Authorization" >
   <claims>{"scope":"Scope2"}</claims>
<jwtBuilder id="jwtBuilder"
    scope="scope"
    audiences="myApp1"
    jti="true"
    signatureAlgorithm="RS256"
    keyStoreRef="myKeyStore"
    keyAlias="jwtsigner"
    issuer="z/OS Connect EE Default"/>
```


mitchj@us.ibm.com

In this presentation we covered

- 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

Summary

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

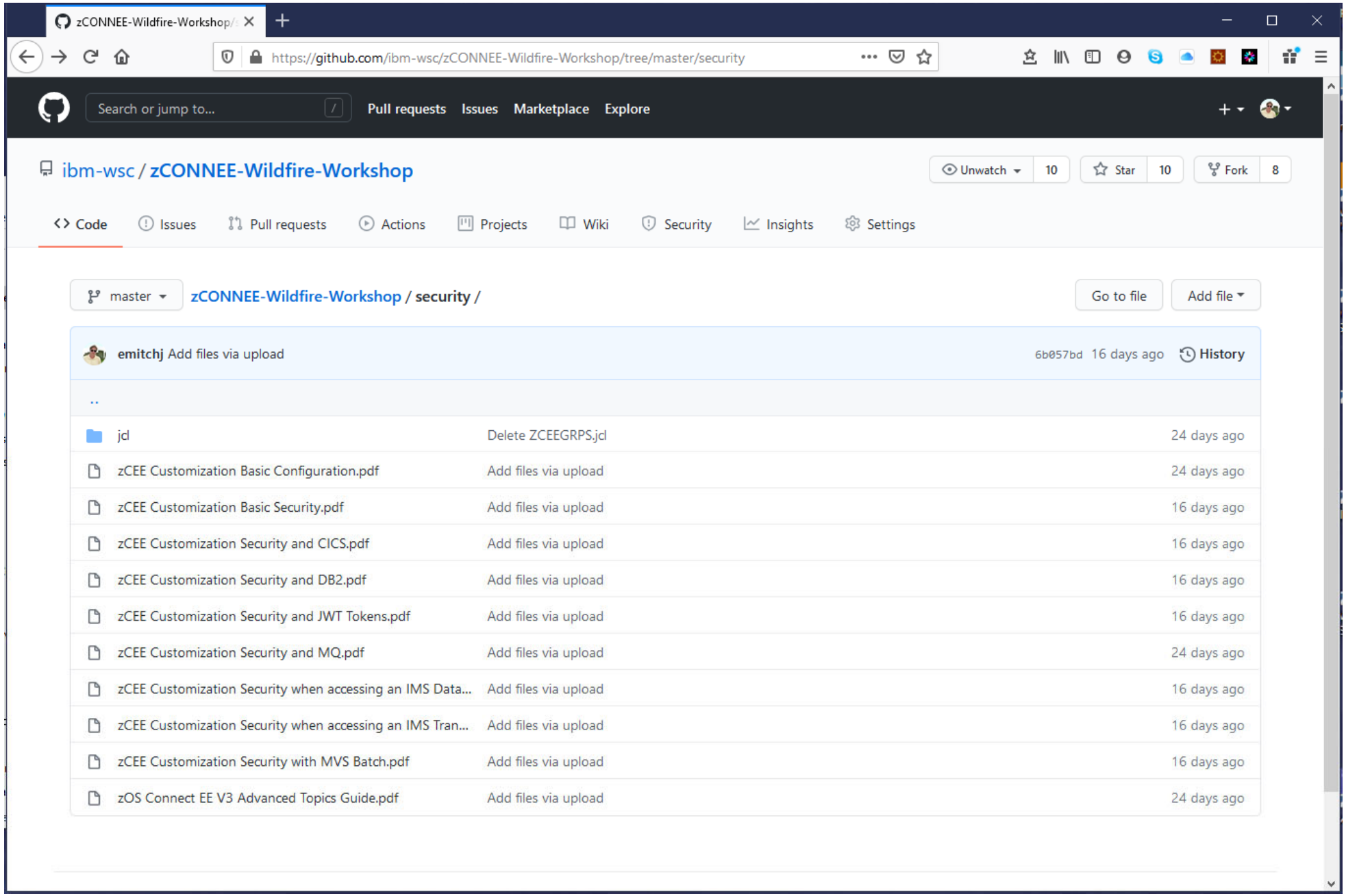
- Security design needs to consider
  - Authentication
  - Encryption
  - Authorization

- 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?

mitchj@us.ibm.com

Step by step exercises available on GitHub



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

mitchj@us.ibm.com