# WebSphere Liberty Profile on z/OS

## Introduction to Liberty Security

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

# Notes and Disclaimers

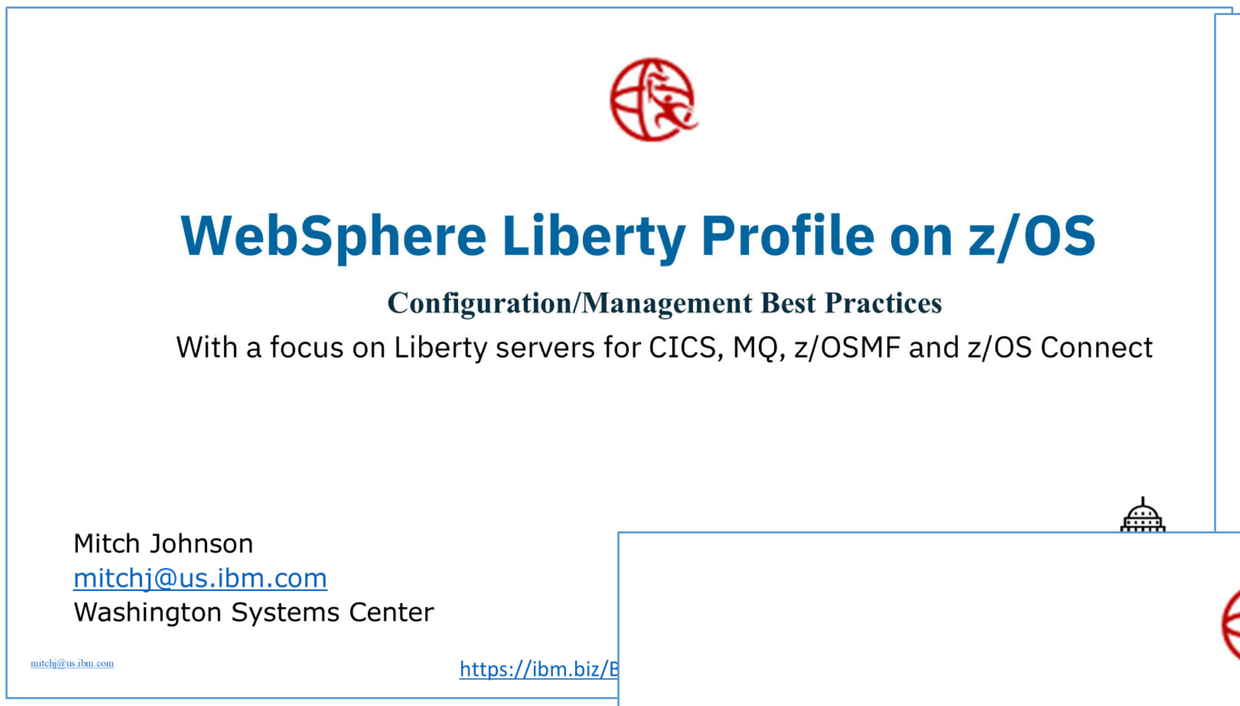
- Additional information included in this presentation was distilled from experience implementing security using RACF with z/OS products like CICS, IMS, Db2, MQ, etc. as well as Java runtimes environments like WebSphere Application Server and WebSphere Application Server Liberty (commonly called Liberty).

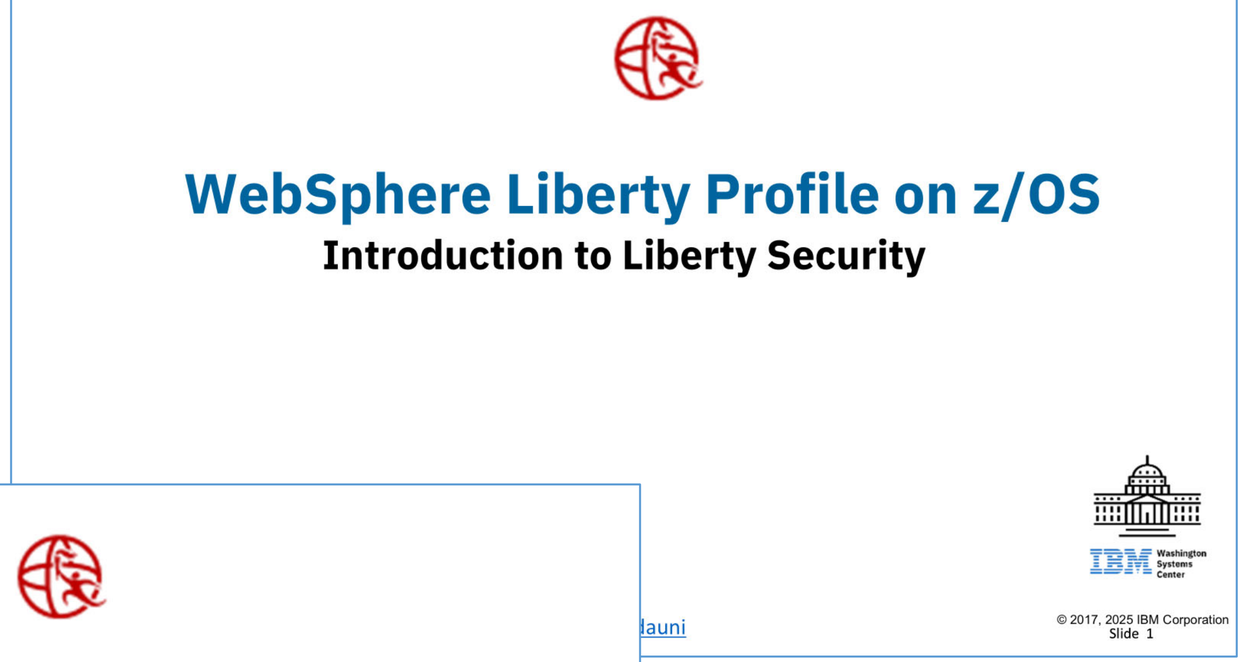
- There will be  additional information on slides that will  be designated as Tech/Tips. These contain information that at perhaps at least interesting and hopefully, useful to the reader.

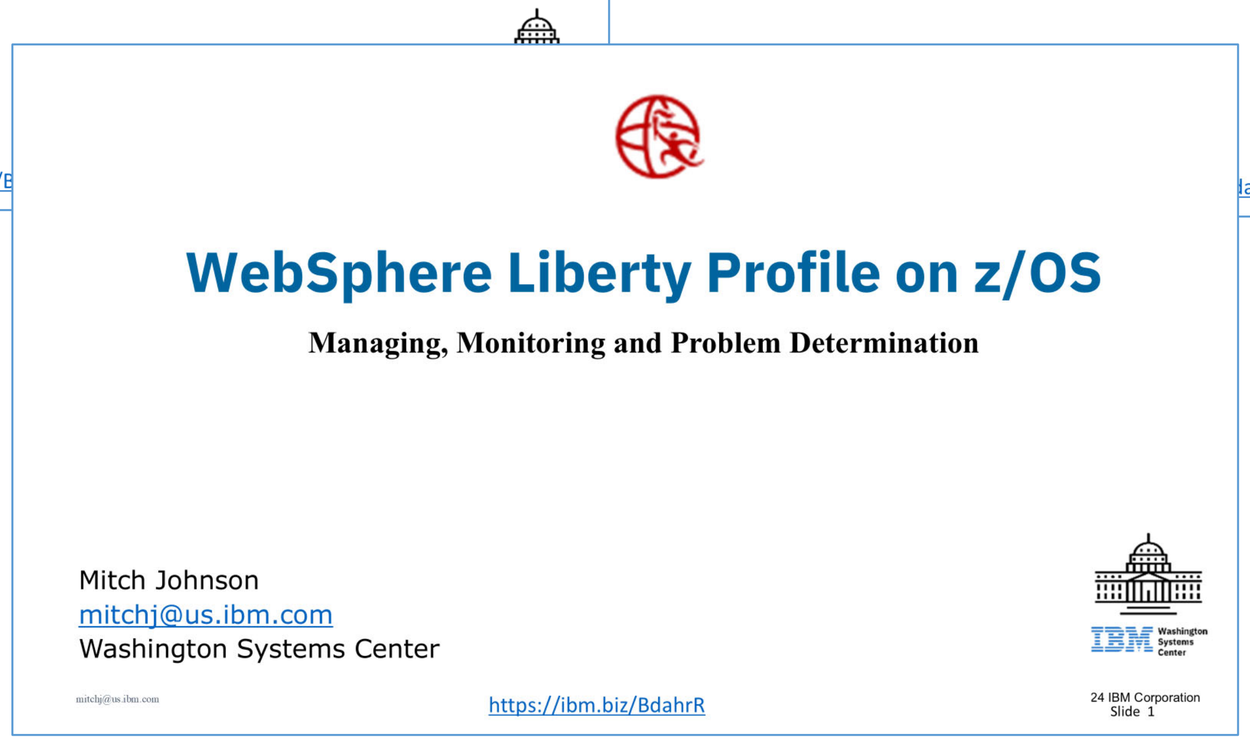
- A product icon will appear on a slide where the information is specific to a particular product. The icon will be for z/OS, or for Java, or for Liberty, or for CICS, or for MQ, or for IMS, or for Db2 or for z/OS Connect. Don't hesitate to ask questions as to why an icon does or does not appear on certain slides.

- The examples, tips, etc. present in this material are based on firsthand experiences.

# This session is part of a series of Liberty workshops. . .






https://www.ibm.com/support/pages/mainframe-system-education-wildfire-workshops

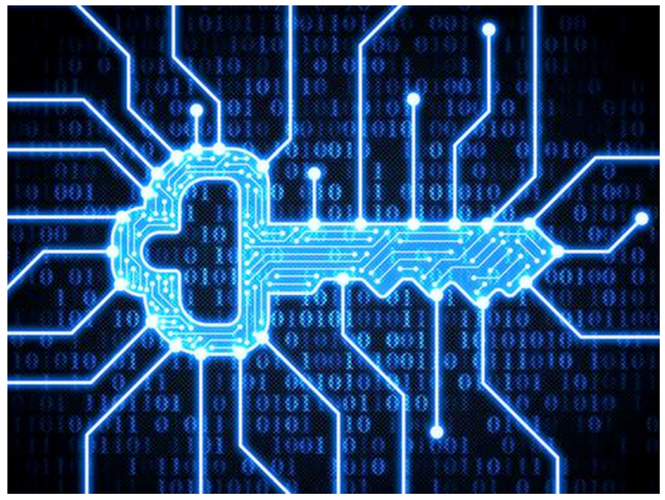
# General security terms or considerations

Security involves

- Identifying who or what is requesting access (**Authentication)**
  - Basic Authentication
  - Mutual Authentication using Transport Layer Security (TLS), formerly known as SSL
  - 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 using a digital signature)

- Controlling access (**Authorization)**
  - Is the authenticated identity authorized to access to Liberty
  - Is the authenticated identity authorized to access a specific web applications, API, Services, etc.



mitchj@us.ibm.com

https://ibm.biz/Bdauni

# We need to understand the challenges

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

  ➢This is a driver for implementing open security models like OAuth and OpenID Connect and standard tokens like JSON Web Token(JWT).

- Integrating security involves different products including 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 monolithic solution 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 always with the ultimate goal in mind. Providing security by understanding these waypoints and the corresponding options is the focus of this presentation.***

mitchj@us.ibm.com

# Client Security Options

mitchj@us.ibm.com

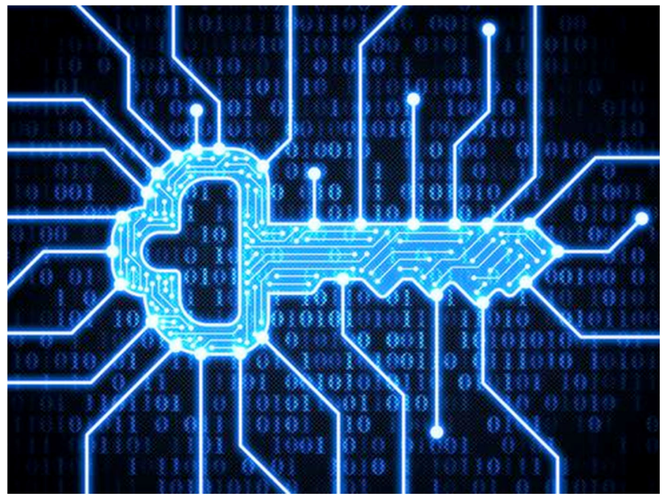
# General security terms or considerations

Security involves

- Identifying who or what is requesting access (**Authentication)**
  - Basic Authentication
  - Mutual Authentication using Transport Layer Security (TLS), formerly known as SSL
  - 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 using a digital signature)

- Controlling access (**Authorization)**
  - Is the authenticated identity authorized to access to Liberty
  - Is the authenticated identity authorized to access a specific web applications, API, Services, etc.

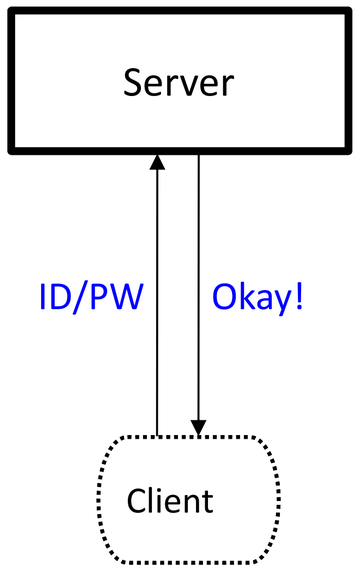


mitchj@us.ibm.com

# Liberty Client Authentication Options

Several different ways this can be accomplished:

## Basic Authentication

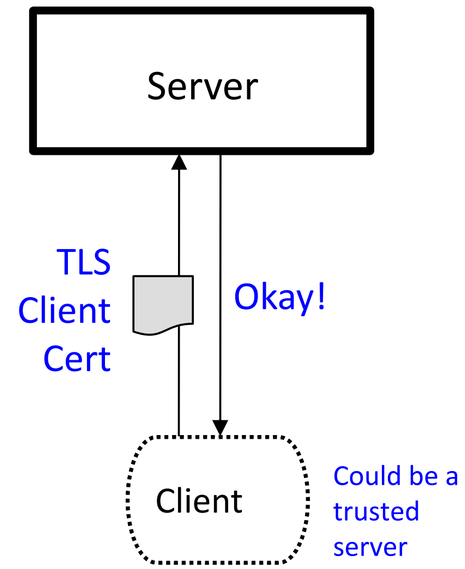



**Client supplies ID/PW or ID/PassTicket**

**Server checks registry:**

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

## Client Digital Certificate



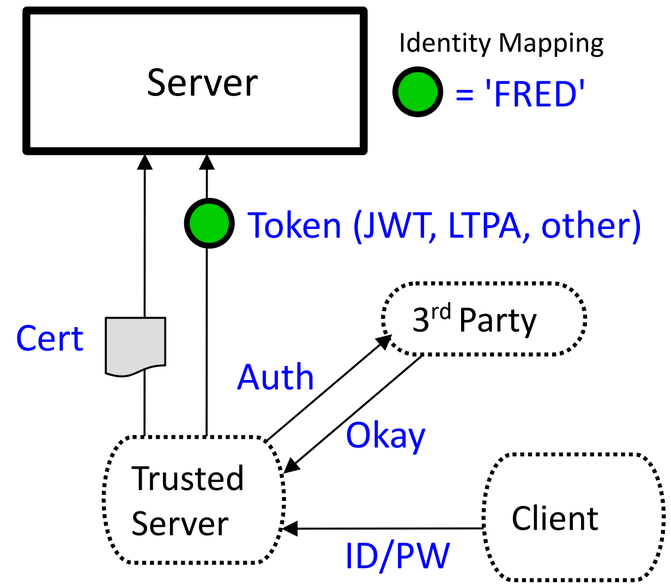

**Client supplies client personal certificate**

**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 Authentication Configuration (OpenAPI 2)

- requireAuth - requires the client to provide credentials

```
<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>
```

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.

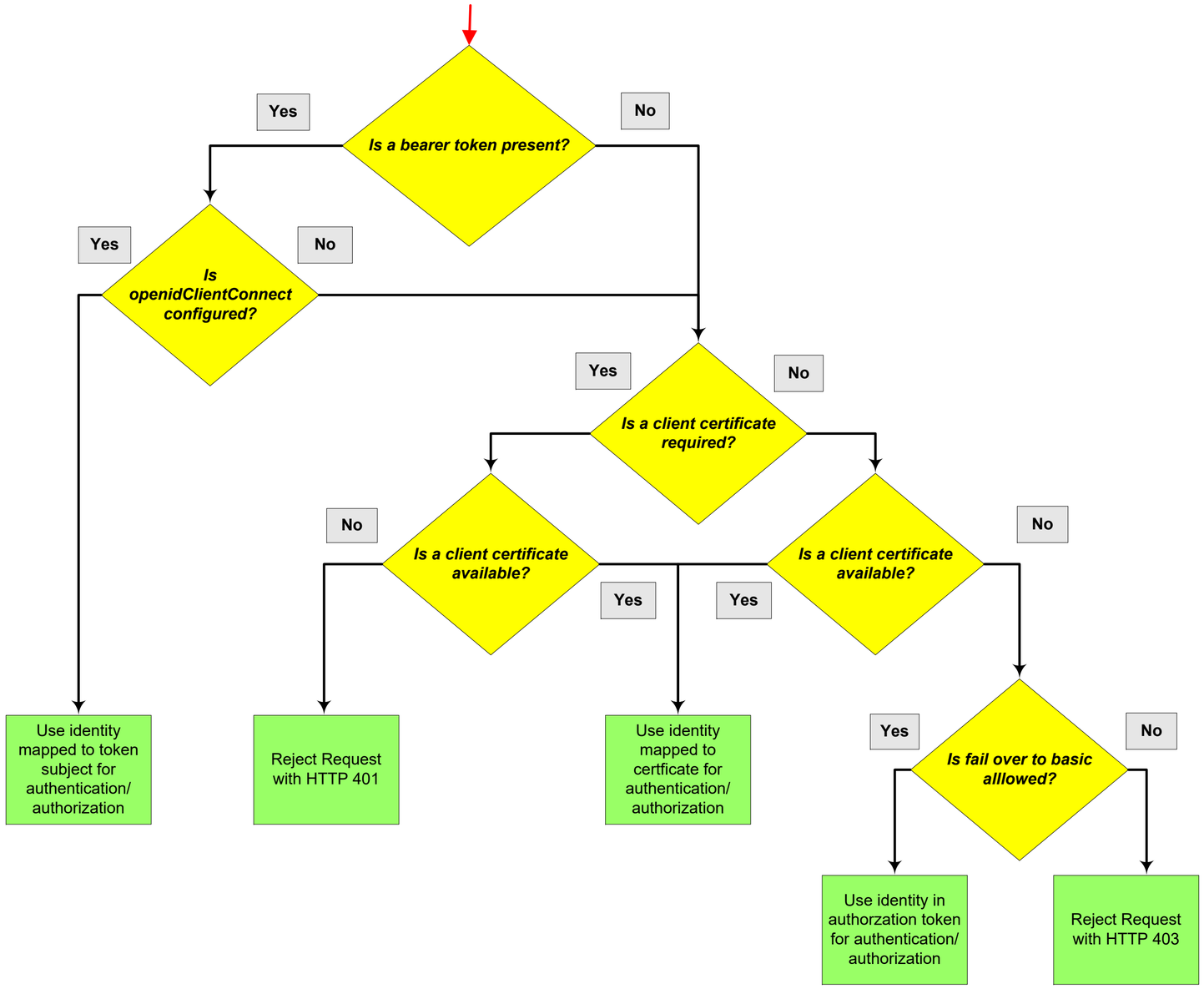
Requires that users specify security credentials to be authenticated in order to access all API requesters. If the requireAuth attribute is not set, the global setting on the zosconnect_zosConnectManager element is used instead, unless the requireAuth attribute is overridden on the specific API requester.

**The requireAuth attribute controls whether an inbound request must provide credentials using one of the three authentication methods, e.g., basic, client certificate, or third-party token.**

**Note that there are no equivalent configuration elements for an OpenAPI 3 server.**

mitchj@us.ibm.com

# Precedence order for determining an authorization identity

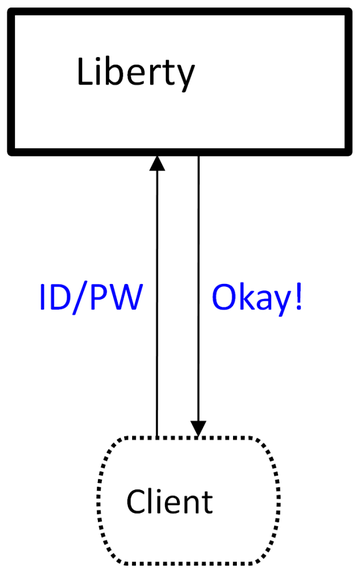



mitchj@us.ibm.com

# Authentication - Basic Authentication

Several different ways this can be accomplished:

## Basic Authentication

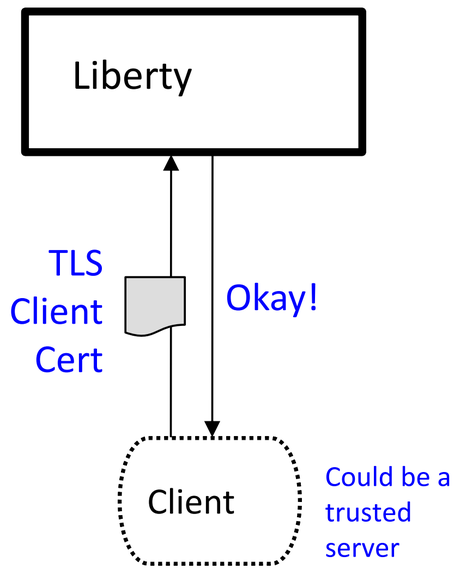



**Client supplies an identity/password or an identity/PassTicket**

**Server checks registry:**

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

## Client Certificate



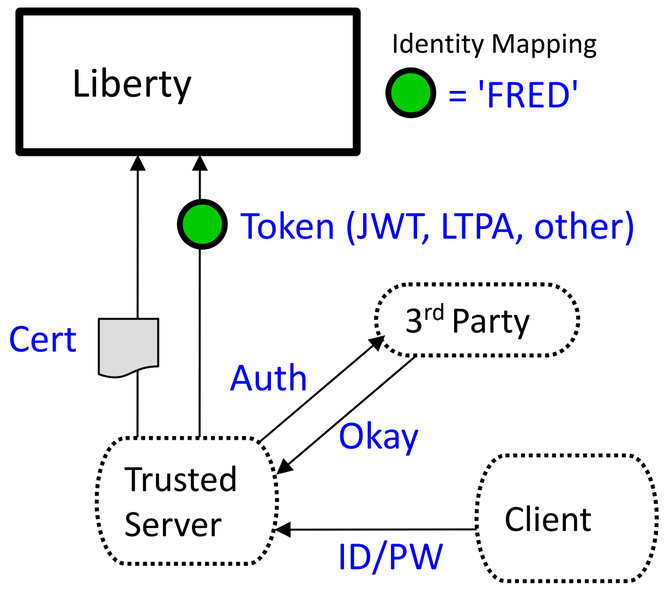

Client supplies client certificate

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.

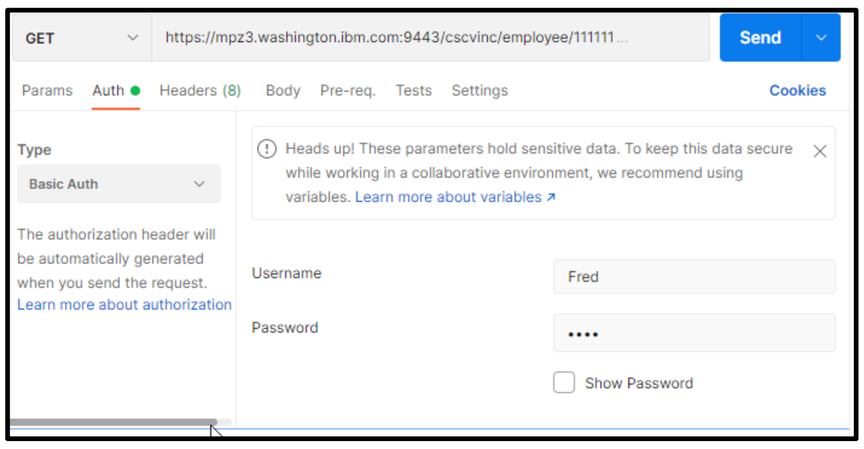
mitchj@us.ibm.com

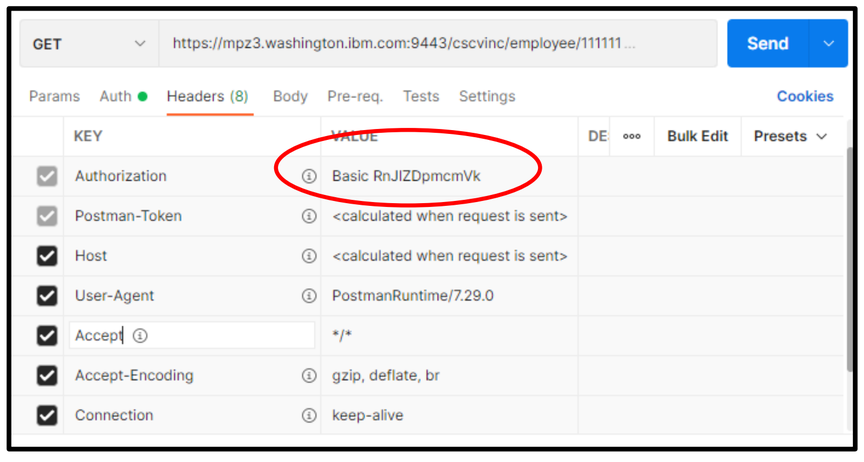
# Basic authentication – When the client provides an identity and password

- 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" />
```

- When sending a request to a Liberty server, 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:



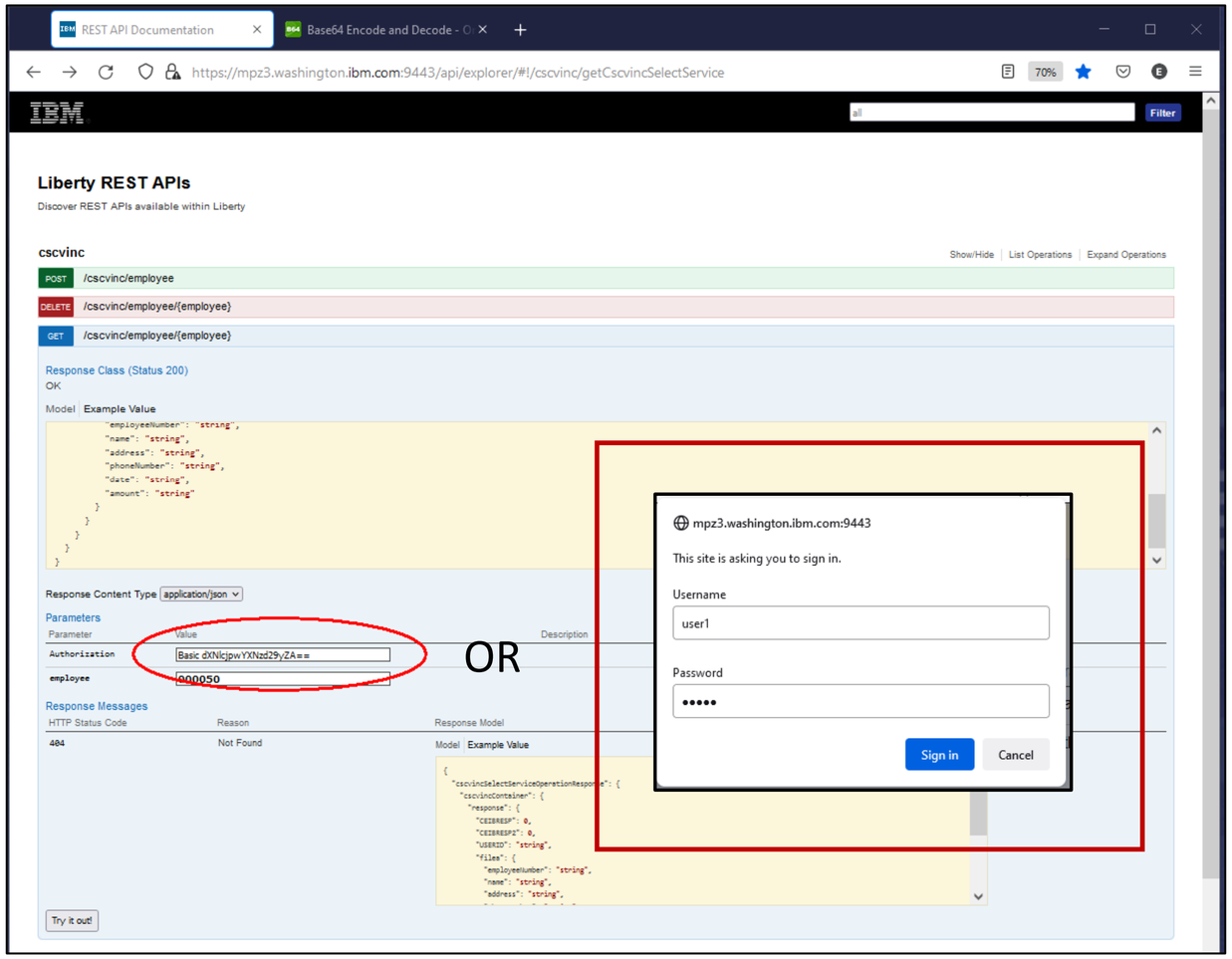



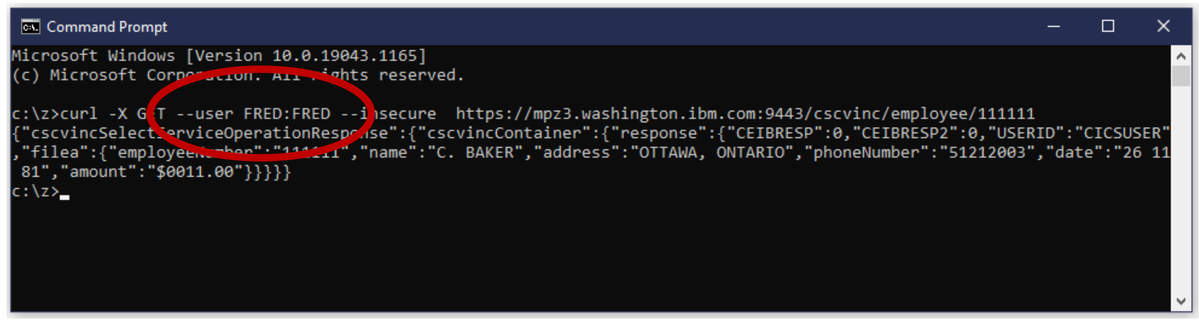

https://www.base64encode.org/

mitchj@us.ibm.com

# There are multiple ways to provide an identity and password

- 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.
  - Examples using the API Explorer feature , cURL, and a Java client.





mitchj@us.ibm.com

https://www.base64encode.org/

# Tech-Tip: A RACF PassTicket is 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

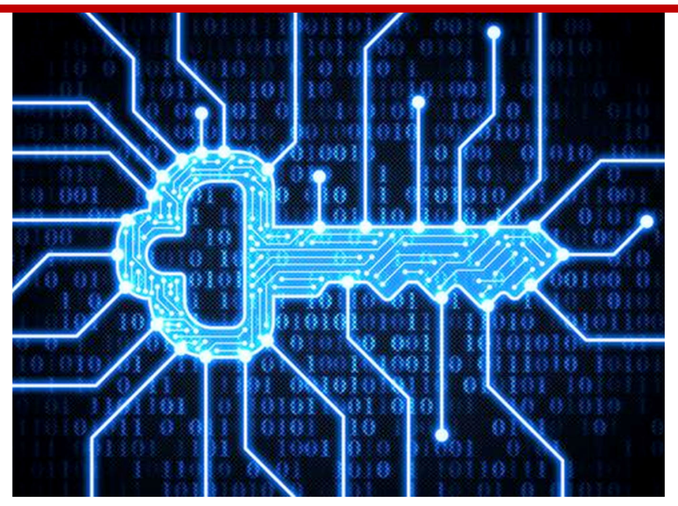
# General security terms or considerations

Security involves

- Identifying who or what is requesting access (**Authentication)**
  - ~~Basic Authentication~~
  - Mutual Authentication using Transport Layer Security (TLS), formerly known as SSL
  - ~~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 using a digital signature)

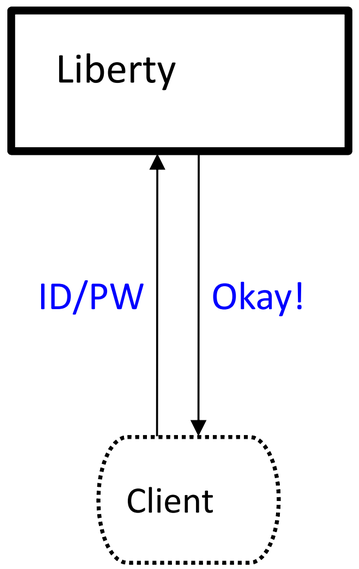
- Controlling access (**Authorization)**
  - Is the authenticated identity authorized to access to Liberty
  - Is the authenticated identity authorized to access a specific web applications, APIs, Services, etc.



# Authentication - TLS Mutual Authentication

Several different ways this can be accomplished:

## Basic Authentication

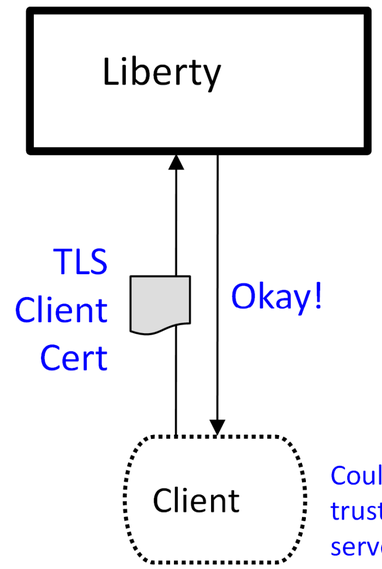



Server prompts for ID/PW

Client supplies ID/PW or ID/PassTicket

Server checks registry:
- Basic (server.xml)
- SAF

## Client Certificate

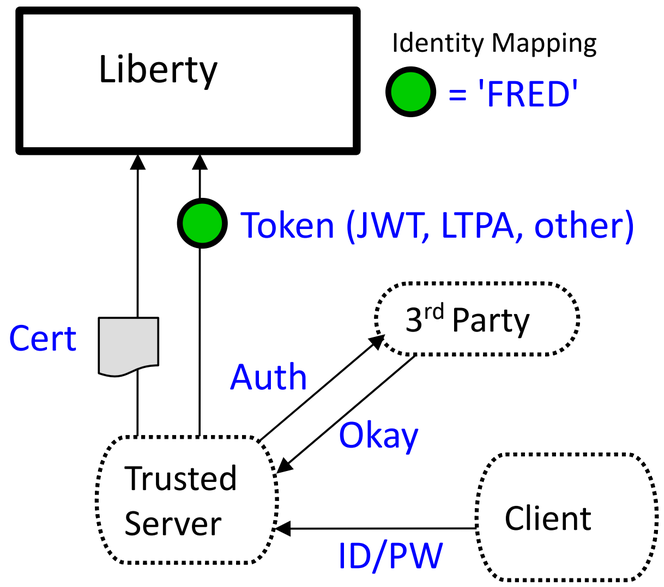



**Server prompts for client certificate information**

**Client supplies personal certificate information**

**Server validates client certificate information and maps it to an identity**

## Third Party Authentication




Client authenticates to 3rd party sever

Client receives a trusted 3rd party token

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

Registry options:
- We may not need to know these details.

mitchj@us.ibm.com

# Liberty JSSE (HTTPS) 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 send to for clients

Key ring for client connections to server endpoints

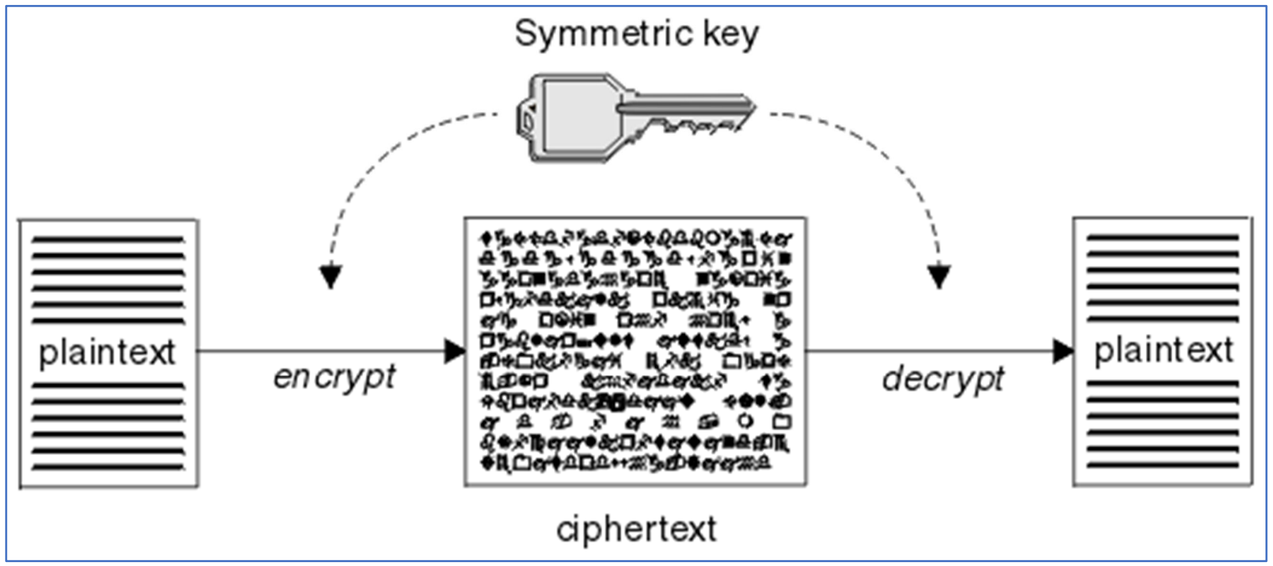
```
OpenAPI 2
<zosconnect_zosConnectManager
            requireAuth="true"
            requireSecure="true|false"/>

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

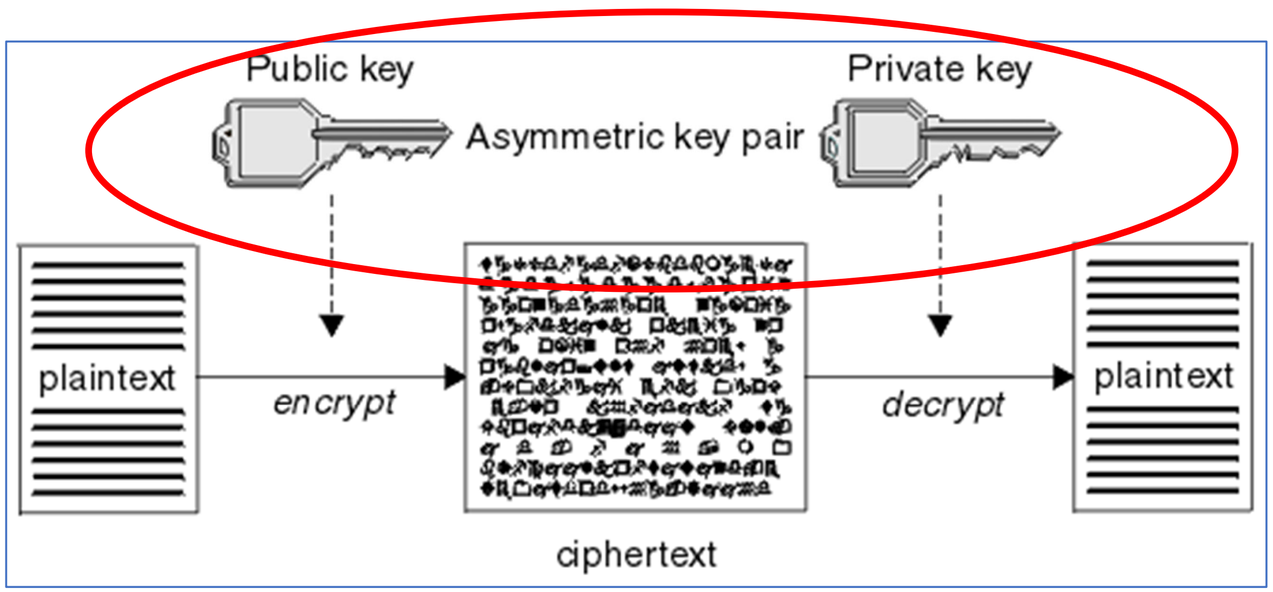
safkeyring:///**KeyRing** v safkeyring:**//owner/KeyRing**

**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.**

mitchj@us.ibm.com

# Tech-Tip: Symmetric key encryption v. Asymmetric key encryption




A symmetric key is a key shared by the endpoints. Both endpoints use the same key to encrypt and decrypt messages.

An asymmetric key pair is the preferred solution. There is no risk of compromise by sending a symmetric or shared key outside of a protected communication flow.

A message encrypted with a public key can only be decrypted by endpoint that has the private key. The privacy of the messages is ensured.

If an endpoint can successfully decrypt a message message encrypted received with a private key, the endpoint sending the message has successfully asserted its validity by proving it has the private used to encrypt the message.

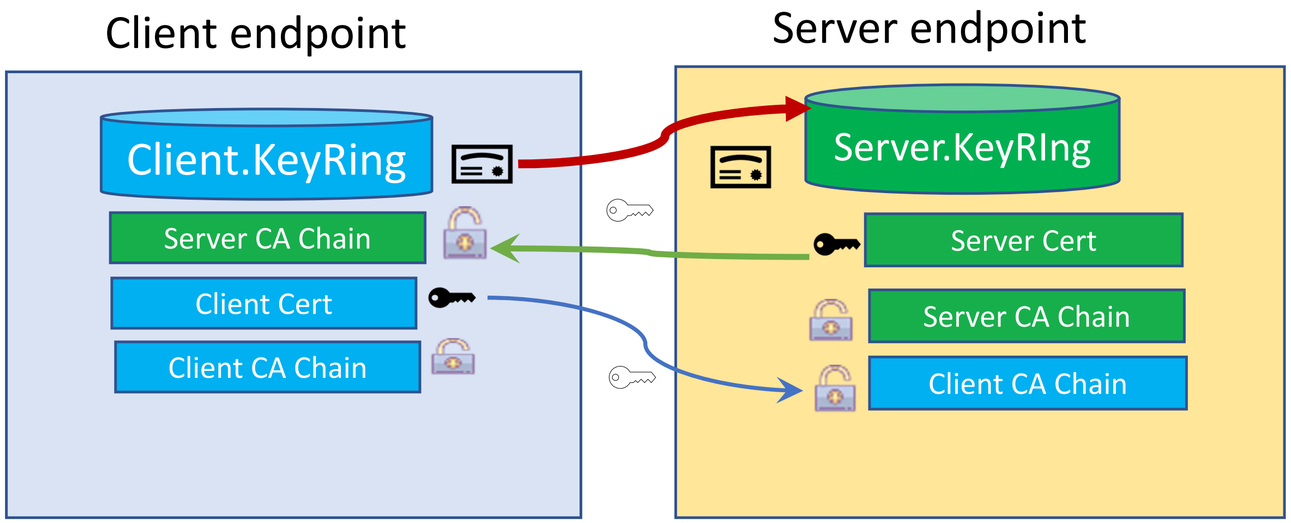
mitchj@us.ibm.com

# The basic TLS Handshake Flow (HTTPS)

The HTTPS protocol involves a TLS handshake –

Server Endpoint Authentication (always occurs when HTTPS is the protocol)

Mutual Authentication (optional, at the request of the server endpoint)




A certificate with a private key* is required when an endpoint needs to prove its identity

Certificate's public key

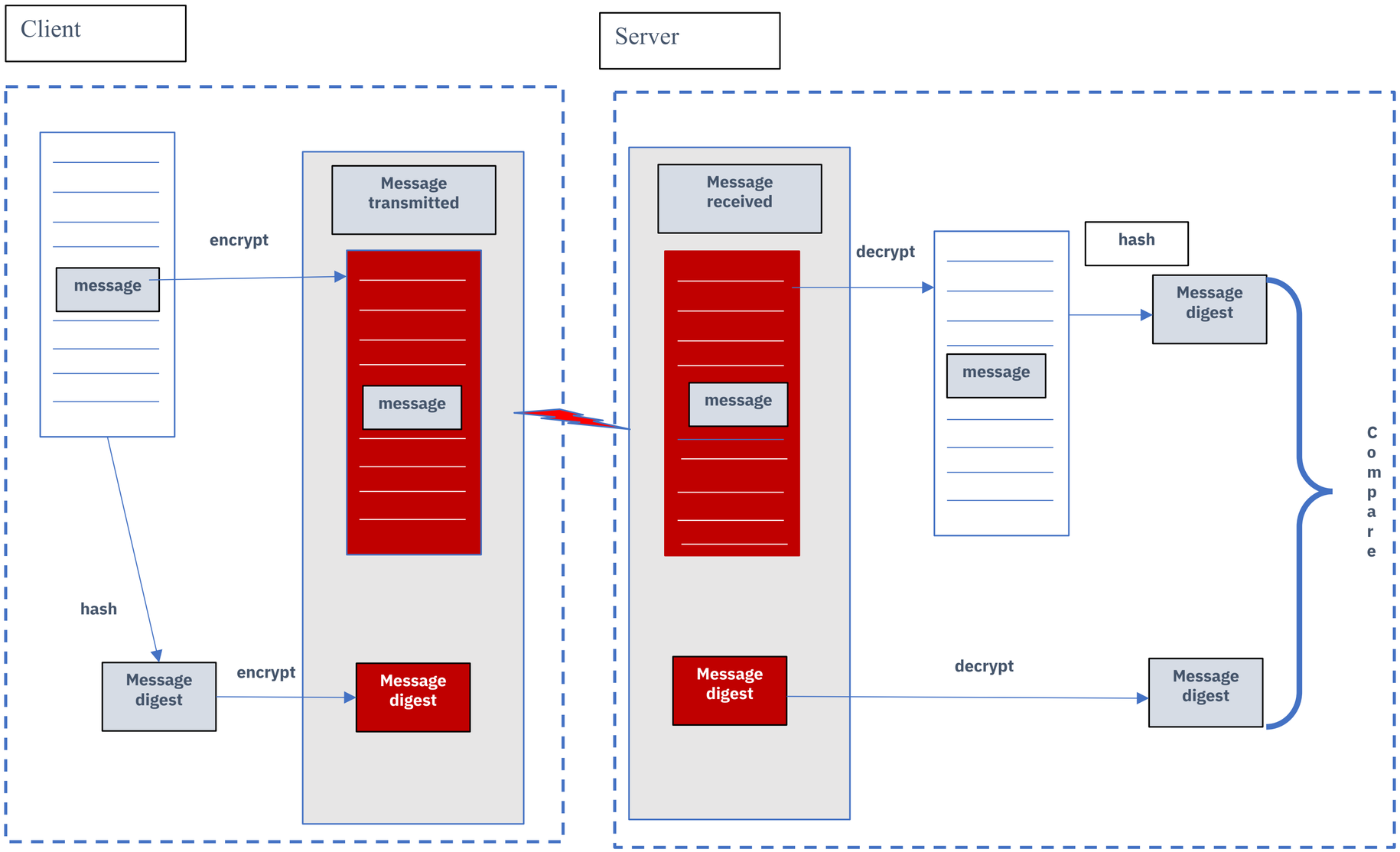
Certificate Authority (CA) certificate chain#

*For server and/or mutual authentication to work, the endpoint sending the 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 up to and including the root CA.

mitchj@us.ibm.com

# Tech-Tip: Message Integrity and Encryption (client to server endpoint)




The handshake process generates a symmetric key and at regular intervals, the two endpoints will use the original asymmetric keys to generate a new symmetric key.

mitchj@us.ibm.com

# Tech-Tip: RACF digital certificate (RACDCERT) command review

```
RACDCERT ID(LIBSERV) GENCERT SUBJECTSDN(CN('wg31.washington.ibm.com') +
 O('IBM') OU('LIBERTY')) WITHLABEL('Liberty Server Cert') ALTNAME(DOMAIN('wg31z.washington.ibm.com')
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

[1] mitchj@us.ibm.com

# Tech-Tip: Accessing a certificate's private key in non-virtual key rings

Two types of RACF profiles resources are used to control access to key ring and certificates

- **RDATALIB** for controlling access to a specific key ring
- **FACILITY** for controlling access to key rings globally

**User certificates (connected to the key ring with usage PERSONAL)**

- Global profiles uses the **FACILITY** resource **IRR.DIGTCERT.LISTRING**:
  - **READ** access is required to access one's own key ring and private key
  - **UPDATE** access is required to access another user's key private key

- Specific key rings uses the **RDATALIB** class **<ring owner>.<ring name>.LST**
  - **READ** access is required to access one's own private key
  - **UPDATE** access is required to access another identity's private keys

**CERTAUTH and SITE certificates (connected to the key ring with usage PERSONAL)**

- Global profiles uses the **FACILITY** resource **IRR.DIGTCERT.GENCERT**:
  - **CONTROL** access is required to access a CERTAUTH or SITE certificate private key ring

- Specific key rings uses the **RDATALIB** class **<ring owner>.<ring name>.LST**
  - **CONTROL** access is required to access the private keys of CERTAUTH and SITE certificates

**Remember:** When switching from global FACILITY class profiles to specific ring RDATALIB class profiles, the RDATLIB resources will be checked first

mitchj@us.ibm.com

# Tech/Tip:  Details of the flow with mutual authentication (TLS 1.2)

1. A Client sends a request to server for a protected session in a ***ClientHello*** message. Included in the request is the TLS capabilities of the client (e.g., TLS 1.2 or 1.3) and a list of supported ciphers in preference order.
2. The server selects the TLS version and selects cipher from the list sent by the client and returns this information in a ***ServerHello*** message.
3. The server's certificate public information (including the **public key**) is sent to the client in a ***Certificate*** message.
4. The server sends cryptographic information for the client to use for encrypting a pre-master key in a **Server key exchange** message.
5. **For mutual authentication, the server sends a *CertificateRequest* message requesting a client's personal certificate.**
6. The server concludes by sending a ***ServerHelloDone*** message.
7. The client verifies the server's certificate with its trust store.
8. **If mutual authentication is requested, the client sends its public personal certificate information in a *Certificate* message**
9. The client then uses the ***server's public* key** to generate and encrypt a 48 byte "premaster secret" message which is sent to the server in a ***ClientKeyExchange*** message.
10. **When mutual authentication is requested, a digitally signature (hashed) of the concatenation of all previous handshake messages is encrypted with the client's private key sent in a *CertificateVerify* message.**
11. The ***Change Cipher*** message is used to change the from cipher used during the handshake so all subsequent messages will be encrypted using a different cipher.
12. The server uses its **private key** to decrypt the "premaster secret" message (**only the private key can be used to decrypt the message**).
13. **If mutual authentication is requested, the server verifies the client's personal certificate with its key ring and uses the client's public key to decrypt and verify the message sent in the *CertificateVerify* message.**
14. Both the Client and Server use the "premaster secret" to compute a 'master secret', also know as "shared secret" or "session key" (symmetric encryption)
15. Client and server will use this "shared secret" or "session key" to encrypts messages sent between the endpoints.

mitchj@us.ibm.com

# Tech/Tip:  Details of the flow with mutual authentication (TLS 1.2)

1. A Client sends a request to server for a protected session in a ***ClientHello*** message. Included in the request is the TLS capabilities of the client (e.g., TLS 1.2 or 1.3) and a list of supported ciphers in preference order.
2. The server selects the TLS version and selects cipher from the list sent by the client and returns this information in a ***ServerHello*** message.
3. The server's certificate public information (including the **public key**) is sent to the client in a ***Certificate*** message.
4. The server sends cryptographic information for the client to use for encrypting a pre-master key in a **Server key exchange** message.
5. **For mutual authentication, the server sends a *CertificateRequest* message requesting a client's personal certificate.**
6. The server concludes by sending a ***ServerHelloDone*** message.
7. The client verifies the server's certificate with its trust store.
8. **If mutual authentication is requested, the client sends its public personal certificate information in a *Certificate* message**
9. The client then uses the ***server's public* key** to generate and encrypt a 48 byte "premaster secret" message which is sent to the server in a ***ClientKeyExchange*** message.
10. **When mutual authentication is requested, a digitally signature (hashed) of the concatenation of all previous handshake messages is encrypted with the client's private key sent in a *CertificateVerify* message.**
11. The ***Change Cipher*** message is used to change the from cipher used during the handshake so all subsequent messages will be encrypted using a different cipher.
12. The server uses its **private key** to decrypt the "premaster secret" message (**only the private key can be used to decrypt the message**).
13. **If mutual authentication is requested, the server verifies the client's personal certificate with its key ring and uses the client's public key to decrypt and verify the message sent in the *CertificateVerify* message.**
14. Both the Client and Server use the "premaster secret" to compute a 'master secret', also know as "shared secret" or "session key" (symmetric encryption)
15. Client and server will use this "shared secret" or "session key" to encrypts messages sent between the endpoints.

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/docs/en/ibm-mq/9.3?topic=java-tls-cipherspecs-ciphersuites-in-mq-classes

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/docs/en/ibm-mq/9.3?topic=java-tls-cipherspecs-ciphersuites-in-mq-classes

mitchj@us.ibm.com

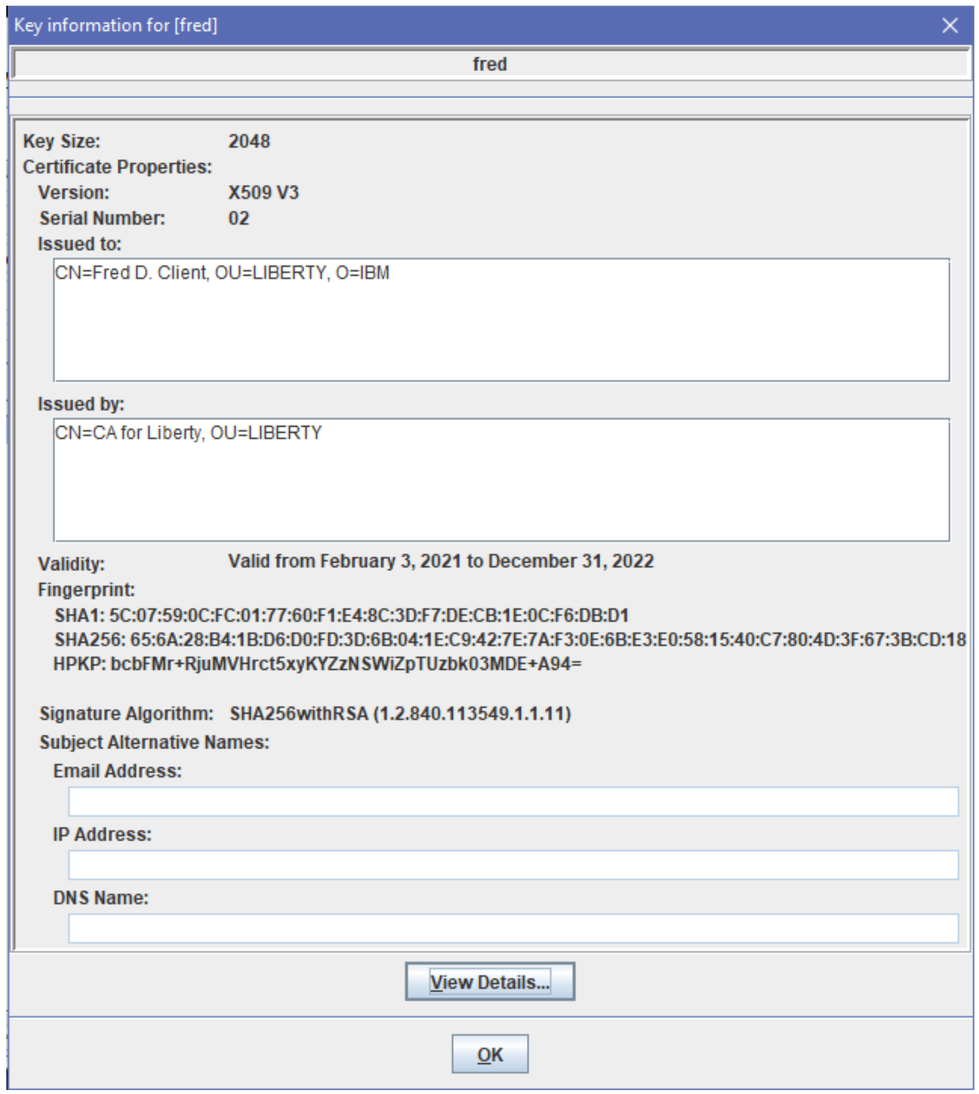
# Tech-Tip: Anatomy of a RACF Personal Digital Certificate

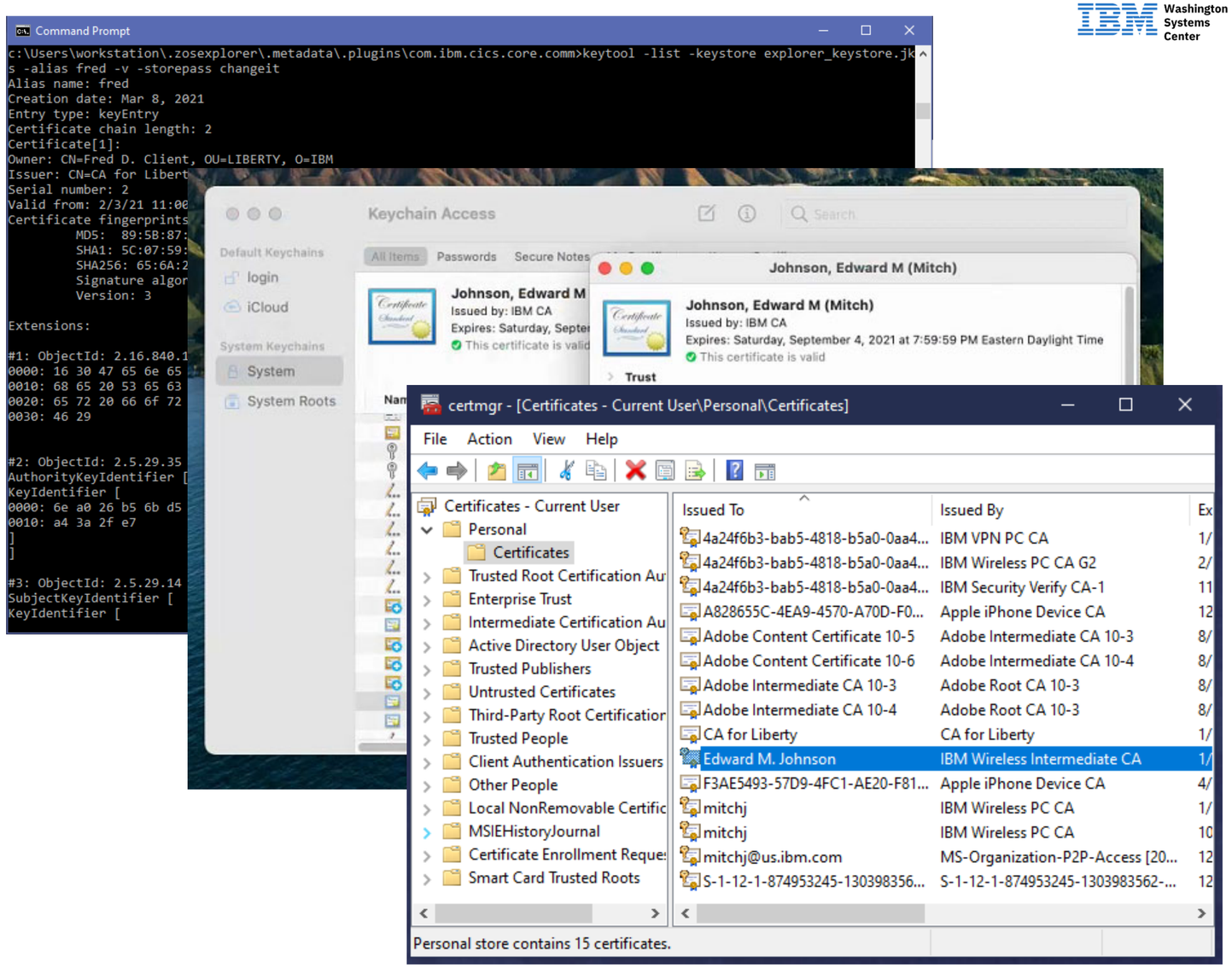
```
Digital certificate information for user ATSSERV:

  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: *.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<
```

Some clients are more sensitive than others when checking common names (CN). They will check the endpoint's actual host name versus the CN and if they do not match, the certificate is rejected. The *AltName* attribute can be used to resolves this issue.

[1] mitchj@us.ibm.com

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



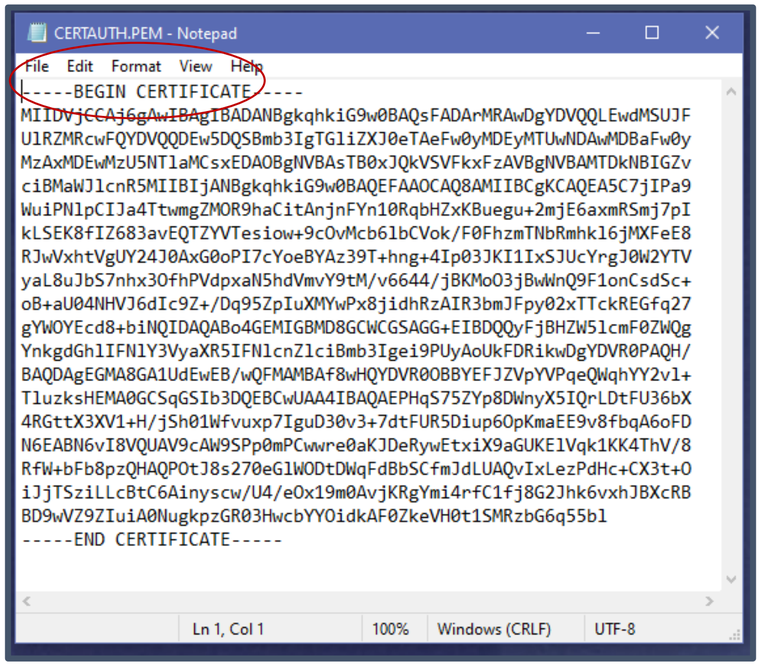



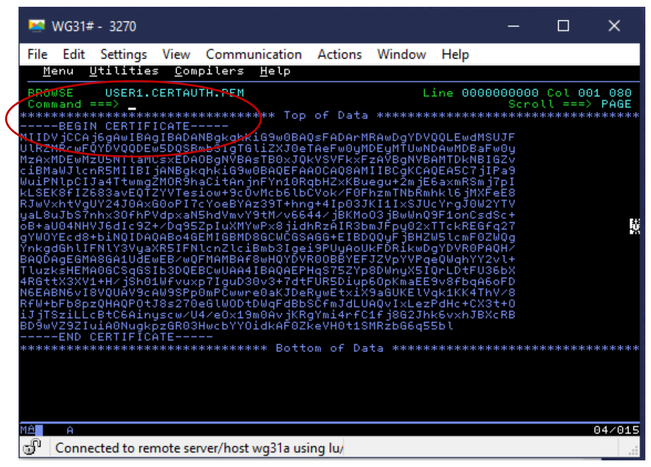

Both can be found in /usr/lpp/java/J8.0_64/bin or c:/Program Files/IBM/Java80/jre/bin

mitchj@us.ibm.com
mitchj@us.ibm.com

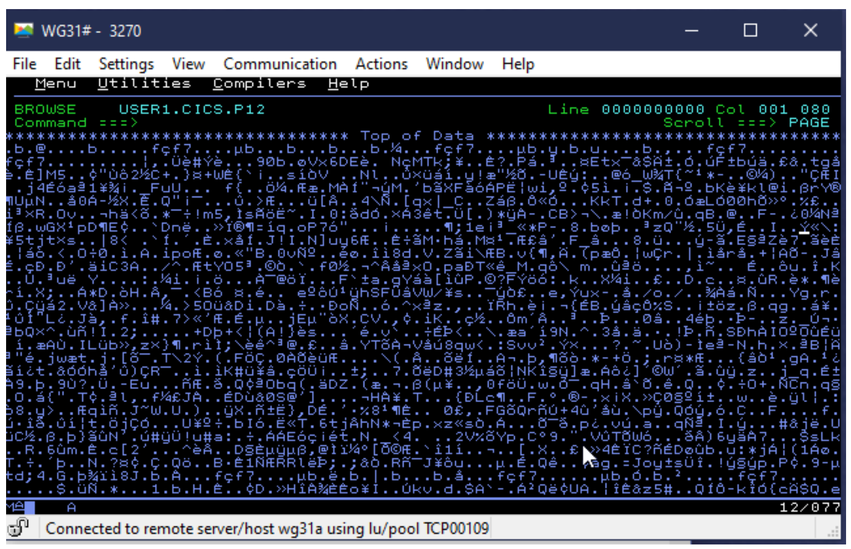
# Tech-Tip: How to tell what kind of certificate you have?

Public certificate (site or certificate authority or certificate request)




Personal certificate with private key


mitchj@us.ibm.com

mitchj@us.ibm.com

# 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

# Liberty JSSE (HTTPS) 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 send to for clients

Key ring for client connections to server endpoints

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

**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.**

mitchj@us.ibm.com

# Tech/Tip: Combining TLS mutual and basic authentication

```
//*************************************************************
//*  SET SYMBOLS
//*************************************************************
//EXPORT EXPORT SYMLIST=(*)
// SET CURL='/usr/lpp/IBM/foz/v1r1/bin'
//*************************************************************
//*  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     [1]
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"/>
```

[1]BM Open Enterprise Foundation for z/OS® is a no-cost collection of vetted open source developer tools (Git, Curl, GNU Bash, Vim, GPG, JQ and more) for z/OS that provides popular Linux® and UNIX software development tools for z/OS.

mitchj@us.ibm.com

# The Liberty JSSE server XML configuration for outbound connections

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

   <ssl id="cicsTLSSettings"
      keyStoreRef="CICSKeyStore"
      trustStoreRef="CICSKeyStore"
      clientKeyAlias="Liberty Client Cert"/>
   <keyStore id="CICSKeyStore"
      location="safkeyring:///Liberty.CICS.KeyRing"
      password="password" type="JCERACFKS"
      fileBased="false" readOnly="true" />
   <ssl id="db2TLSSettings"
      keyStoreRef="Db2KeyStore"
      trustStoreRef="Db2KeyStore"
      clientKeyAlias="Liberty Client Cert"/>
   <keyStore id="Db2KeyStore"
      location="safkeyring:///Liberty.Db2.KeyRing"
      password="password" type="JCERACFKS"
     fileBased="false" readOnly="true" />
<ssl id="otherTLSSettings"
    keyStoreRef="OtherKeyStore"
    trustStoreRef="OtherKeyStore">
    outboundConnection
      host="wg31.washington.ibm.com"
      port="9555"
      clientCertificate="Client Cert"/>
 </ssl>
 <keyStore id="OtherKeyStore"
    location="safkeyring:///Other.KeyRing"
    password="password" type="JCERACFKS"
    fileBased="false" readOnly="true" />
```

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

<zosconnect_authorizationServer sslCertsRef="SSL repertoire"/>
<zosconnect_cicsIpicConnection sslCertsRef="cicsTLSSettings"/>
<zosconnect_db2Connection sslCertsRef="db2TLSettings"> *
<zosconnect_endpointConnect sslCertsRef= "SSL repertoire"/>
<zosconnect_zosConnectRestClient sslCertsRef="SSL repertoire"/>
<zosconnect_zosConnectServiceRestClientConnection sslCertsRef="SSL repertoire"/>
```

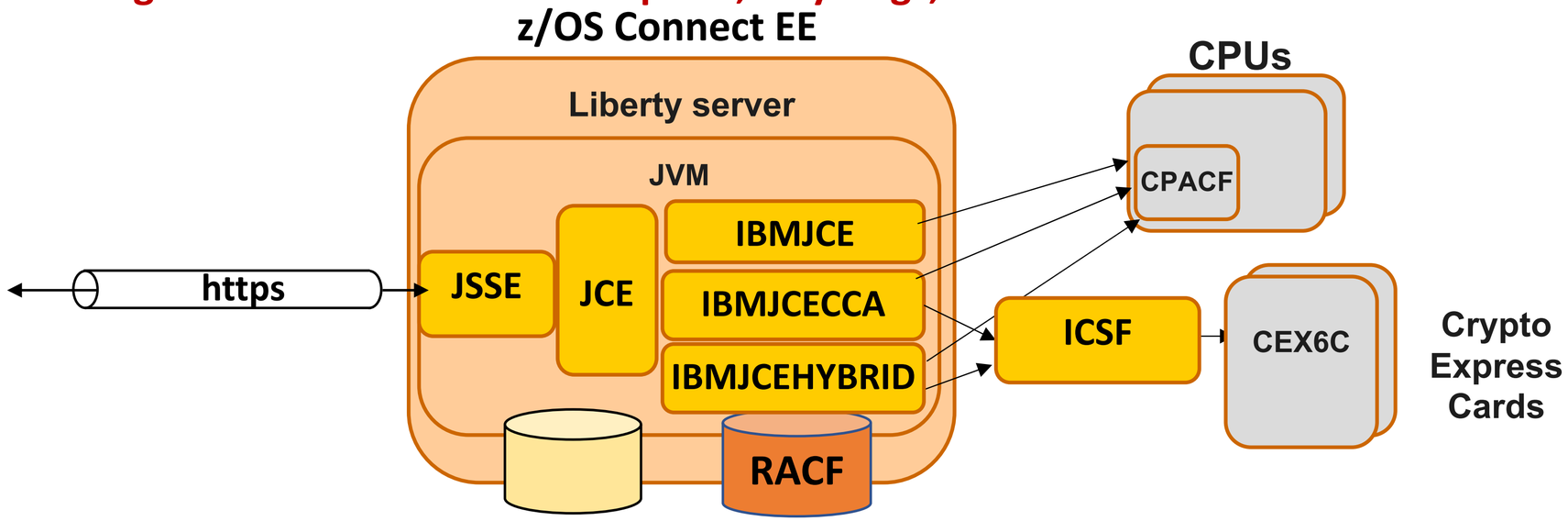
**F BAQSTRT,REFRESH,KEYSTORE**
**F BAQSTRT,REFRESH,KEYSTORE,ID=CICSKeyStore**
**F BAQSTRT,REFRESH,KEYSTORE,ID=Db2KeyStore**
**F BAQSTRT,REFRESH,KEYSTORE,ID=OtherKeyStore**

mitchj@us.ibm.com

* OpenAPI 3 only

# Using JSSE with Liberty

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




server.xml

- 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.

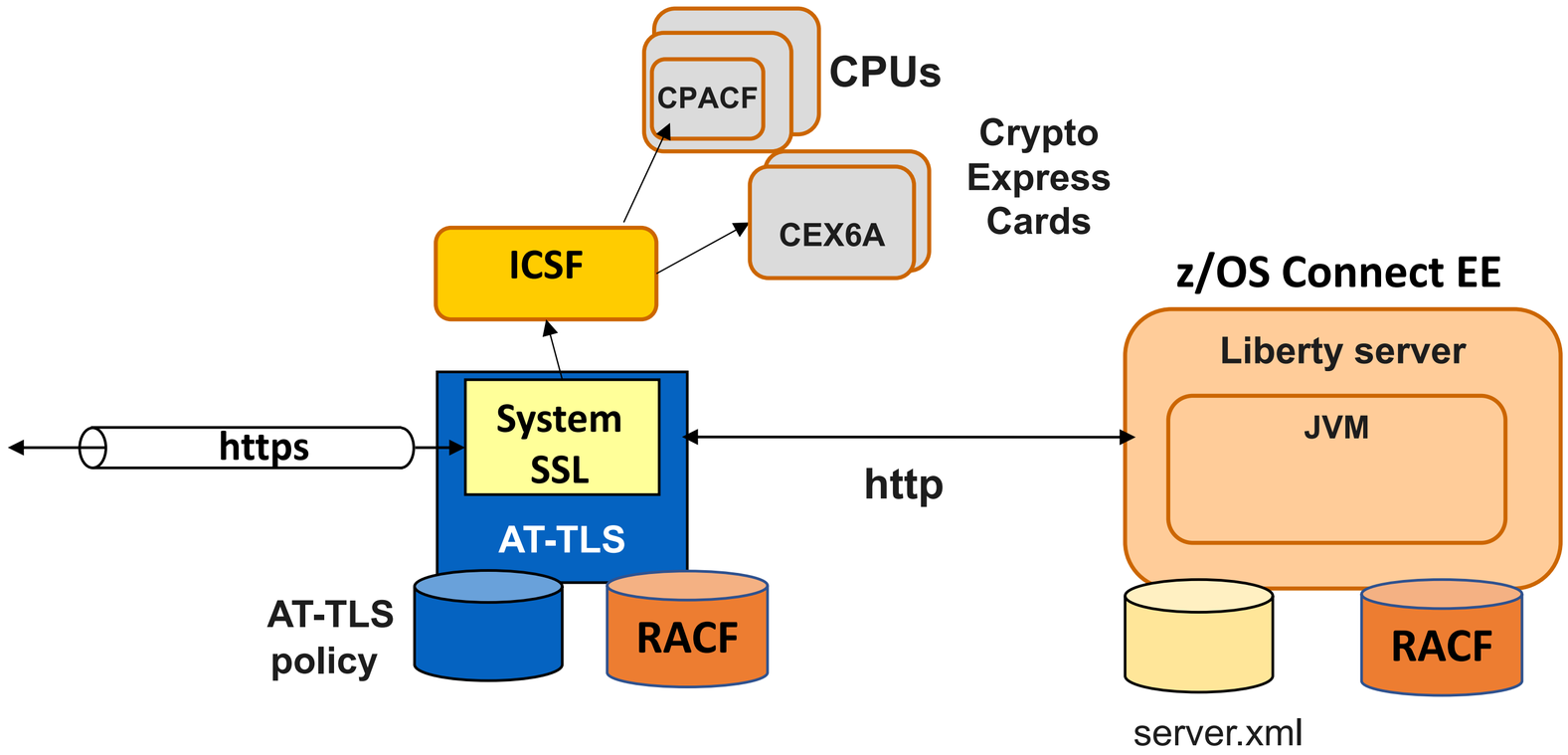
mitchj@us.ibm.com




Slide 34

# Using AT-TLS with Liberty

**The server XML configuration uses <u>no</u> HTTPS protocol, key rings or other JSSE attributes**



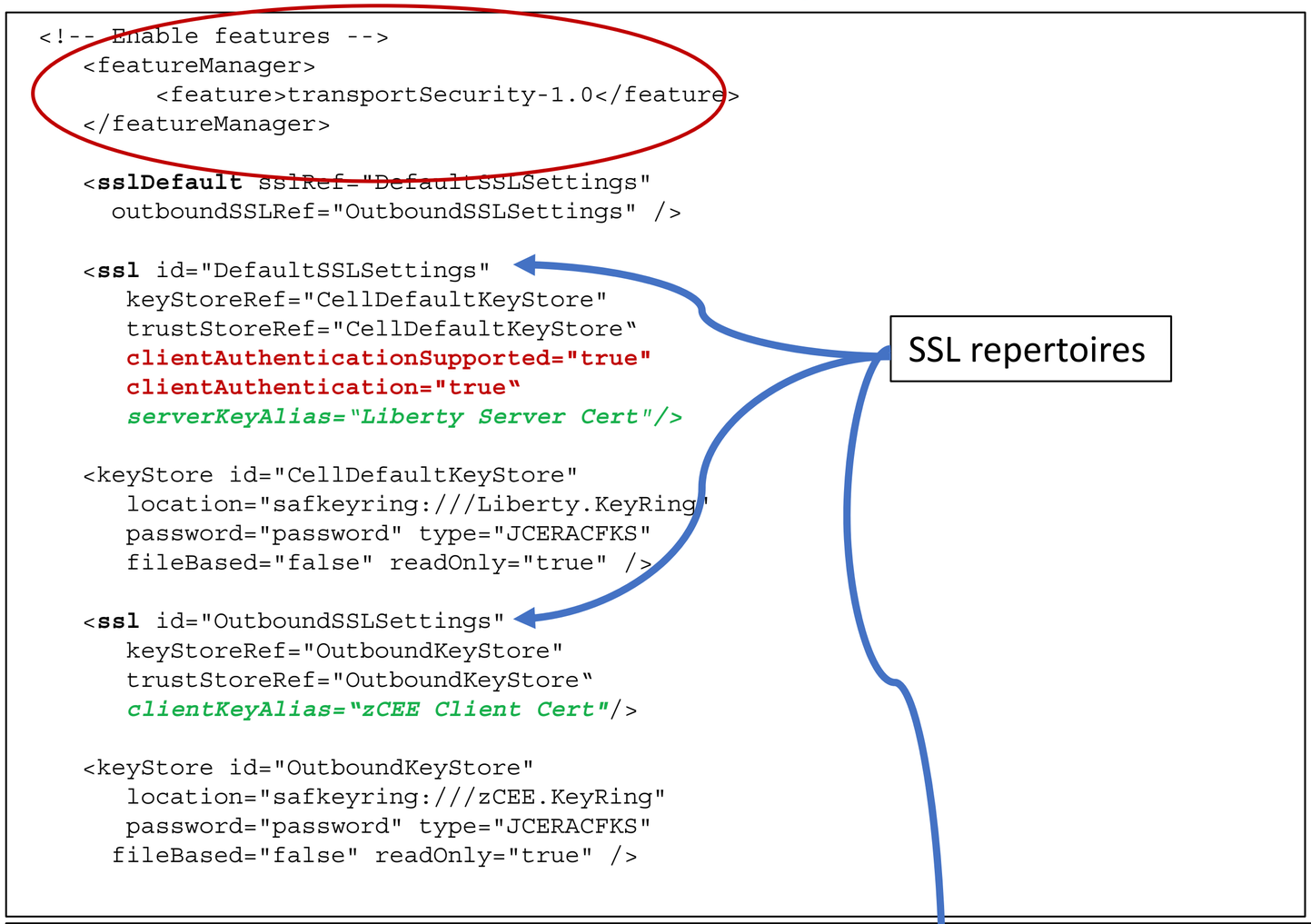

- **Application Transparent TLS** (AT-TLS) creates a secure session on behalf of z/OS Connect
- Only define http ports in server.xml (z/OS Connect does not know that TLS session exists)
- Define TLS protection for all applications (including z/OS Connect) in **AT-TLS policy**
- AT-TLS uses **System SSL**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

mitchj@us.ibm.com

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

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"
      clientKeyAlias="zCEE Client Cert"/>

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

```
<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




Slide 37

# Let’s explore TLS options using the contents of these key rings

Liberty’s outbound key ring

Liberty’s inbound key ring

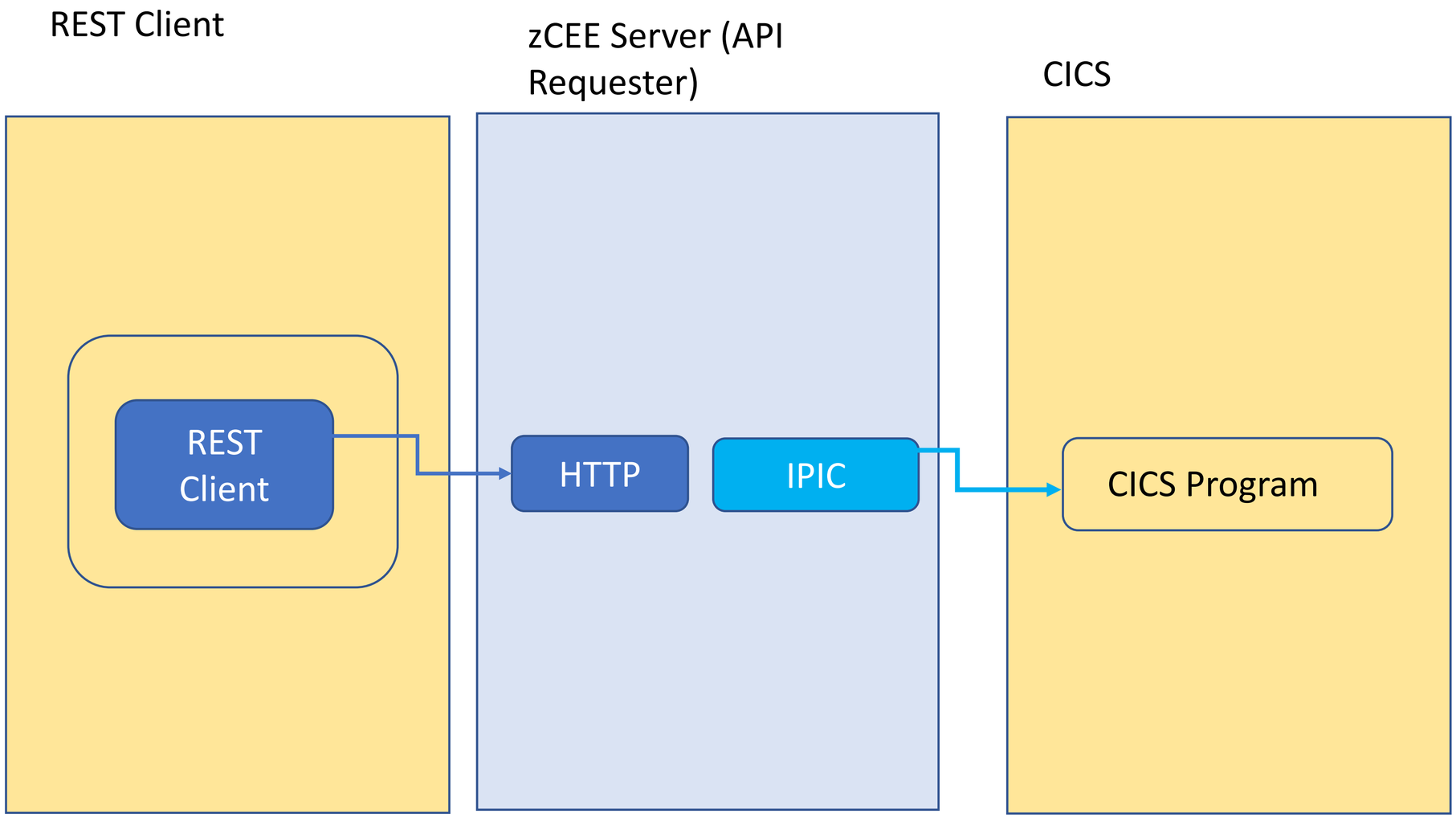
```
Digital ring information for user LIBSERV:
  Ring:
       >zCEE.KeyRing<
  Certificate Label Name              Cert Owner     USAGE      DEFAULT
  --------------------------------    ------------   --------   -------
  zCEE CA                             CERTAUTH       CERTAUTH     NO
  Liberty CA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

Tech-Tip: when Liberty is the client endpoint, and more than one personal certificate is connected to a key ring. Use the SSL repertoire *clientKeyAlias* attributes to select the personal certificate to be used in a handshake.

Tech-Tip: when Liberty is the server endpoint, and more than one personal certificate is connected to a key ring. Use the SSL repertoire *serverKeyAlias* attributes to select the personal certificate to be used in a handshake.

mitchj@us.ibm.com

# No TLS between any endpoints

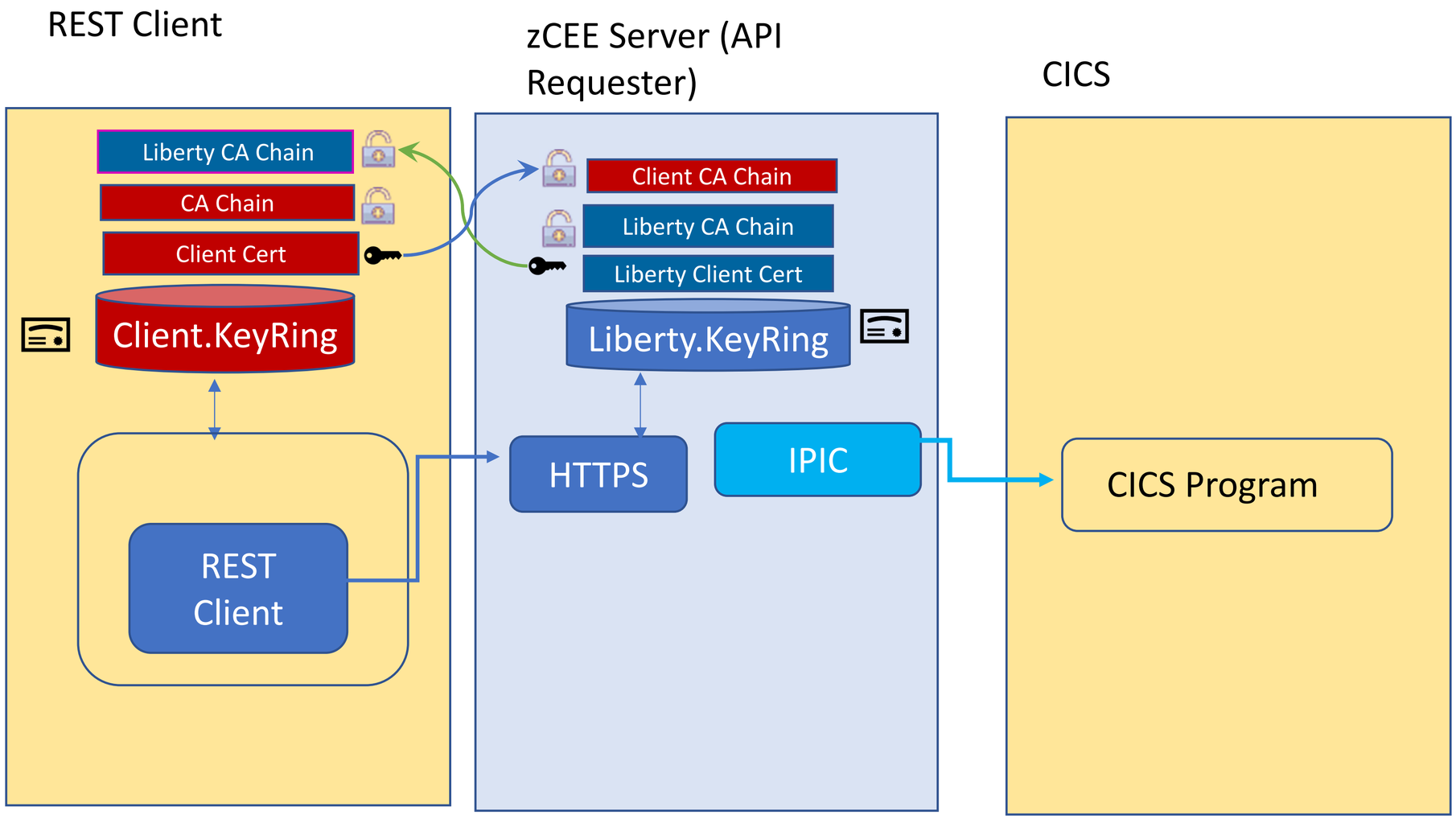



mitchj@us.ibm.com



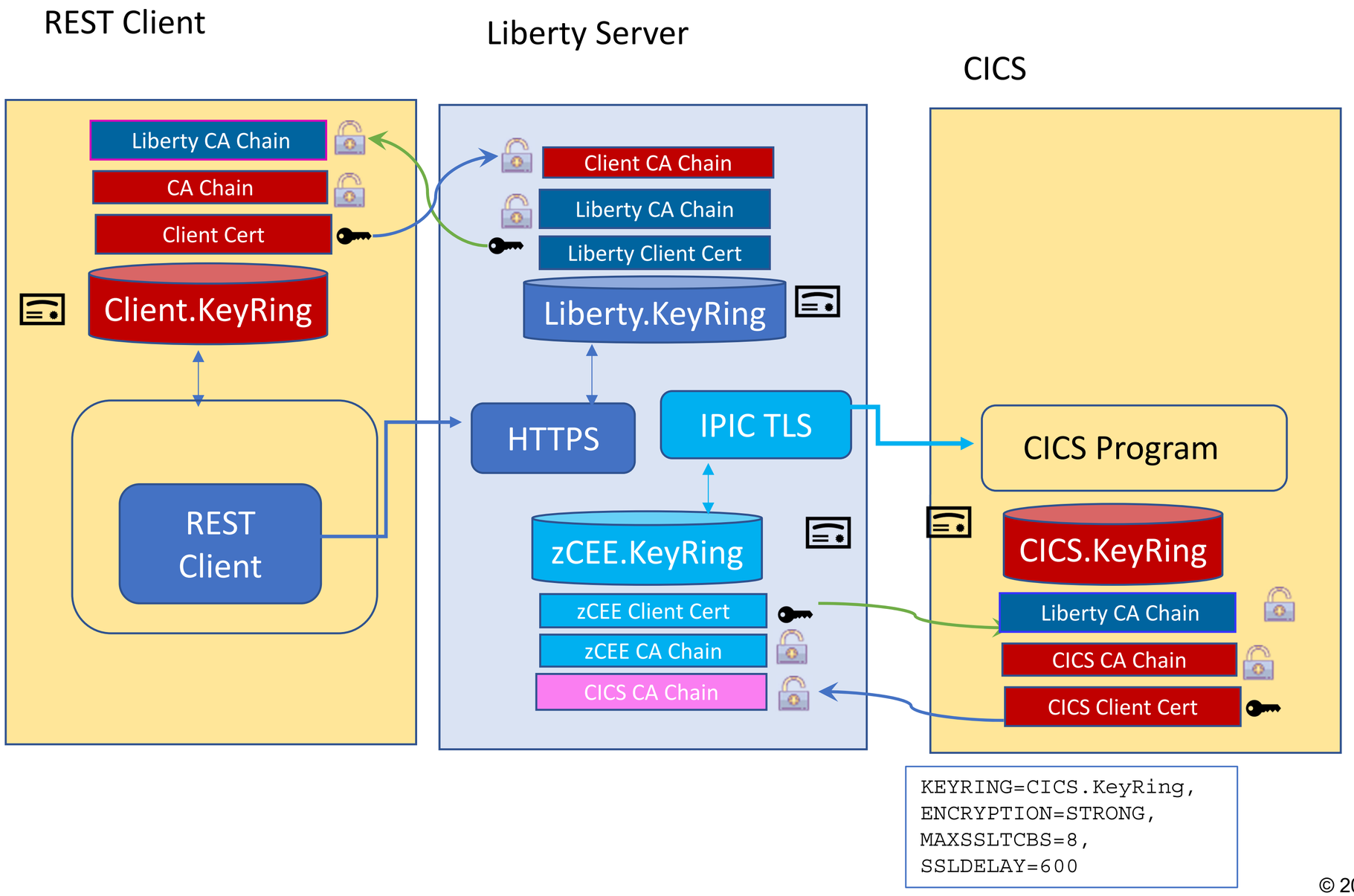

Slide 39

# TLS handshake between the client and z/OS Connect server




mitchj@us.ibm.com

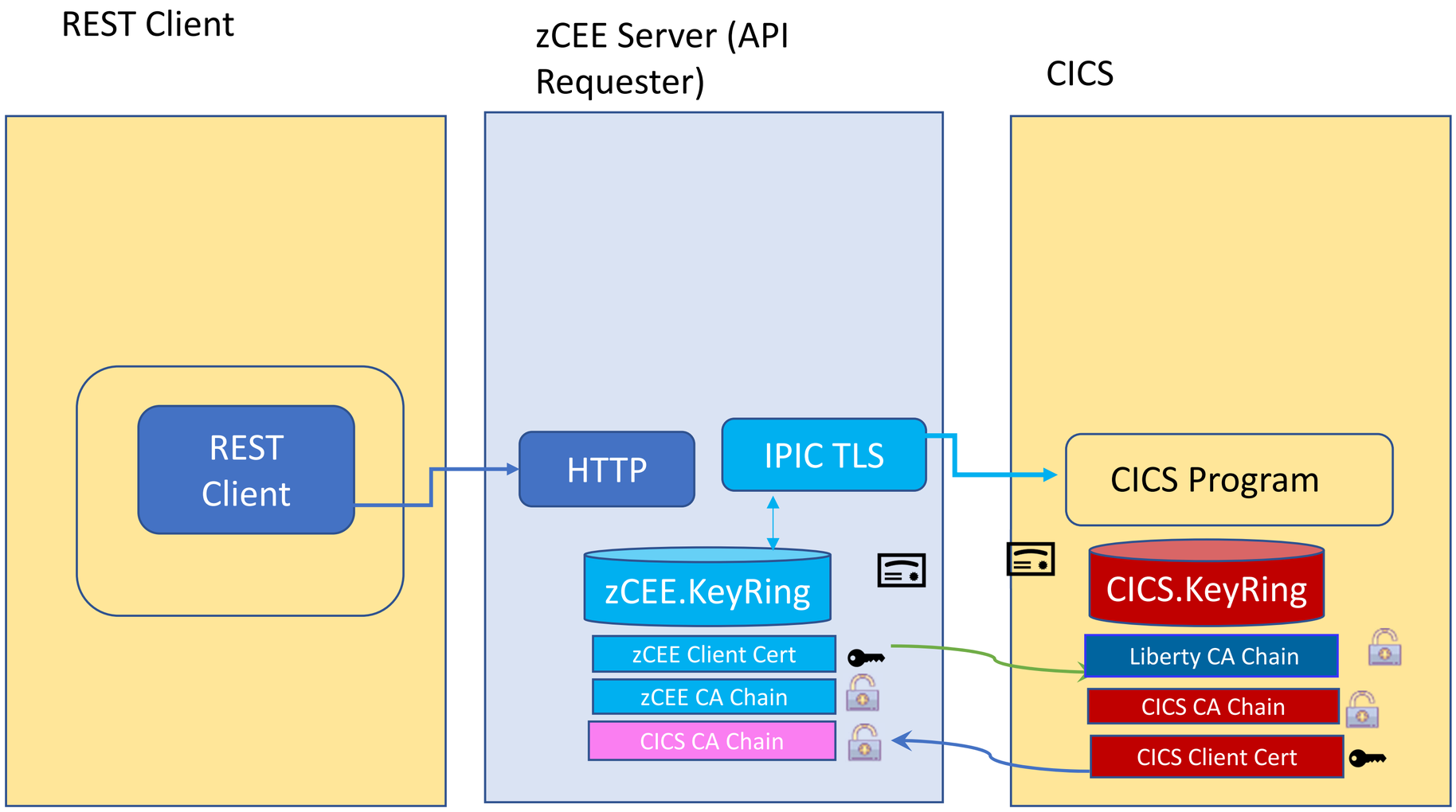
# TLS handshakes between all endpoints




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

mitchj@us.ibm.com

# TLS handshake between the z/OS Connect server and the target endpoint




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

mitchj@us.ibm.com

# 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 the 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
For difference between IBM Java 8 and IBM Semeru 11 and later see URL
https://www.ibm.com/docs/en/semeru-runtime-ce-z/11?topic=security-jsse-provider-differences

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


Slide 45

# 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

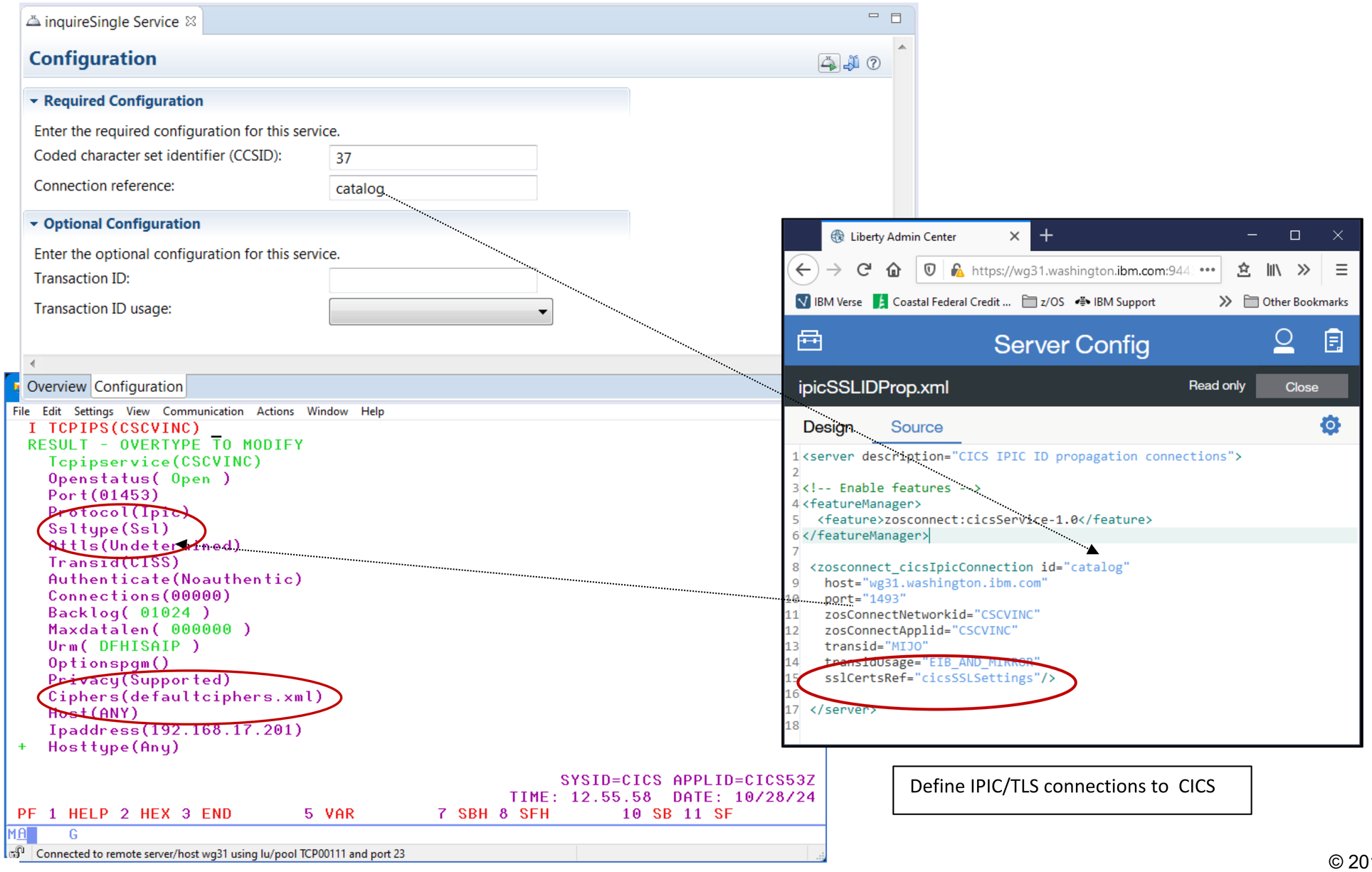
```
<keyStore id="CellDefaultKeyStore"
   location="safkeyringhybrid:///Liberty.KeyRing"
   password="password" type="JCEHYBRIDRACFKS"
   fileBased="false" readOnly="true" />
```

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

mitchj@us.ibm.com

# CICS IPIC using TLS

The server.xml file is the key configuration file:




Define IPIC/TLS connections to CICS

mitchj@us.ibm.com

https://www.ibm.com/docs/en/zos/3.1.0?topic=programming-cipher-suite-definitions

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

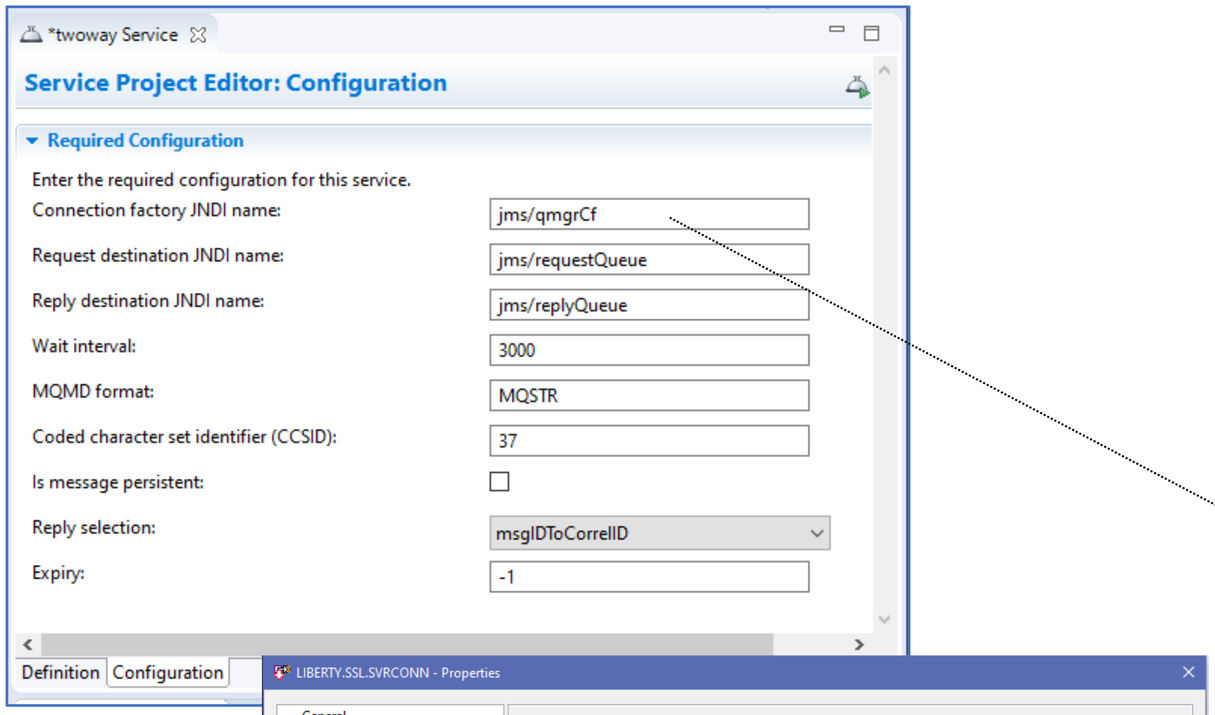
Table 2. 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3

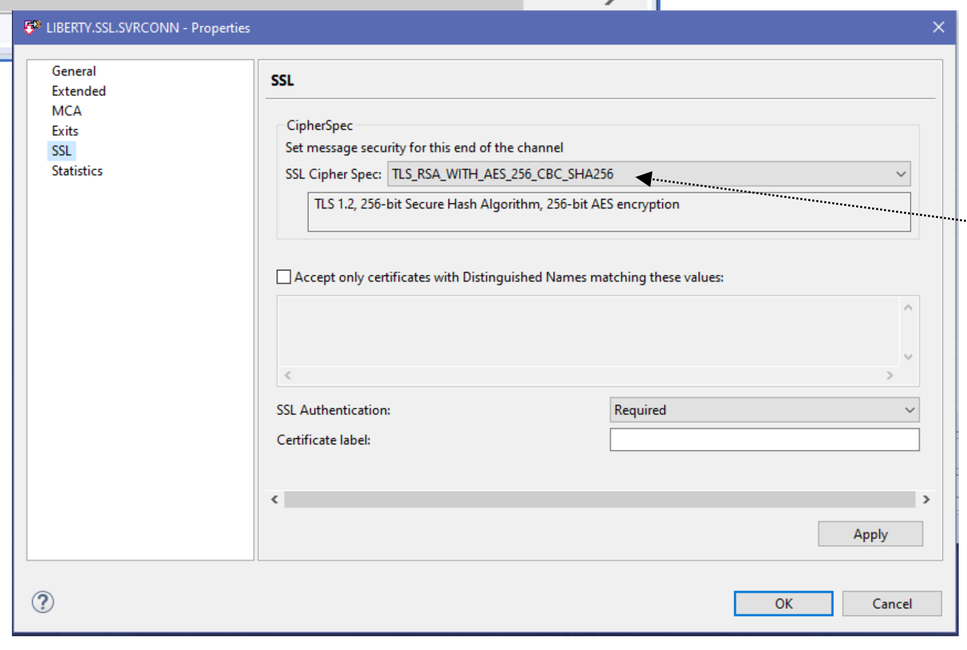
| 2-character cipher number | 4-character cipher number | Short name | Description [1] | FIPS 140-2 | Base security level FMID HCPT510 | Security level 3 FMID JCPT511 |
|---|---|---|---|---|---|---|
| 00 | 0000 | TLS_NULL_WITH_NULL_NULL | No encryption or message authentication and RSA key exchange | | 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 |
| 06 | 0006 | TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 | 40-bit RC2 encryption with MD5 message authentication and RSA (export) key exchange | | X | X |

https://www.ibm.com/docs/en/zos/3.1.0?topic=programming-cipher-suite-definitions

mitchj@us.ibm.com

# MQ JMS using TLS



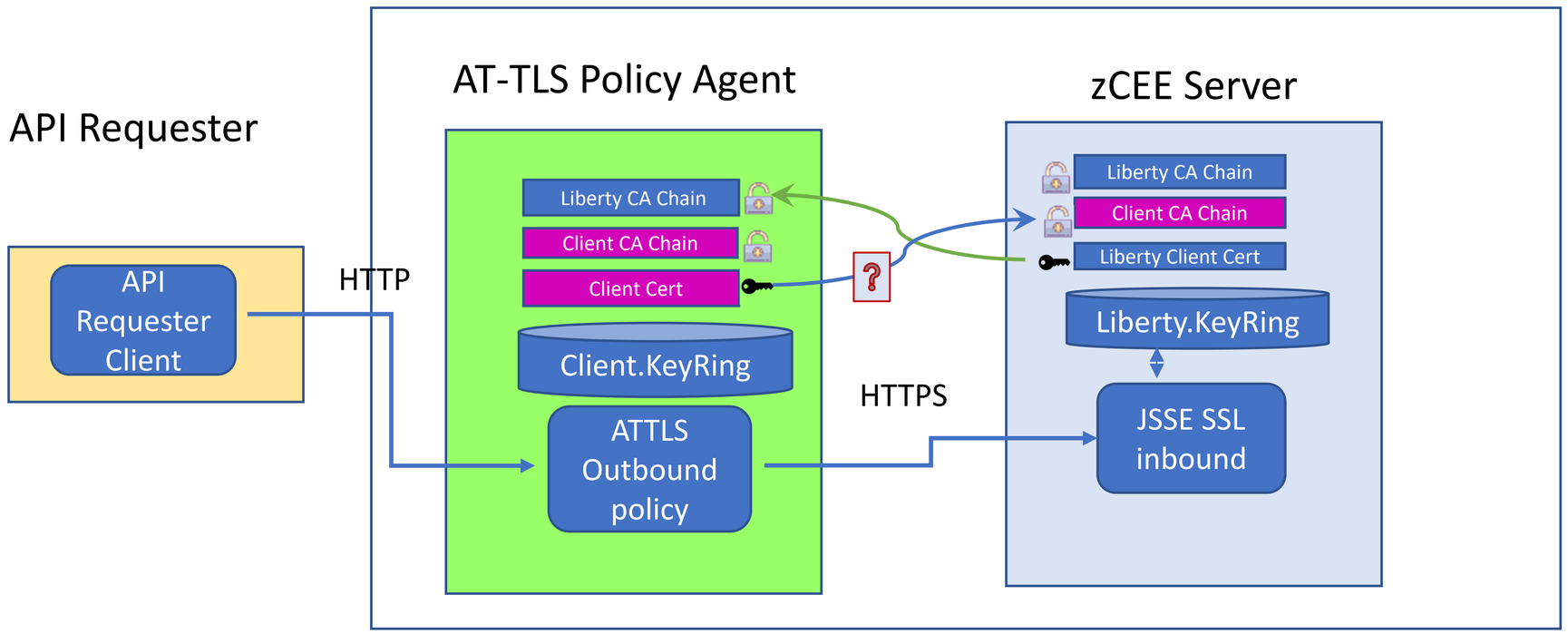




The server.xml file is the key configuration file:


mitchj@us.ibm.com

# AT-TLS - outbound policy handshake scenarios

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



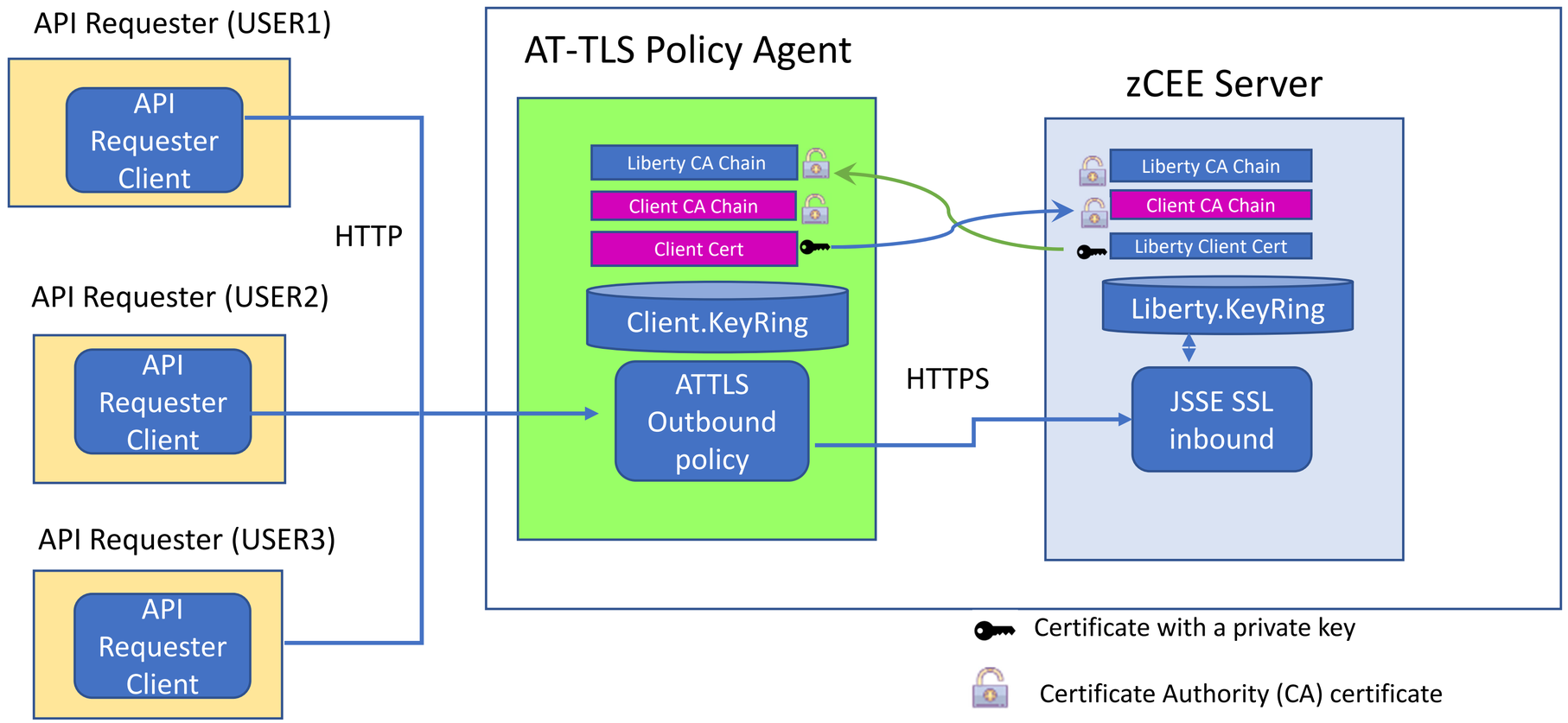

```
<zosconnect_apiRequesters idAssertion="ASSERT_ONLY">
</zosconnect_apiRequesters>
```

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

mitchj@us.ibm.com

# AT-TLS - outbound policy handshake scenario

Use of a common key ring name for multiple client identities



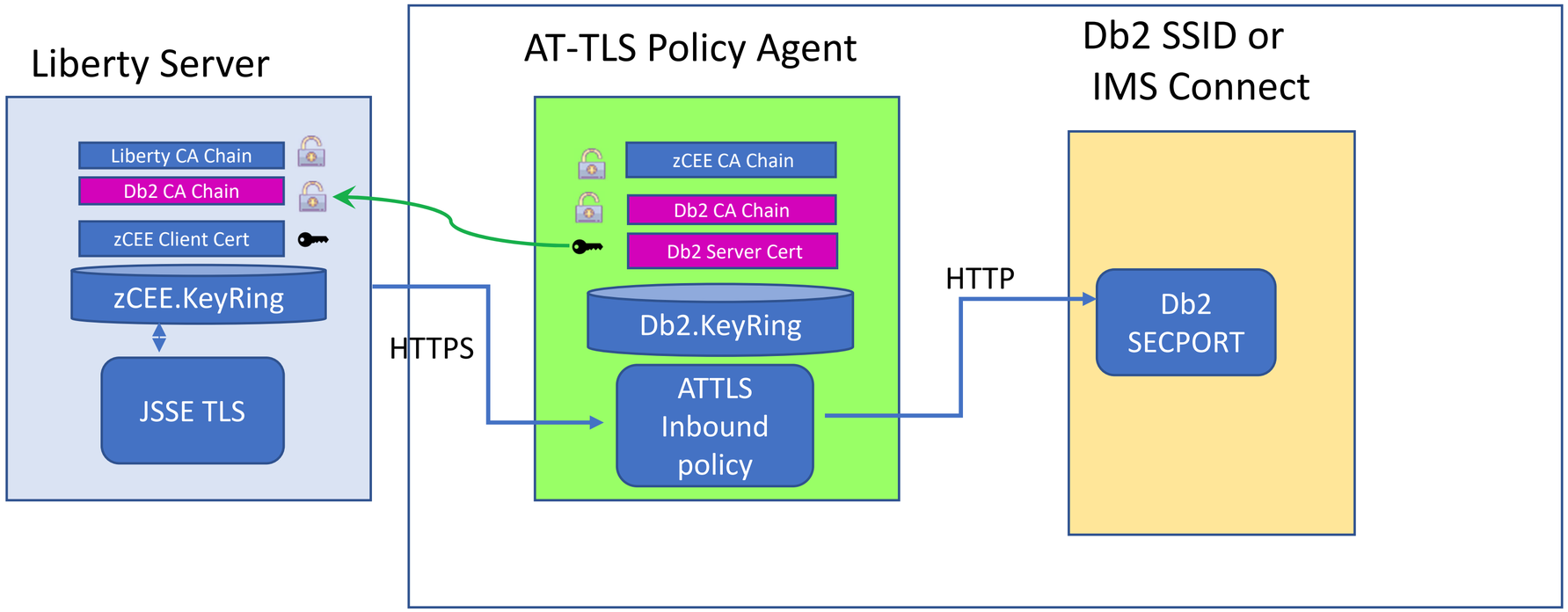

- Each user owns a keyring with the name Liberty.KeyRing.
- Each key ring has a different default client certificate for mutual authentication purposes.

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

mitchj@us.ibm.com

# AT-TLS - inbound policy handshake scenario (Db2 and 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 DB2 is AT-TLS aware
IMS is AT-TLS unware

Certificate with a private key

Certificate Authority (CA) certificate

mitchj@us.ibm.com

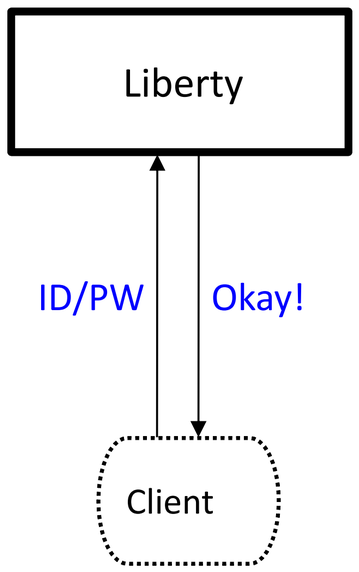



Slide 52

# Authentication - Third Party Authentication

Several different ways this can be accomplished:

## Basic Authentication

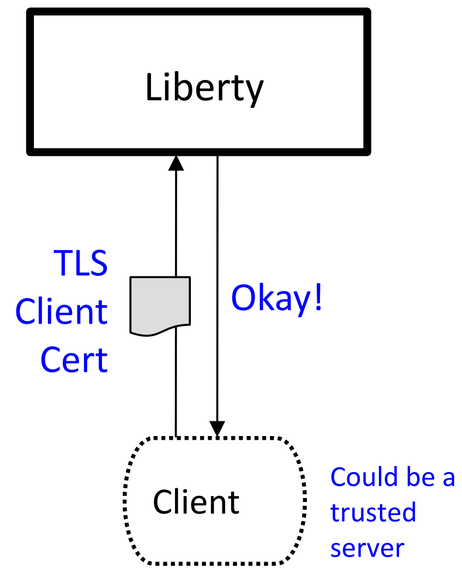



Server prompts for ID/PW

Client supplies ID/PW or ID/PassTicket

Server checks registry:
- Basic (server.xml)
- SAF

## Client Certificate

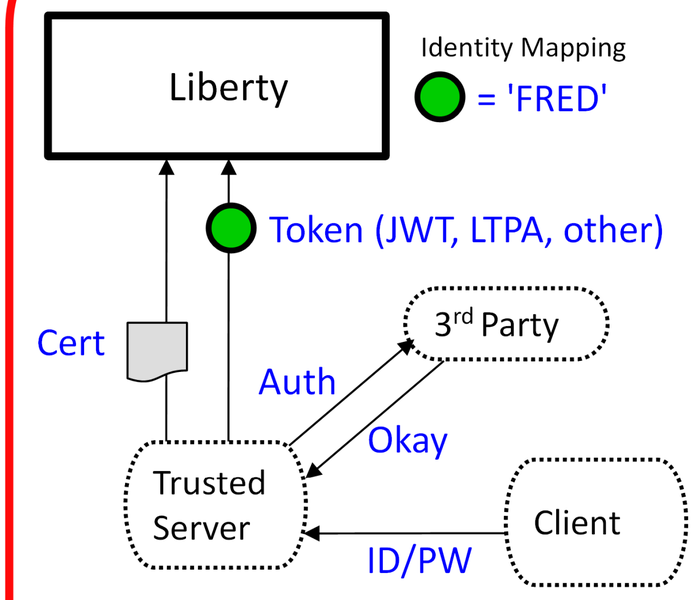



Server prompts for client certificate information

Client supplies certificate information

Server validates client certificate information and maps to an identity

## Third Party Authentication




**Client authenticates to 3rd party sever**

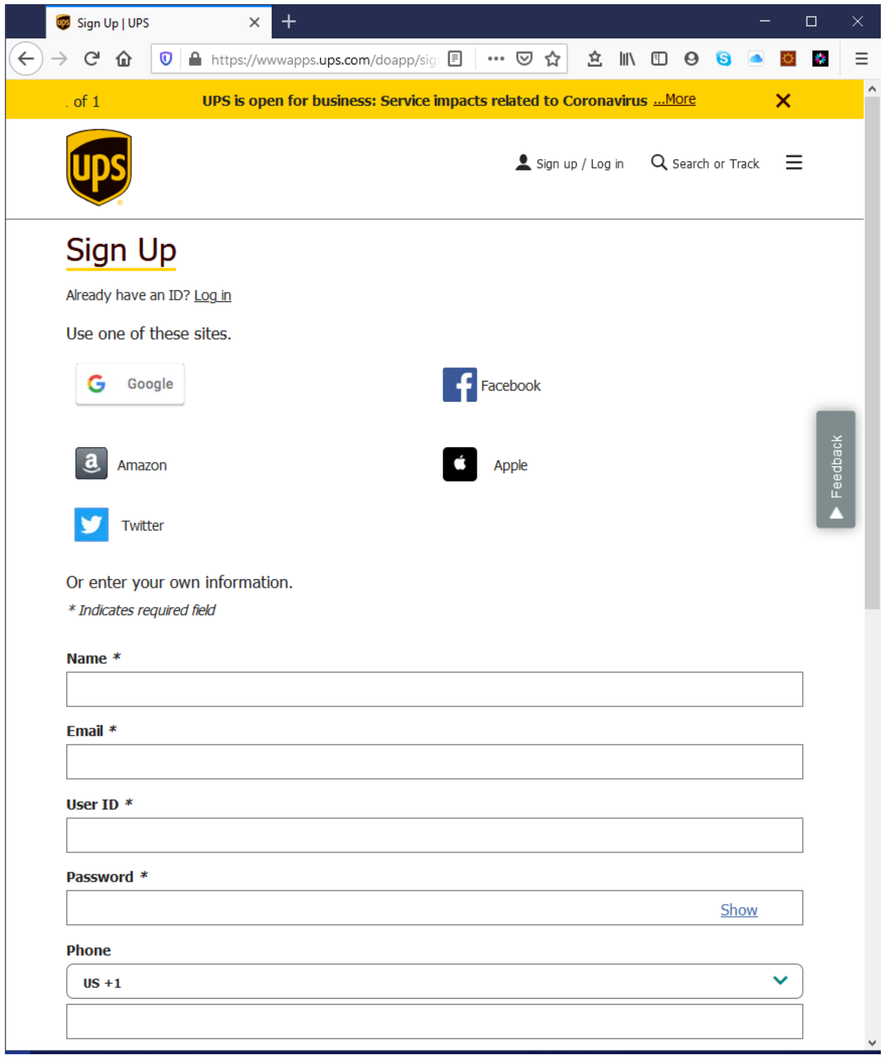
**Client receives a trusted 3rd party token**

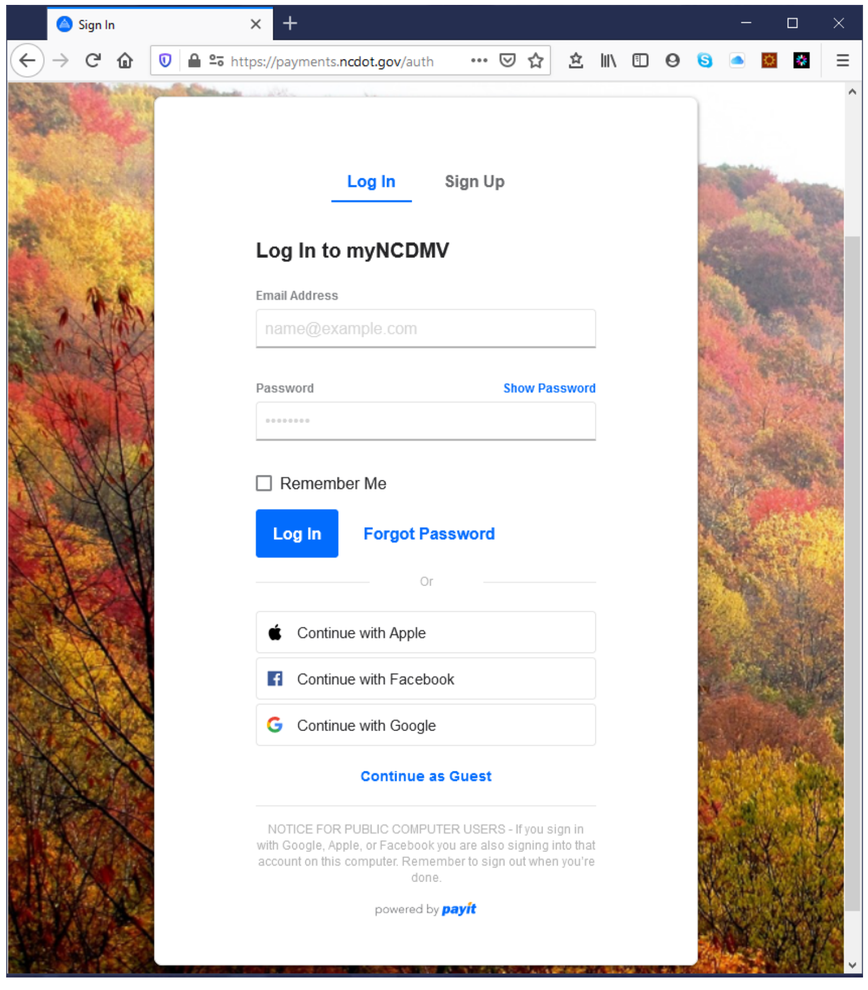
**Token flows to Liberty z/OS and is mapped to an identity**

**Registry options:**
- **We may know these detail.**

mitchj@us.ibm.com

# Third Party Authentication Examples




mitchj@us.ibm.com

# 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.

- ***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

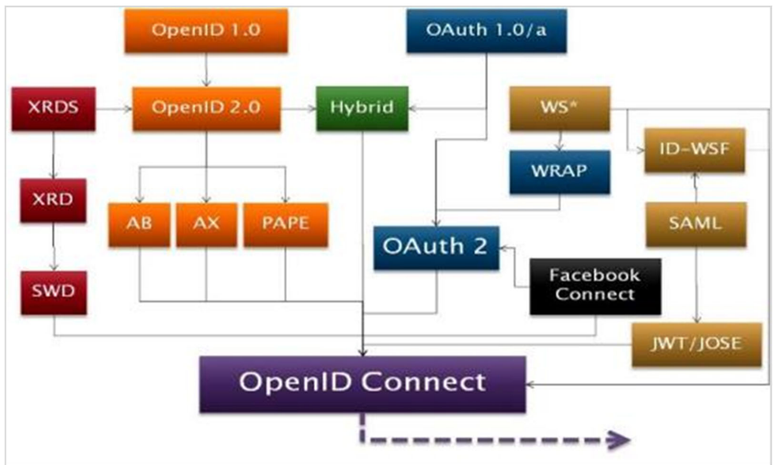
See the YouTube videos:

*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



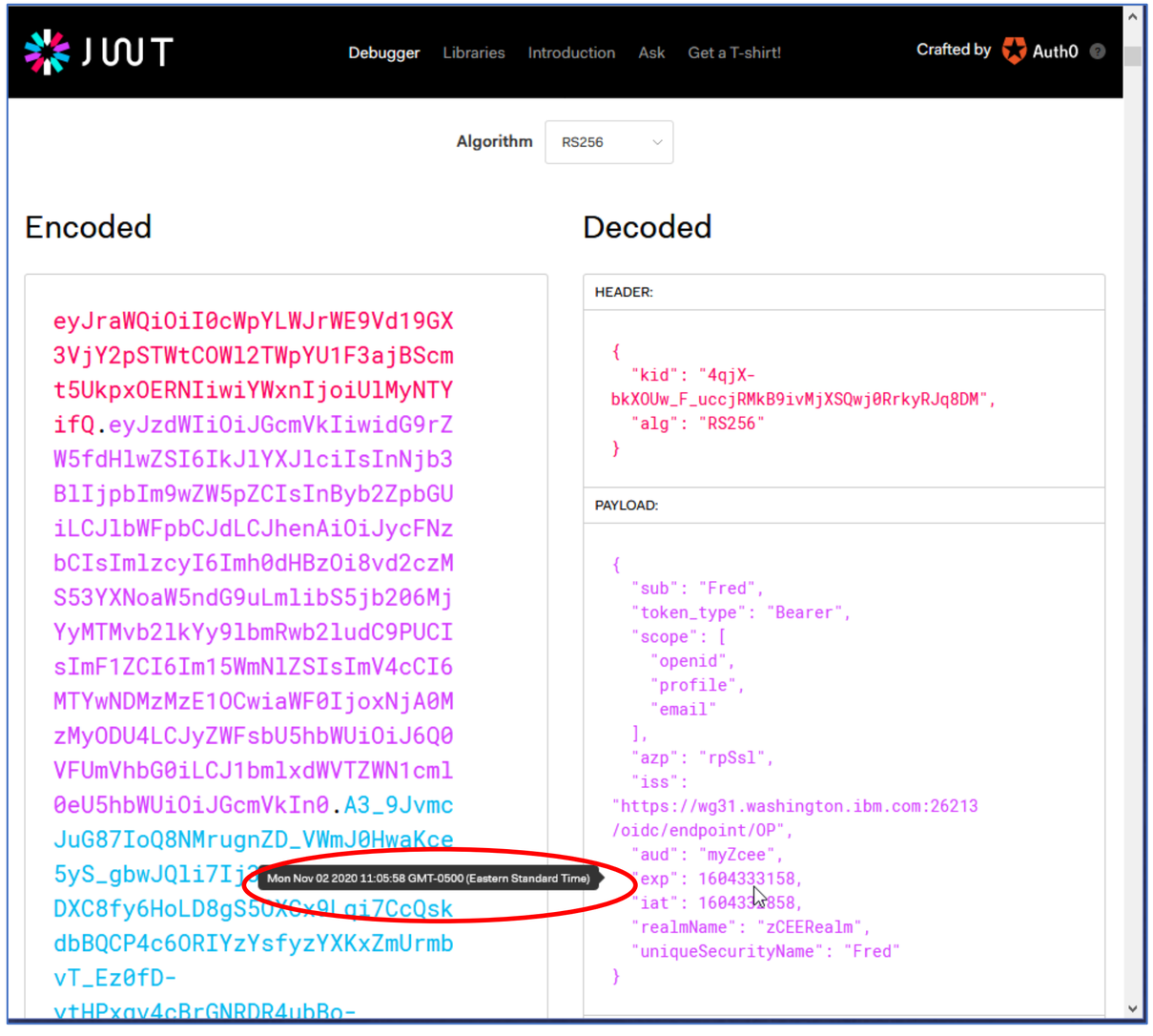

mitchj@us.ibm.com

# 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



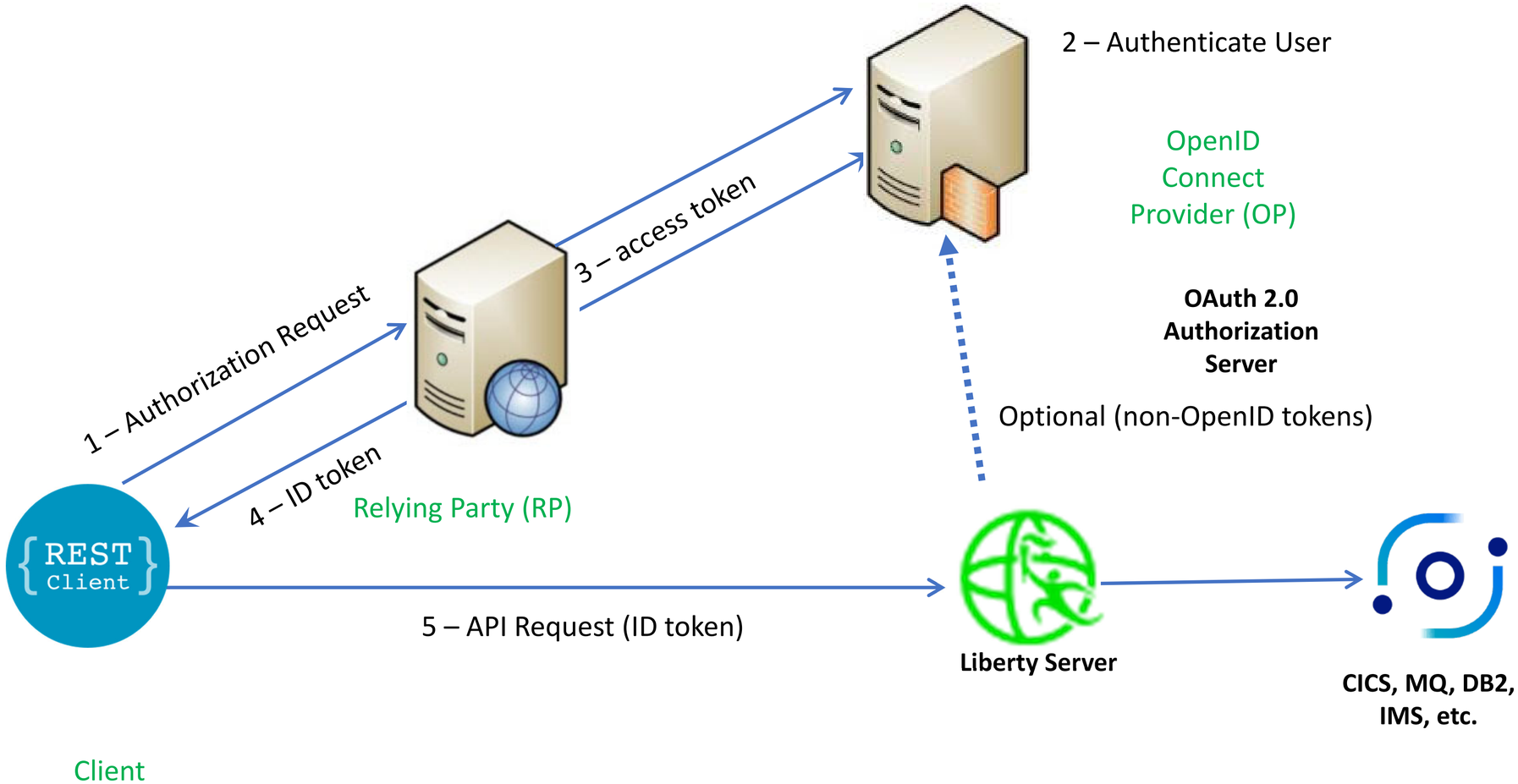

Values derived from the OAUTH configuration:

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

https://jwt.io

mitchj@us.ibm.com

# Typical Authorization Flow for am OpenID Connect token to a Liberty server




mitchj@us.ibm.com

# OpenID Connect/OAuth

- ***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

# Tech-Tip: 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

# Tech-Tip: 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.*

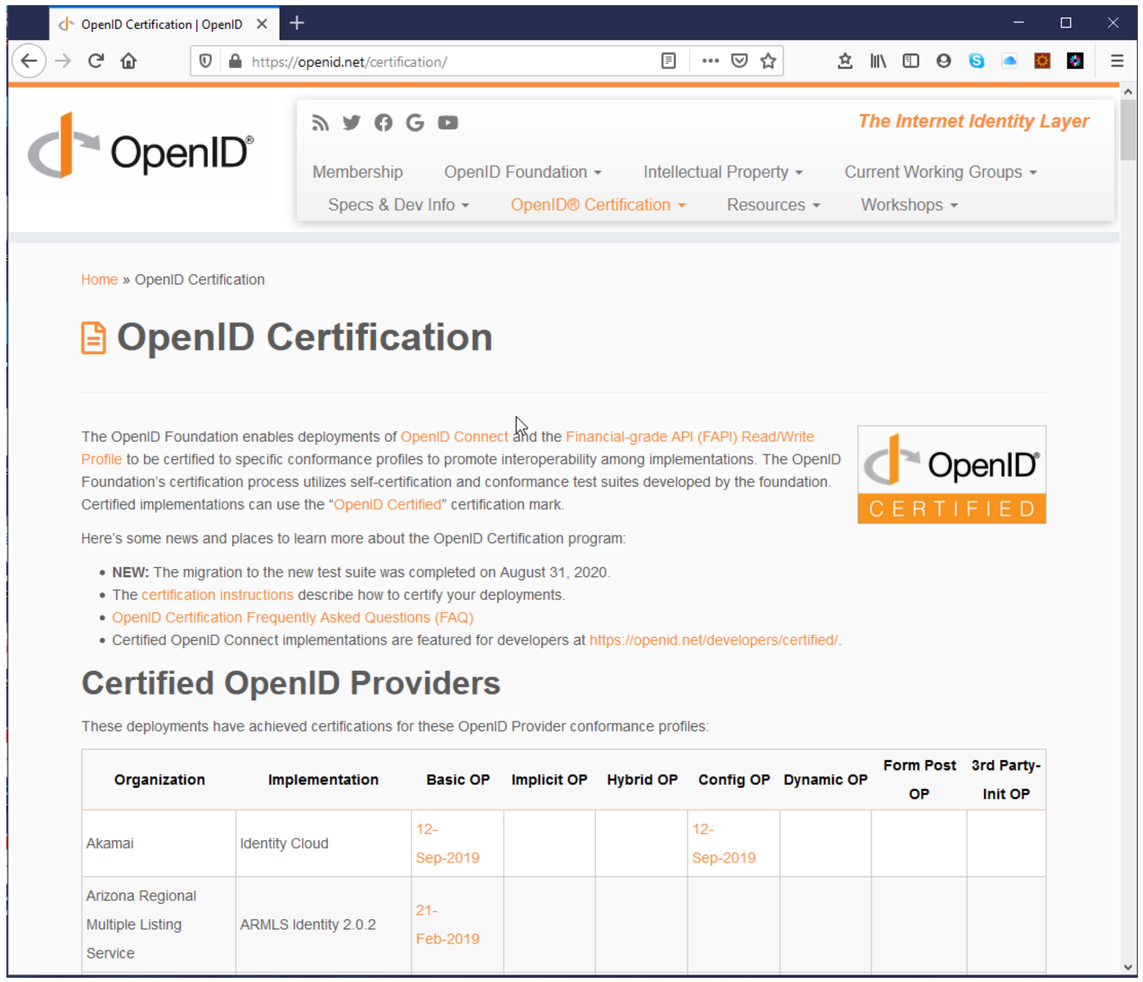
- **OpenID Connect Provider (OP) -** An OAuth 2.0 authorization server that is capable of providing claims to a client or Relying Party (RP) , *an OpenID component.*

- ***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.

# Tech-Tip: There are a multitude of OpenID Certified Providers




https://openid.net/certification/



mitchj@us.ibm.com

# Liberty OpenID Client identity mapping configuration attributes

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "kid": "kvjtqdLMjOTWiJrjOr73fu2MMt-FjiQrxUOYBzJLR4o",
  "alg": "RS256"
}
```

PAYLOAD: DATA

```
{
  "sub": "auser",
  "token_type": "Bearer",
  "scope": [
    "openid",
    "profile",
    "email"
  ],
  "azp": "rpSsl",
  "iss": "https://wg31.washington.ibm.com:26213
/oidc/endpoint/OP",
  "aud": "myZcee",
  "exp": 1646761228,
  "iat": 1646760928,
  "realmName": "zCEERealm",
  "uniqueSecurityName": "auser"
}
```

```
<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, using IDIDMAP resources and the RACMAP command.

```
<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 (SAF) is not used to create the user subject.

mitchj@us.ibm.com

# Tech-Tip: RACMAP Command Summary

```
RACMAP ID(USER1) MAP USERDIDFILTER(NAME('distuser1'))
     REGISTRY(NAME('*')) WITHLABEL('zCEE token user1')
RACMAP ID(USER1) MAP USERDIDFILTER(NAME('distribute_User1'))
     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('USER X500 DN’)

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

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

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

Label: zCEE user1
Distributed Identity User Name Filter:
  >distributed_User1<
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:
  >*<
```

mitchj@us.ibm.com

# Liberty 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"
    httpsRequired="true"
    authFilterRef="jwkAuthFilter"
    inboundPropagation="required"
    signatureAlgorithm="RS256"
    userIdentifier="sub"
    mapIdentityToRegistryUser="true"
    issuerIdentifier=https://wg31.washington.ibm.com:26213/oidc/endpoint/OP
    disableLtpaCookie="true"
    audiences="myZcee"
    tokenReuse="true"
    jwkEndpointUrl=https://wg31.washington.ibm.com:26213/oidc/endpoint/OP/jwk
    jwkClientId="jwtClient"
    jwkSecret="jwtSecret"/>
  </openidConnectClient>
```

mitchj@us.ibm.com

# JWT used in scenario – putting it all together


```
{
  "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

Liberty 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="false"
    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

# Use authorization filters  to associate request for security configurations

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"/>
<openidConnectClient id="RPsslG" . . . authFilterRef= "API Gateway"/>
<openidConnectClient id="RPsslURL" . . . authFilterRef= "URLFilter"/>
<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

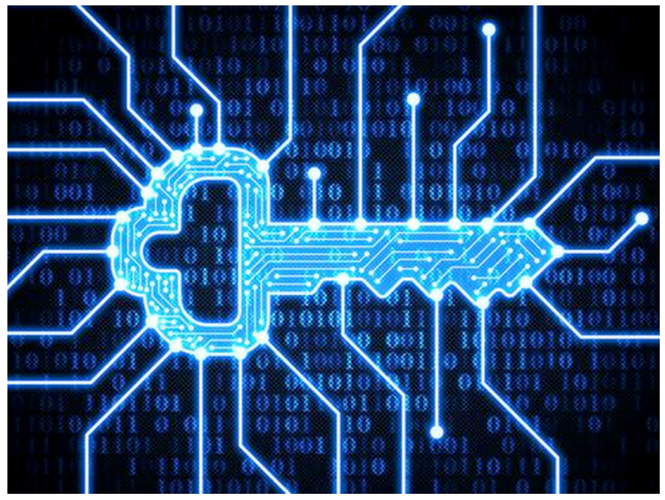
# General security terms or considerations

Security involves

- Identifying who or what is requesting access (**Authentication)**
  - Basic Authentication
  - Mutual Authentication using Transport Layer Security (TLS), formerly known as SSL
  - Third Party Tokens

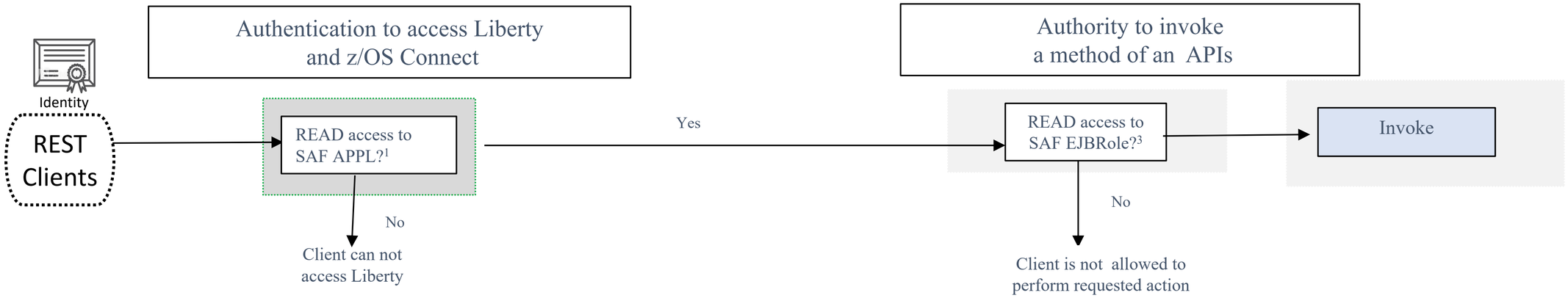
- Ensuring that the message has not been altered in transit (**Data Integrity)** and ensuring the confidentiality of the message in transit (**Encryption)**
  - TLS (encrypting messages and using a digital signature)

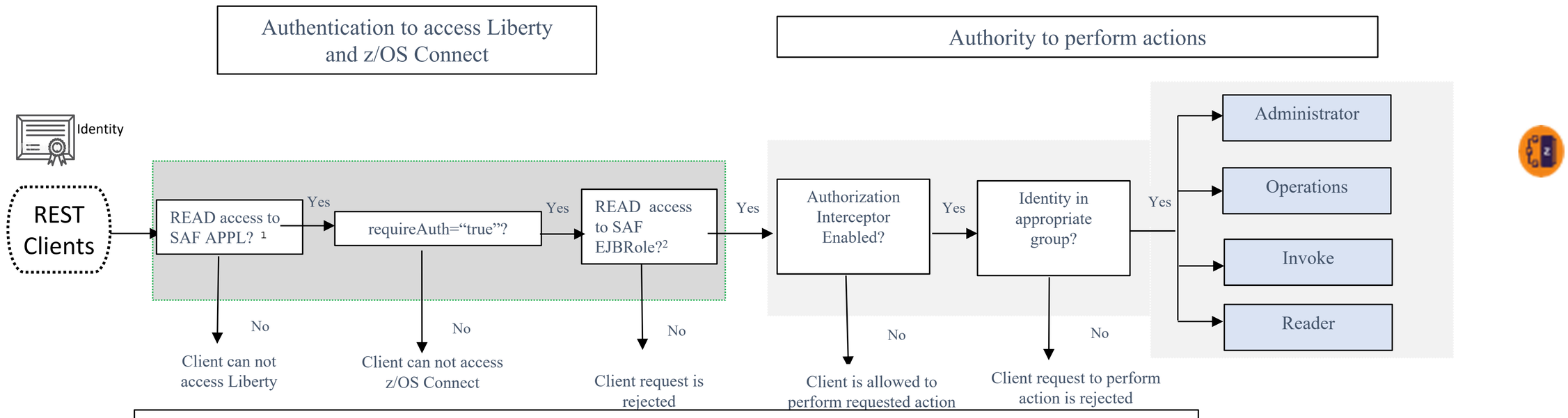
- Controlling access (**Authorization)**
  - Is the authenticated identity authorized to access to z/OS Connect
  - Is the authenticated identity authorized to access a specific API, Services, etc.



mitchj@us.ibm.com

# Security flow – authentication/authorization




[1]RDEFINE APPL *profilePrefix*
[2]**RDEFINE EJBROLE *profilePrefix*.zos.connect.access.roles.zosConnectAccess**
[3]**REDFINE EJBROLE *profilePrefix.resourceName.role***

[1] mitchj@us.ibm.com

# Role based authorization (EJBRole)

Once an identity has been authenticated the System Authorization Facility (SAF) role mapper (*safRoleMapper*) configuration element is used to determine authorization.

- Authorization is based on access to a SAF EJBRole resource  where the resource name is generated from application-defined role names.
- Read access to the EJBRole is required to access the resource protected by the application-define role name.
- The role mapper generates a SAF EJBRole name and does a SAF authorization for that resource for the access permitted for the identity.

```
<safRoleMapper profilePattern="%profilePrefix%.%resource%.%role%"/>
```

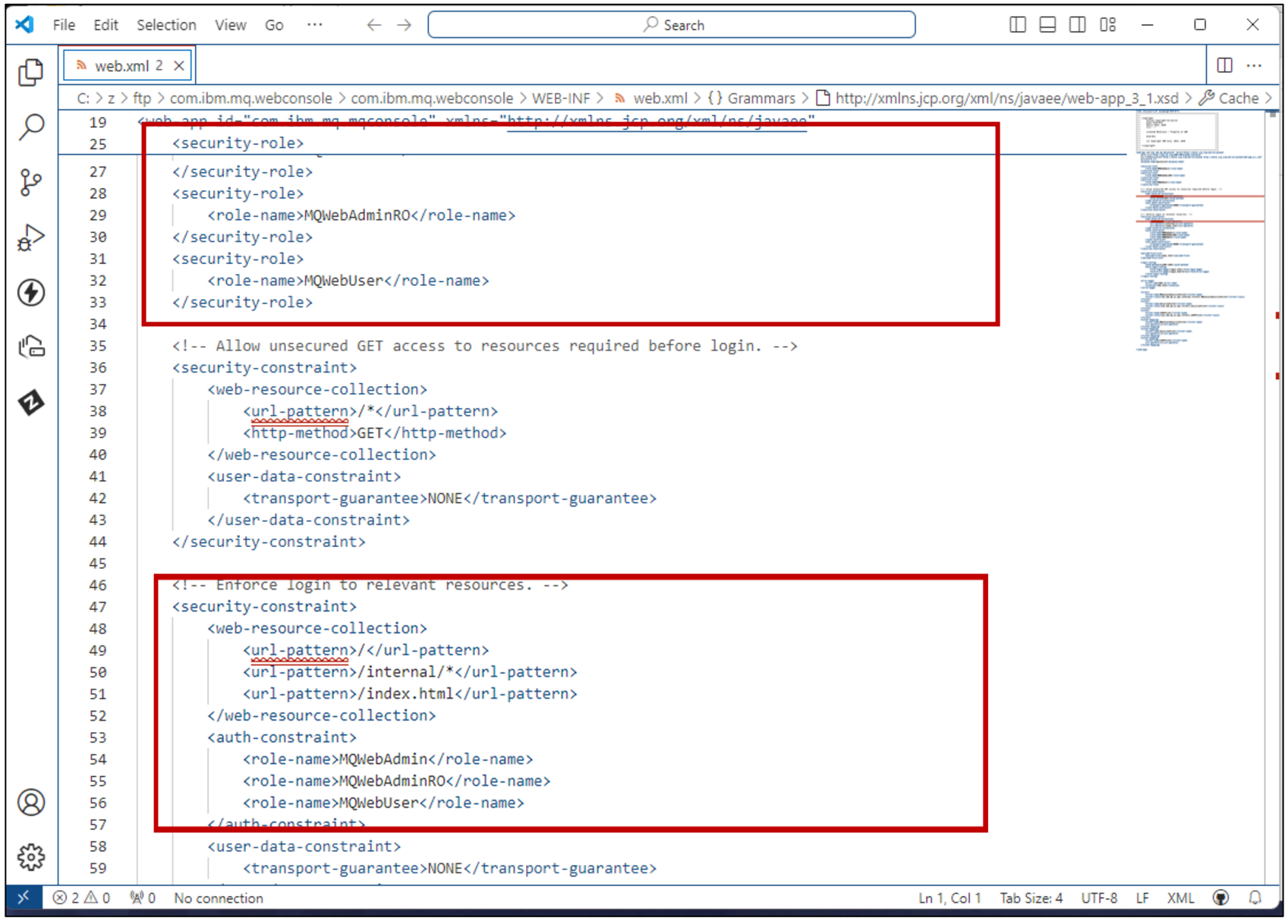
Where:

- ***profilePrefix***, is derived form the *profilePrefix* attribute in the <safCredentials/> element, e.g., <*safCredentials profilePrefix=“BBGZDFLT“/>*
- ***resource*** is derived from various sources, e.g., the value of the *id* attribute of an <enterpriseApplication/> or <webApplication/>  configuration elements. For security administration, the resource name is *com.ibm.ws.management.security.resource*.
- ***role*** is derived from the contents of the *web.xml* file embedded in an enterprise archive (EAR) or web archive (WAR) file

So, the required SAF EJB roles for security administration would be:

- RDEFINE EJBROLE BBGZDFLT.com.ibm.ws.management.security.resource.Administrator
- RDEFINE EJBROLE BBGZDFLT.com.ibm.ws.management.security.resource.Reader
- RDEFINE EJBROLE BBGZDFLT.com.ibm.ws.management.security.resource.allAuthenticatedUsers

# Tech-Tip: The role names are embedded in the application's WAR or EAR files




- MQWebAdmin
- MQWEBAdminRO
- MQWebUser

WAR - **W**eb application **AR**chive: a compressed file that contains the libraries, JAR files, JSP, etc. that an application requires for deployment.

EAR – **E**nterprise application **AR**chive: a compressed file that contains the libraries, enterprise beans, and JAR files that an application requires for deployment.

mitchj@us.ibm.com

# Security for MQ Console and REST

**/usr/lpp/mqm/web/mq/etc/mqweb.xml**

```
<enterpriseApplication id="com.ibm.mq.console" location="${wlp.install.dir}/mq/apps/com.ibm.mq.webconsole.ear"
name="com.ibm.mq.console" . . .
</enterpriseApplication>

 <enterpriseApplication id="com.ibm.mq.rest" location="${wlp.install.dir}/mq/apps/com.ibm.mq.rest.ear"
   name="com.ibm.mq.rest"  . . .
 </enterpriseApplication>
```

**/var/mqm/servers/mqweb/mqwebuser.xml**

<safCredentials profilePrefix="**MQWEB**" unauthenticatedUser="WSGUEST"/

So, the required SAF EJB roles to be defined would be:

- REDFINE EJBROLE *MQWEB.com.ibm.mq.console.MQWebAdmin*
- REDFINE EJBROLE *MQWEB.com.ibm.mq.console.MQWebAdminRO*
- REDFINE EJBROLE *MQWEB.com.ibm.mq.console.MQWebUser*
- REDFINE EJBROLE *MQWEB.com.ibm.mq.rest.MFTWebAdmin*
- REDFINE EJBROLE *MQWEB.com.ibm.mq.rest.MFTWebAdminRO*
- REDFINE EJBROLE *MQWEB.com.ibm.mq.rest.MQWebAdmin*
- REDFINE EJBROLE *MQWEB.com.ibm.mq.rest.MQWebAdminRO*
- REDFINE EJBROLE *MQWEB.com.ibm.mq.rest.MQWebUser* .

https://www.ibm.com/docs/en/ibm-mq/9.3?topic=roles-mq-console-rest-api

mitchj@us.ibm.com

# Security roles for z/OS Connect OpenAPI3 are defined manually

https://www.ibm.com/docs/en/zos-connect/zos-connect/3.0?topic=authorization-how-define-roles

## About this task

Define the roles required to invoke each API operation. The required roles are assigned in the APIs OpenAPI definition by using a custom z/OS Connect OpenAPI 3.0 specification extension called `x-ibm-zcon-roles-allowed`.

This task illustrates how to:

- Define roles that apply to all operations of an API.
- Define roles for specific API operations.
- Define a generic role that allows all authenticated users to start an API or specific API operations when using an LDAP or a basic user registry.

**Warning:** You must define roles for user authentication to be enforced. Any API operation that does not have either a specific or API level role definition can be started by unauthenticated users.

## Procedure

1. Locate and open the OpenAPI document.
   If the OpenAPI document isn't imported into the Designer UI, then this is your original OpenAPI document.
   If the OpenAPI document is imported into the Designer UI, then this is the `openapi.yaml` or `openapi.json` file in the API project `src/main/api` directory. This might be in your local Designer workspace or might be stored in a Source Control Manager.
   Open the OpenAPI document in edit mode.

2. Optional: Assign the roles that apply to all operations in the API.
   Define the `x-ibm-zcon-roles-allowed` in the root of the OpenAPI definition, where the value is an array of role names.
   Example 1: to add the Staff role as the default role for all operations in the API by using YAML.

4. Regenerate the API to use the specified roles.
   If you are using the Designer, the API is automatically regenerated to use the specified roles.
   If you are making this change as part of a DevOps pipeline, follow your usual procedure to generate the API from the API project.

mitchj@us.ibm.com

# z/OS Connect Designer basic security xml (1 of 2)

```
<authorization-roles id="Manager">
        <security-role name="Manager">
            <group name="managerGroup"/>
        </security-role>
    </authorization-roles>
    <authorization-roles id="Staff">
        <security-role name="Staff">
            <group name="staffGroup"/>
        </security-role>
    </authorization-roles>

</server>
```

*cscvinc.yaml - Notepad

File Edit Format View Help

```
openapi: 3.0.1
info:
  title: cscvinc
  description: ""
  version: 1.0.0
servers:
- url: "/"
x-ibm-zcon-roles-allowed:
- Manager
paths:
  /cscvinc/employee:
    post:
      tags:
      - cscvinc
      operationId: postCscvincInsertService
      x-ibm-zcon-roles-allowed:
        - Staff
      parameters:
......
  /cscvinc/employee/{employee}:
    get:
      tags:
      - cscvinc
      operationId: getCscvincSelectService
      x-ibm-zcon-roles-allowed:
        - Staff
 ....
    put:
      tags:
      - cscvinc
      operationId: putCscvincUpdateService
      x-ibm-zcon-roles-allowed:
        - Staff

........
```

Ln 35, Col 9 | 100% | Unix (LF) | UTF-8

mitchj@us.ibm.com

# z/OS Connect Designer basic security XML (2 of 2)

```
<server description="basic security">

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

    <webAppSecurity allowFailOverToBasicAuth="true" />

    <basicRegistry id="basic" realm="zosConnect">
        <user name="Fred" password="fredpwd" />
        <user name="user1" password="user1" />
        <user name="user2" password="user2" />
        <user name="user3" password="user3" />
        <group name="Manager">
           <member name="Fred"/>
        </group>
        <group name="Staff">
           <member name="Fred"/>
           <member name="user1"/>
           <member name="user2"/>
        </group>
    </basicRegistry>
```

mitchj@us.ibm.com

# Security for OpenAPI 3 z/OS Connect APIs is configured using Liberty elements

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

<webApplication id="CatalogManager" location="${server.config.dir}/apps/api.war" contextRoot="catalog"
name="CatalogManager"/>

<safRoleMapper profilePattern=%profilePrefix%.%resourceName%.%role%
```

The *name* attribute of the webApplication for the deployed WAR file determines the name of the EJBRoles used manage access to the API's methods.

```
openapi: 3.0.0
. . .
servers:
- url: /
x-ibm-zcon-roles-allowed:
- Manager
. . .
paths:
  /items:
    get:
    . . .
  /items/{id}:
    get:
     . . .
     operationId: itemsIdGet
     x-ibm-zcon-roles-allowed:
        - Staff
 /orders:
     post:
     . . .
     operationId: ordersPost
     x-ibm-zcon-roles-allowed:
       - Staff
```

*From the OpenApi document, the value for %role% would be either* ***Manager*** *or* ***Staff****.*

So, the required SAF EJB roles to be defined would be:

- *BBGZDFLT.CatalogManager.**Manager***
- *BBGZDFLT.CatalogManager.**Staff***

```
RDEFINE EJBROLE BBGZDFLT.CatalogManager.Manager
RDEFINE EJBROLE BBGZDFLT.CatalogManager.Staff
```

Access to use the GET method to invoke /items would require read access to EJB role *BBGZDFLT.CatalogManager.Manager.*
Access to use the GET method to invoke /items/{id} and the POST method to invoke /orders would require read access to EJB role *BBGZDFLT.CatalogManager.Staff.*

mitchj@us.ibm.com

# Deploying multiple APIs in the same native server (OpenAPI 3)

```
<webApplication id="catalogManager" name="catalogManager"
location="${server.config.dir}/apps/catalogManager.war" contextRoot="/catalogManager" />
<webApplication id="db2API" name="db2API"
              location="${server.config.dir}/apps/employees.war" contextRoot="/db2" />
<webApplication id="cicsAPI" name="cicsAPI"
              location="${server.config.dir}/apps/api.war" contextRoot="/cics" />
```

catalogManager.war

```
/META-INF/openapi.yaml
openapi: 3.0.0
. . .
servers:
- url: /
. . .
```

catalogManager.war

```
/META-INF/openapi.yaml
openapi: 3.0.0
. . .
servers:
- url: /catalogManager
. . .
```

employees.war

```
/META-INF/openapi.yaml
openapi: 3.0.0
. . .
servers:
- url: /
. . .
```

employees.war

```
/META-INF/openapi.yaml
openapi: 3.0.0
. . .
servers:
- url: /db2
. . .
```

cscvinc.war

```
/META-INF/openapi.yaml
openapi: 3.0.0
. . .
servers:
- url: /
. . .
```

cscvinc.war

```
/META-INF/openapi.yaml
openapi: 3.0.0
. . .
servers:
- url: /cics
. . .
```

mitchj@us.ibm.com

The ***url*** attribute in each WAR's ***openapi.yaml*** file must be unique and match the ***webApplication***'s ***contextRoot*** value

# z/OS Connect OpenAPI 2 authorization is based on group access

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="SYSPGRP " globalOperationsGroup="GBLOPERS"
          globalInvokeGroup="GBLINVKE" globalReaderGroup="GBLRDR"/>
```

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"/
```

[1] mitchj@us.ibm.com

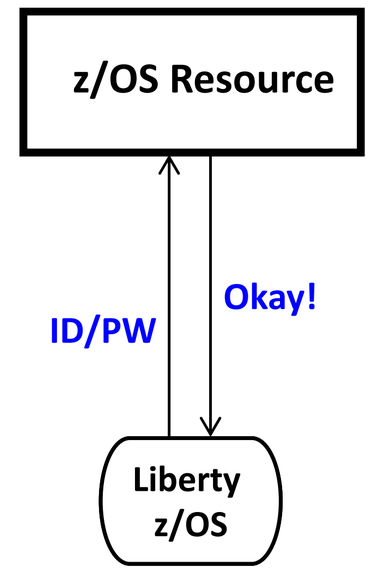
# Security when accessing z/OS subsystems

mitchj@us.ibm.com

# Accessing z/OS resources

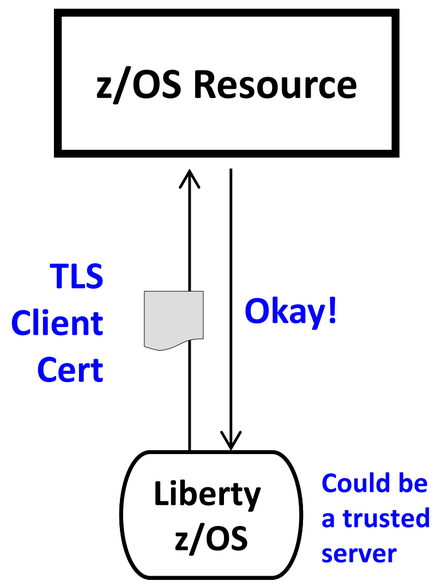
**Several different ways this can be accomplished:**

**Basic Authentication**




**Liberty supplies ID/PW or ID/PassTicket**

**Client Certificate**




**Server prompts for certificate**

**Liberty supplies certificate**

mitchj@us.ibm.com

# Liberty Basic authentication - Identity and Password

Server XML Configuration elements where basic authentication can be provided.

```
<resourceAdapter autoStart="true" id="eciResourceAdapter" location="/usr/lpp/cicstg/ctg93/deployable/cicseci.rar"/>
<library id="DB2JCCLib">
   <fileset dir=“/usr/lpp/db2/jdbc/classes includes="db2jcc4.jar db2jcc_license_cisuz.jar"/>
</library>

<connectionFactory id=“cics">
<properties.eciResourceAdapter connectionURL=“tcp://wg31.Washington.ibm.com” port=“2006” serverName=“CICSZ” userName="identity"
password="password"/>
</connectionFactory>

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

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

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

<dataSource id="DefaultDataSource" jndiName="jdbc/db2"">
<jdbcDriver libraryRef="DB2JCCLib"/>
   <properties.db2.jcc databaseName="SAMPLEDB" serverName="localhost" portNumber="50000"
         user=“identity" password="password"/>
</dataSource>
```

mitchj@us.ibm.com

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

# z/OS Connect 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>

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

<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"/>

<zosconnect_db2Connection id="Db2" host=“wg31.Washington.ibm.com” port=‘2446”
    userName="identity"  password="password"/>
```

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|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 myPassWord

    {aes}AHO0aXdiVD96u4oMRhoKeYH3U7aDqtFXTuHFBsO98Wlb
  - Support was added at Liberty 22.0.0.1 for storing an AES password encryption key in a SAF key ring, see URL https://www.ibm.com/docs/en/was-liberty/zos?topic=slia-storing-aes-password-encryption-key-in-saf-key-ring

    ./securityUtility encode --encoding=aes --keyring=safkeyring://JOHNSON/Liberty.KeyRing --keyringType=JCERACFKS
    --keyLabel="Johnson Client Cert" myPassWord
  - Also supports 1-way hash encoding – for passwords in server.xml with basicRegistry
    - For hash, use encode –encoding=hash

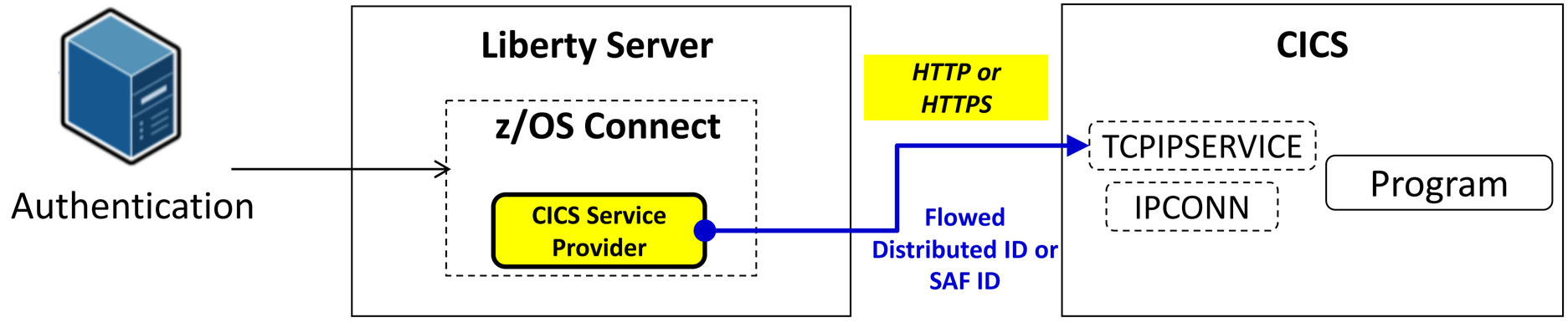
    ./securityUtility encode –-encoding=hash XXXXXXXX

    {hash}ATAAAAAIcqTmHn5qZahAAAAAIMjzy+hP8YFaIO6LiCreVe4etRLUS9a25eVuYtx6WKiv

See the WebSphere Application Server for z/OS Liberty *securityUtility* command at URL:
https://www.ibm.com/docs/en/was-liberty/zos?topic=applications-securityutility-command

mitchj@us.ibm.com

# Flowing a user ID with CICS service provider



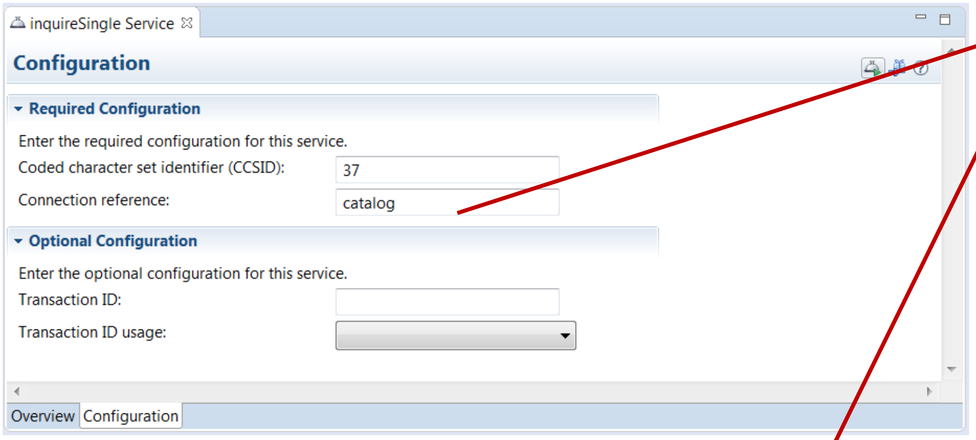

Distributed identities can be propagated to CICS and then mapped to a RACF user ID by CICS. You can then view the distinguished name and realm for a distributed identity in the association data of the CICS task. **Important**: If the z/OS Connect  server is not in the same Sysplex as the CICS system,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

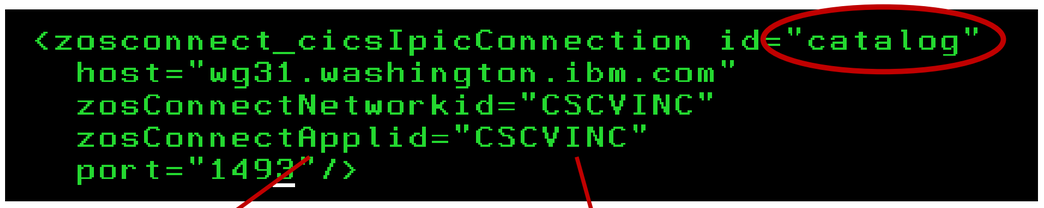
If a SAF ID is used for authentication (e.g., basic authentication with a SAF registry) then the SAF ID is passed to CICS.

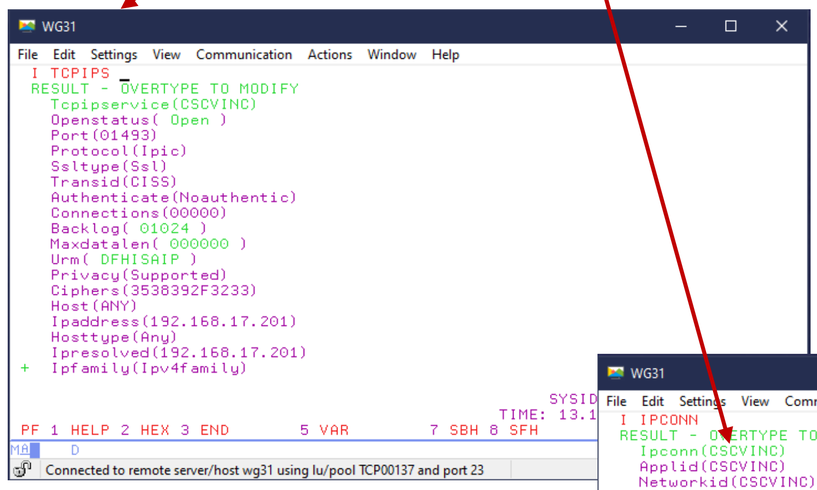
[1] mitchj@us.ibm.com

# Flowing an identity to CICS

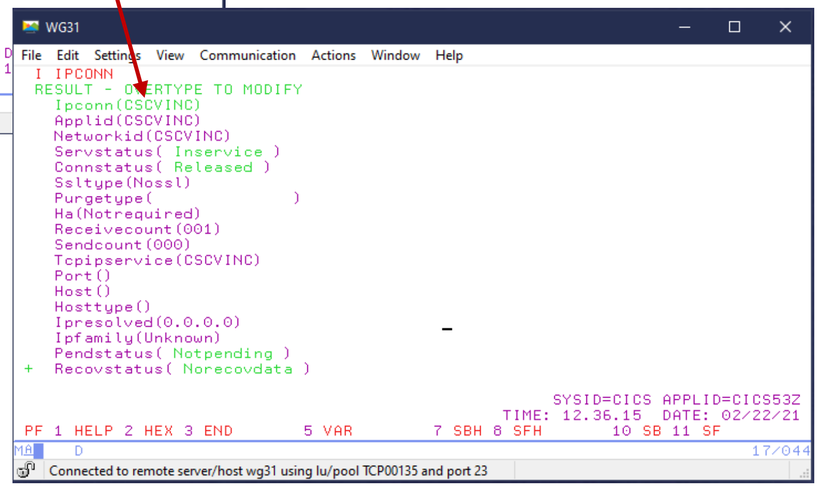
The zosconnect_cicsIpicConnection element is the key :



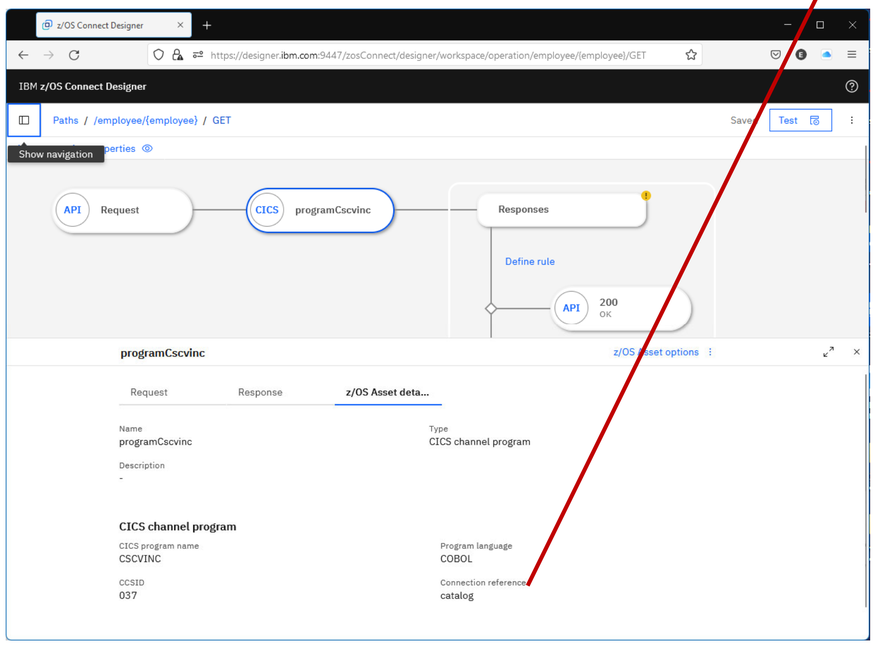




CICS TCPIPService

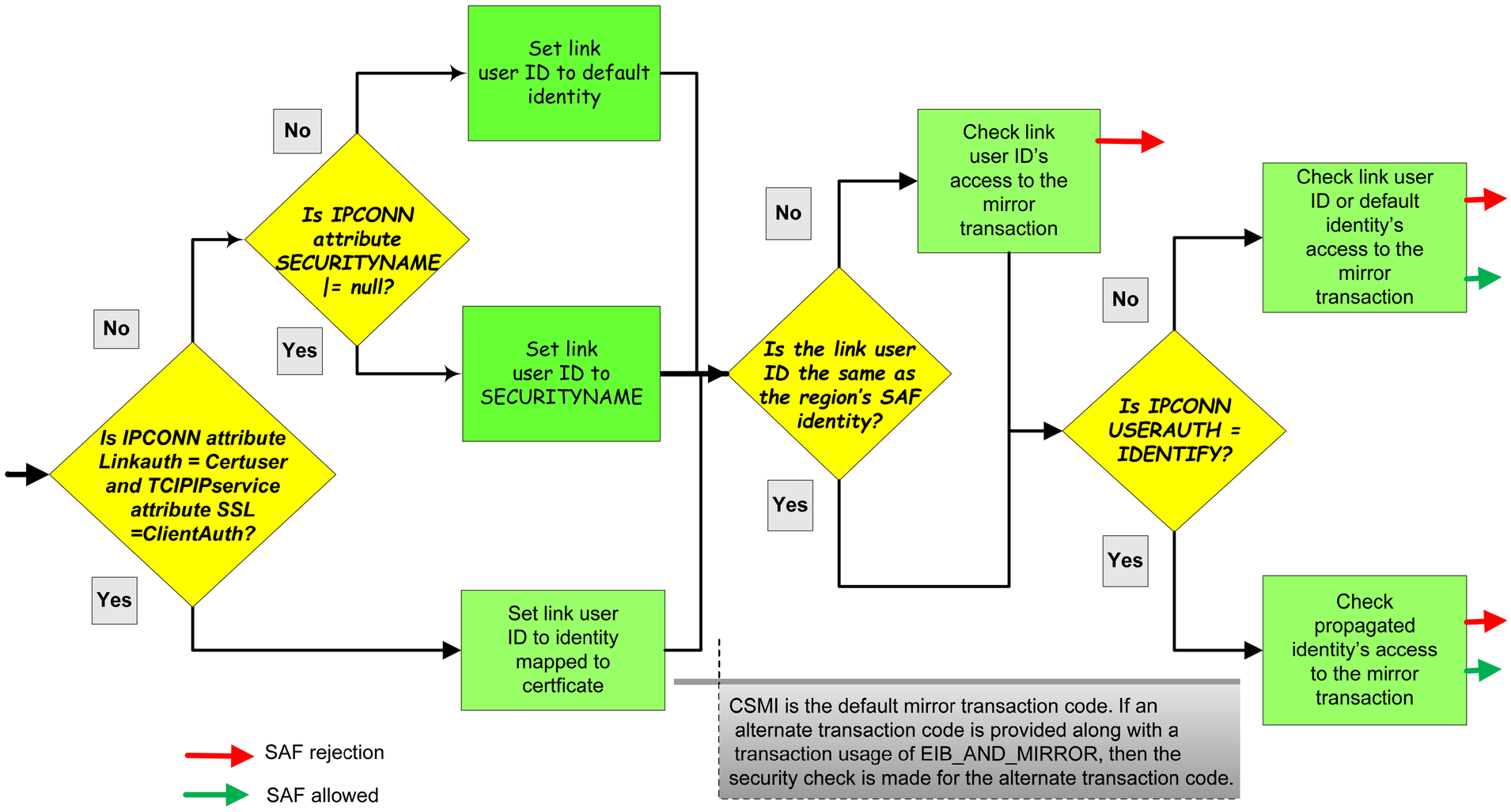



CICS IPCONN



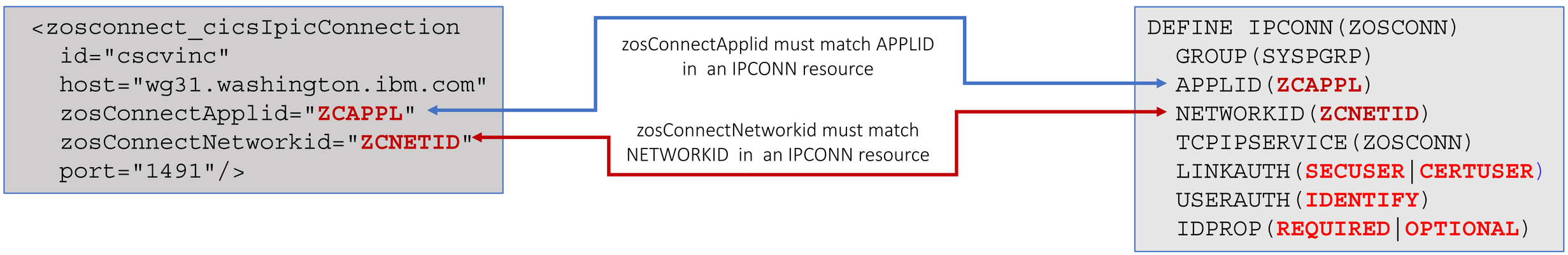

mitchj@us.ibm.com

# Tech/Tip: CICS IPIC Security with USERAUTH(IDENTIFY)




mitchj@us.ibm.com

# CICS IPCONN Resources




**LINKAUTH** Determines the user identity to be used for link security. The value is either ***CERTUSER*** or ***SECUSER***. A value of ***CERTUSER*** sets the link identity to the identity associated with the client certificate received from the client endpoint (TLS mutual authentication is required). A value of ***SECUSER*** sets the link identity to the value of the *SECURITYNAME* attribute as defined in the IPCONN resource.

**USERAUTH** Identifies how the identity under which the attached transaction attach security will run. Since a password is not available, a value of ***VERIFY*** is not possible. A value of ***LOCAL*** means the current link identity is used. A value of ***DEFAULTUSER*** means the CICS default identity is used. For identity propagation purposes, the value of **USERAUTH** should be ***IDENTIFY*** (no password will be required) so the identity provided by the client is used for executing the attached transaction. TLS must be used if the client is in a different Sysplex.

**IDPROP** Determines whether the original distributed identity authenticated by the z/OS Connect server is also propagated to CICS in addition to the mapped identity used for z/OS Connect authorization checks. A value of ***NOTALLOWED*** does not propagate the original distributed identity. A value of ***OPTIONAL*** will propagate to CICS the original distributed identity, if available. A value of ***REQUIRED*** requires that the original distributed identity be propagated to CICS. TLS must be used if the client is in a different Sysplex.

**CERTIFICATE** Provides the label of the certificate connected to the CICS key ring to be used for server endpoint certificate during a TLS handshake.

mitchj@us.ibm.com

# Identity Propagation and CICS High Availability Considerations

Assume the service installed in a server files use the following *Connection reference* values:

- cscvinc
- catalog
- miniloan

If identity propagation is required for all connection, then the CICS IPCONN resources defined in the CICs that correspond to a zosconnect_cicsIpicConnection configuration elements must be dedicated to that z/OS Connect server and connection reference can not be reused.

Simplify administration by still sharing a common cicsIpicConnection XML configuration element by using variables and a bootstrap properties file or "variables" XML file

Server baqsvr1's bootstrap.properties

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

Server baqsvr2's bootstrap.properties

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

Server baqsvr3's bootstrap.properties

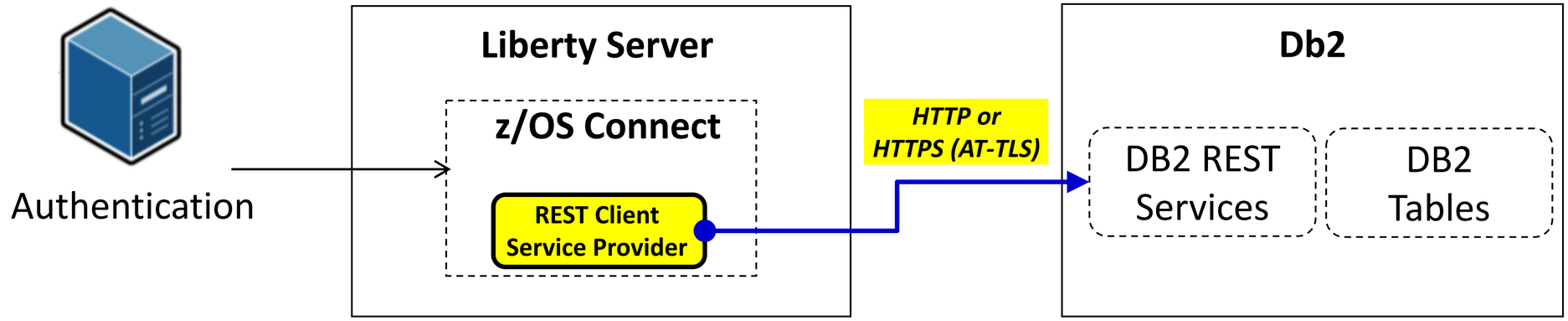
```
cicsHost=dvipa.washington.ibm.com
ipicPort=1491
serverPrefix=baqsvr3
```

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

# Flowing the identity for the REST client (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**

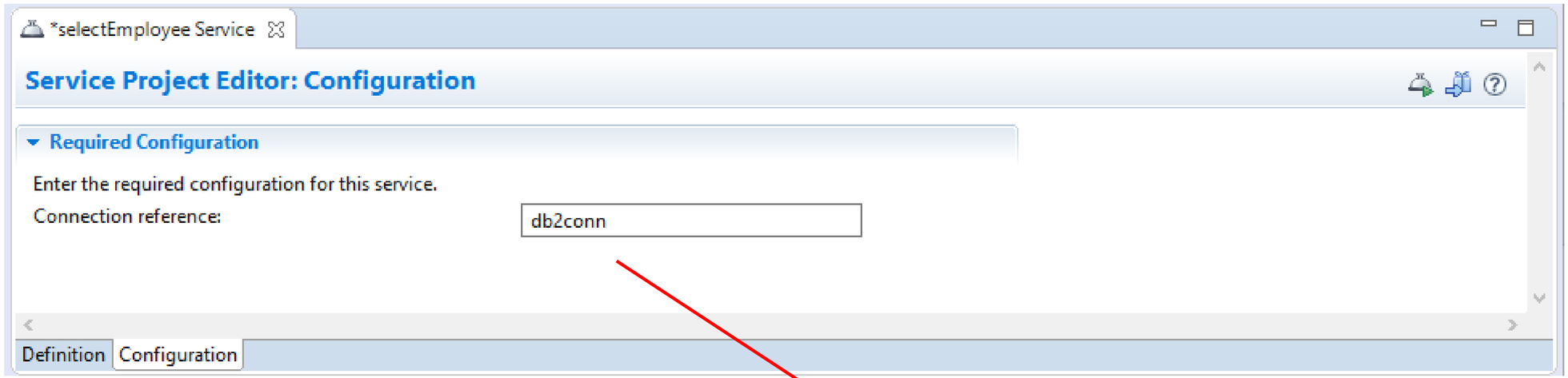
Specify a user identity and password to be used in the HTTP header with the Db2 REST Service

```
<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

# Server XML - Accessing a Db2 REST service




```
DSNL004I  -DSN2 DDF START
COMPLETE
          LOCATION  DSN2LOC
          LU
USIBMWZ.DSN2APPL
          GENERICLU -NONE
          DOMAIN
WG31.WASHINGTON.IBM.COM
          TCPPORT   2446
          SECPORT   2445
          RESPORT   2447
```

```
<zosconnect_zosConnectServiceRestClientConnection id="db2conn"
      host="wg31.washington.ibm.com"
      port="2446"
      basicAuthRef="dsn2Auth" />

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

```
<featureManager>
  <feature>zosconnect:db2-1.0</feature>
</featureManager>

<zosconnect_credential user="${DB2_USERNAME}"
       password="${DB2_PASSWORD}" id="commonCredentials" />

<zosconnect_db2Connection id="db2Conn" host="${DB2_HOST}"
      port="${DB2_PORT}" credentialRef="commonCredentials" />
```

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
```

```
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)
```

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***

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.

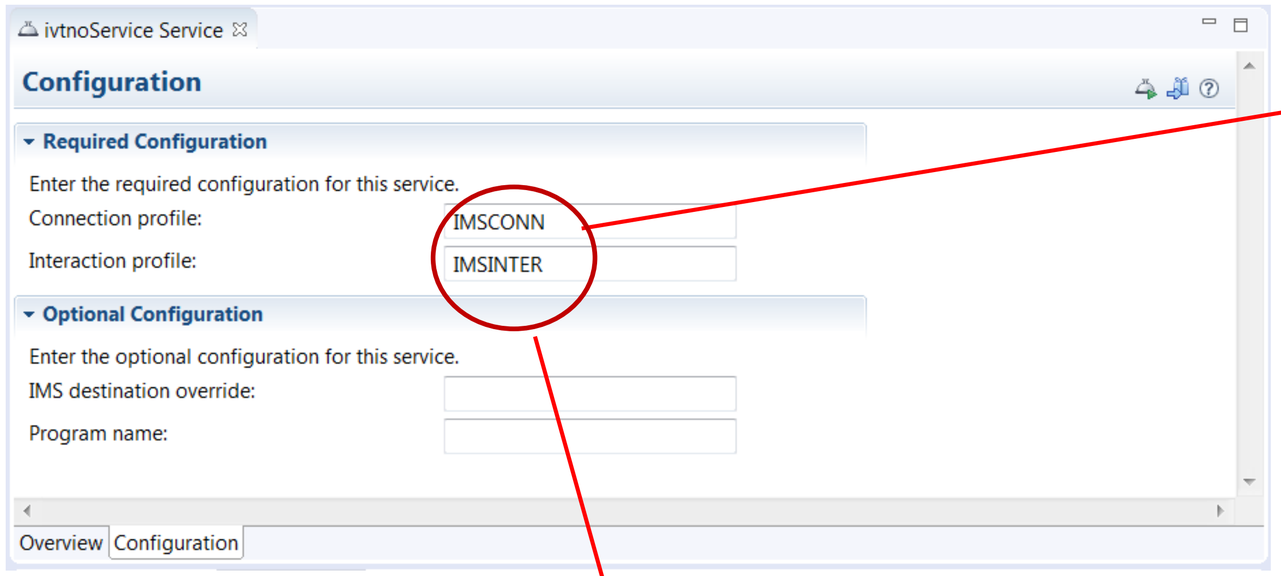
  For example, if service *zCEEService.selectEmployee* is defined to Db2 but not visible in the z/OS Connect tooling or if a **GET** request to URL http://wg31.washington.ibm.com:2446/services/zCEEService/selectEmployee fails with message:

```
{
  "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**

# IMS TM Connections and Interactions




Connection Profile

```
<server>
<imsmobile_imsConnection comment="" connectionFactoryRef="CF1"
connectionTimeout="-1" connectionType="IMSCONNECT" id="IMSCONN"/>
<connectionFactory containerAuthDataRef="Connection1_Auth" id="CF1">
            <properties.gmoa hostName="wg31.washington.ibm.com"
portNumber="4000"/>
</connectionFactory>

<authData id="Connection1_Auth" password="encryptedPassword1"
user="userName1"/>
</server>
```

Interaction Profile

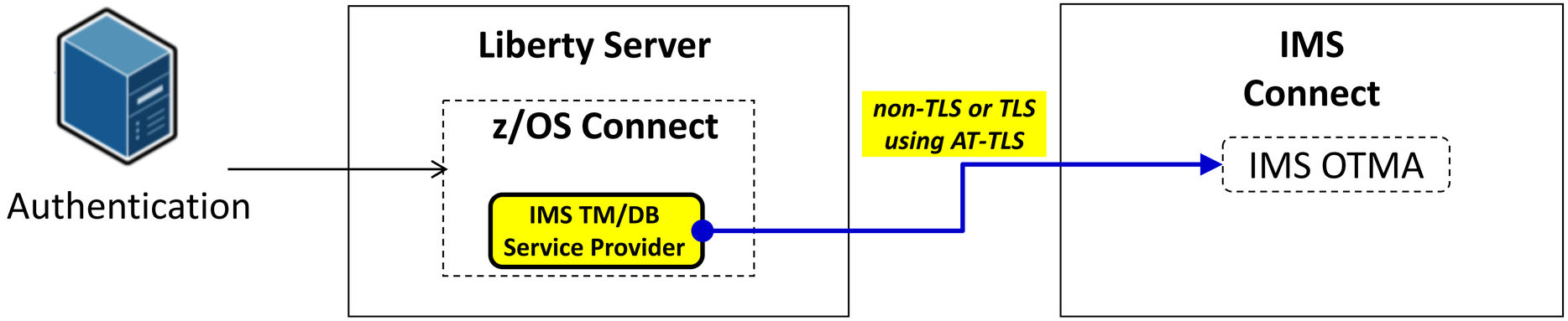
```
<server>
<imsmobile_interaction comment="" commitMode="1" id="IMSINTER"
imsConnectCodepage="Cp1047" imsConnectTimeout="0"
            imsDatastoreName="IVP1" interactionTimeout="-1"
ltermOverrideName="" syncLevel="0"/>
</server>
```

IMS Connect HWSCFG

```
HWS=(ID=IMS14HWS,XIBAREA=100,RACF=Y,RRS=N)
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,
TIMEOUT=5000)
DATASTORE=(GROUP=OTMAGRP,ID=IVP1,MEMBER=HWSMEM,TMEMBER=OT
MAMEM)
```

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

# PassTickets and IMS

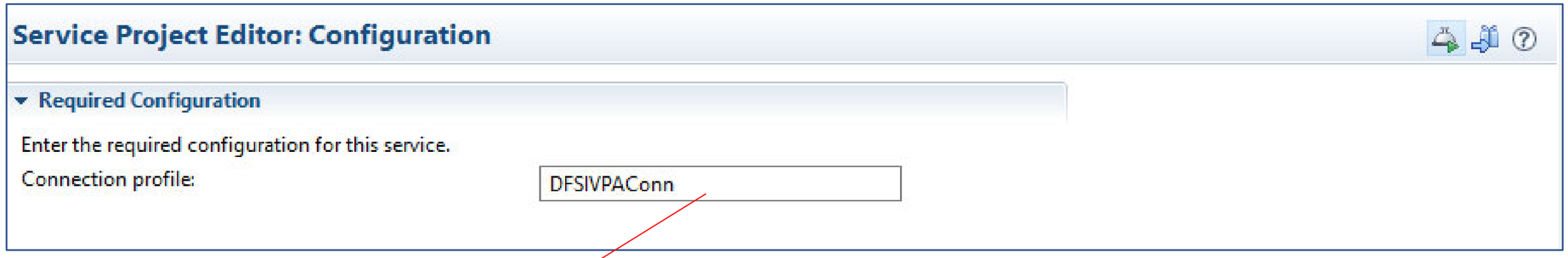
- Basic authentication to IMS Connect using a PassTicket depends on the APPL parameters configured in IMS Connect.

```
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)**

# IMS DB Connection Factory in the server XML




ConnectionFactory

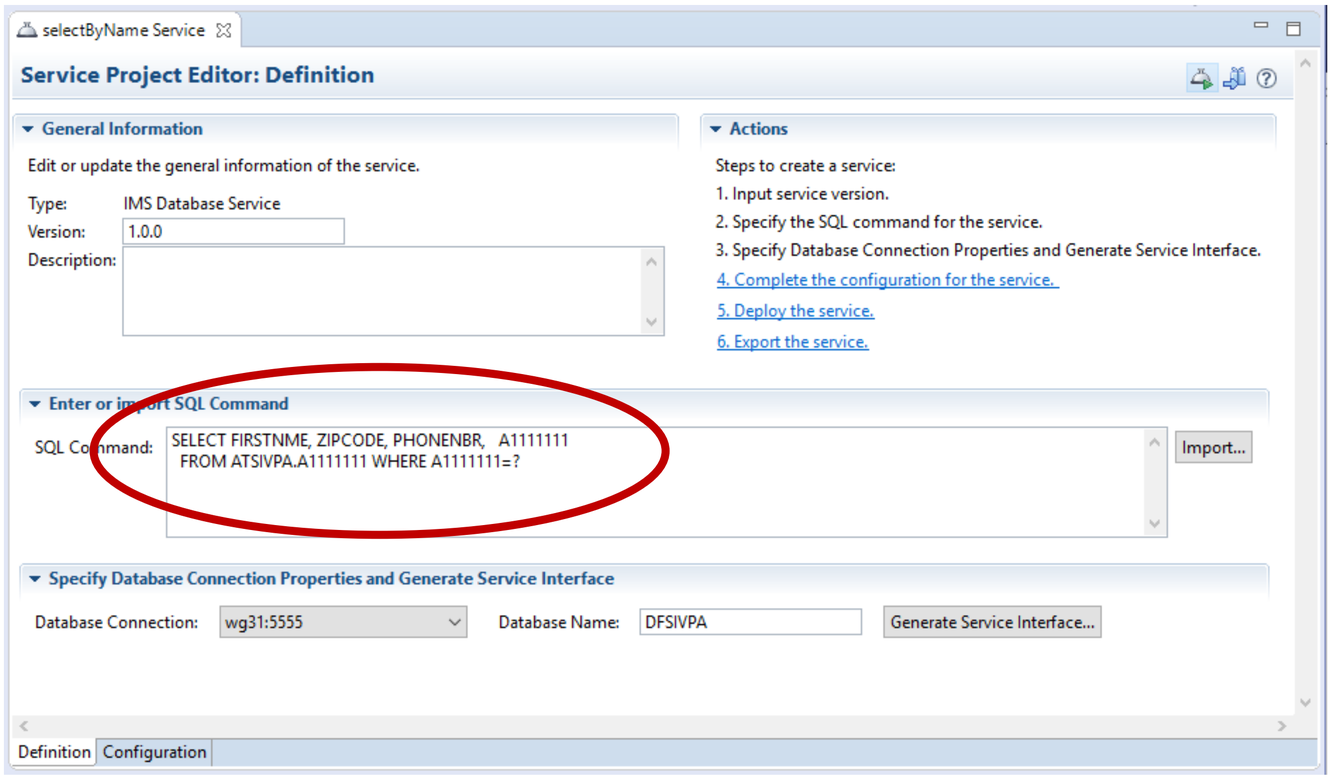
```
<connectionFactory id="DFSIVPAConn">
<properties.imsudbJLocal
   databaseName="DFSIVPA"
   datastoreName="IVP1"
   datastoreServer="wg31.washington.ibm.com"
   driverType="4"
   portNumber="5555"
   user="USER1"
   password="password"
   flattenTables="True"/>
</connectionFactory>
```

IMS Connect HWSCFG

```
HWS=(ID=IMS14HWS,XIBAREA=100,RACF=N,RRS=N)
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,TIMEOUT=5000)
DATASTORE=(GROUP=OTMAGRP,ID=IVP1,MEMBER=HWSMEM,TMEMBER=OTMAMEM)
IMSPLEX=(MEMBER=IMS14HWS,TMEMBER=PLEX1)
ODACCESS=(ODBMAUTOCONN=Y,
DRDAPORT=(ID=5555,PORTTMOT=6000),ODBMTMOT=6000)
```

# API toolkit – Creating Services for IMS DB

## Creating a service project from the IMS Catalog

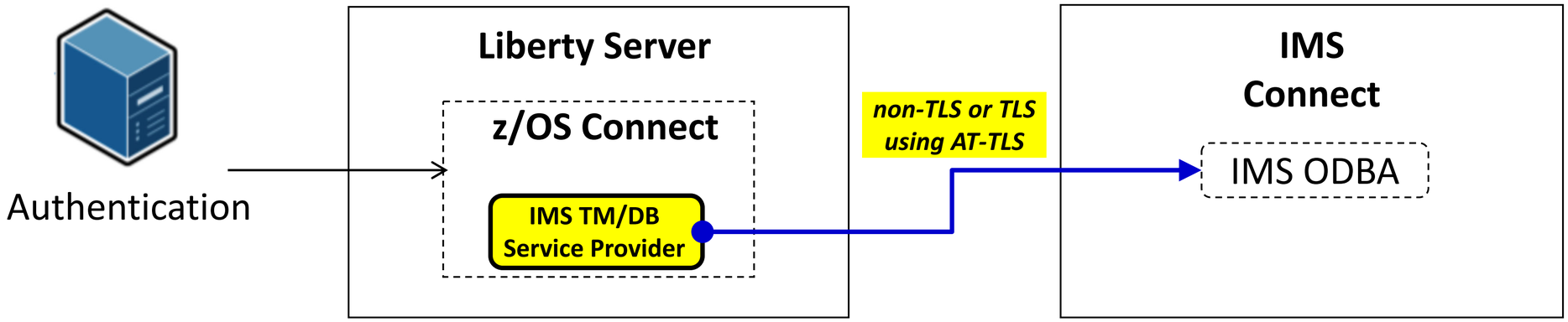



Use the IMS Catalog to assist with developing and testing SQL SELECT commands used for accessing IMS databases.

```
*-------------------------------------------------------------------*
*  SEGMENT DESCRIPTION                                              *
*    ROOT ONLY DATABASE                                             *
*      BYTES  1-10  LAST NAME (CHARACTER) - KEY                     *
*      BYTES  11-20 FIRST NAME (CHARACTER)                          *
*      BYTES  21-30 INTERNAL PHONE NUMBER (NUMERIC)                 *
*      BYTES  31-37 INTERNAL ZIP (CHARACTER)                        *
*      BYTES  38-40 RESERVED                                        *
*                                                                   *
*-------------------------------------------------------------------*
         DBD    NAME=IVPDB1,ACCESS=(HIDAM,OSAM)
         DATASET DD1=DFSIVD1,DEVICE=3380,SIZE=2048
         SEGM   NAME=A1111111,PARENT=0,BYTES=40,RULES=(LLV,LAST),      x
               PTR=(TB,CTR)
         FIELD  NAME=(A1111111,SEQ,U),BYTES=010,START=00001,TYPE=C
         FIELD  NAME=FIRSTNME,BYTES=010,START=00011,TYPE=C
         FIELD  NAME=PHONENBR,BYTES=010,START=00021,TYPE=C
         FIELD  NAME=ZIPCODE,BYTES=7,START=00031,TYPE=C
         LCHILD NAME=(A1,IVPDB1I),POINTER=INDX,RULES=LAST
         DBDGEN
         FINISH
         END
```

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>
```

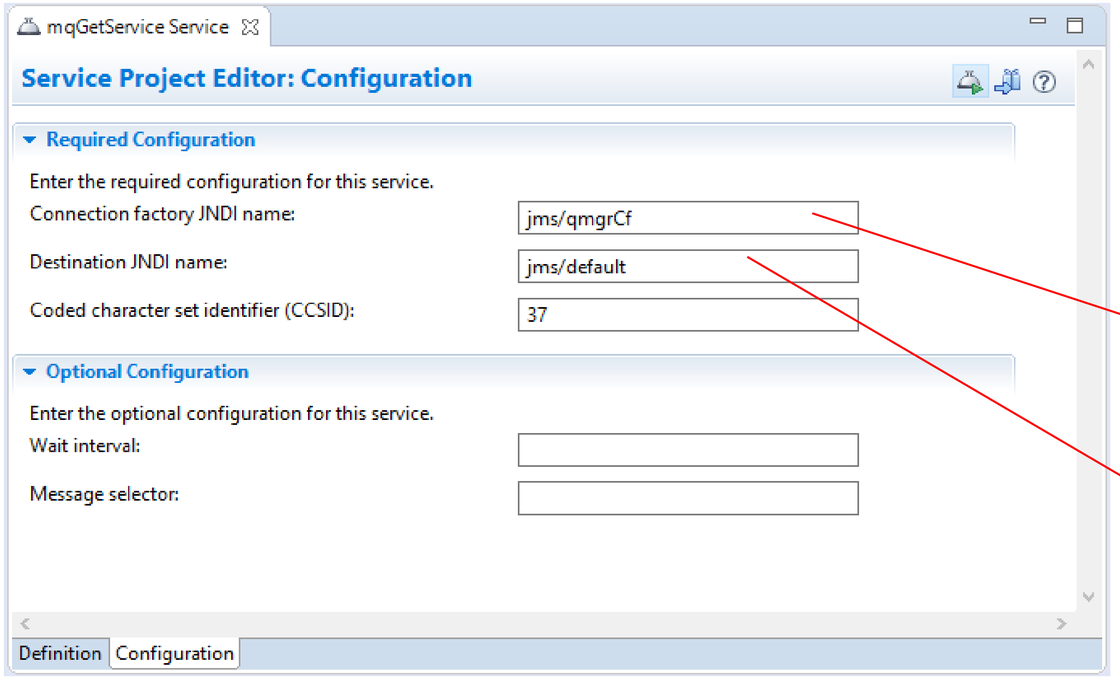
Specify a user identity and password to be used in the request to IMS Connect

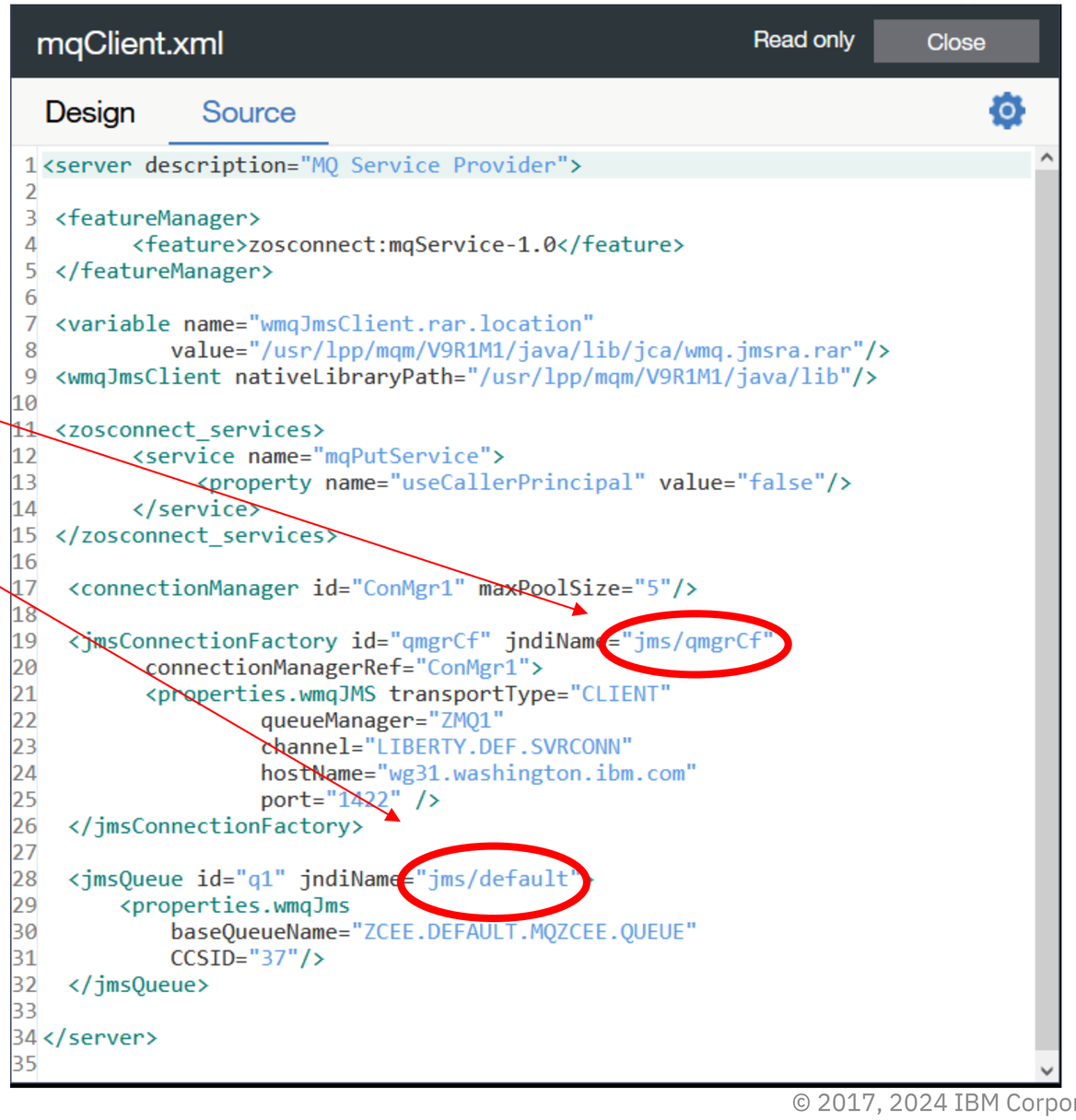
```
<connectionFactory id="DFSIVPAConn">    <properties.imsudbJLocal
databaseName="DFSIVPA" datastoreName="IVP1" portNumber="5555"
datastoreServer="wg31.washington.ibm.com«  driverType="4" flattenTables="True"
applicationName="IMSDBAPL"  "/>
</connectionFactory>
```

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

mitchj@us.ibm.com

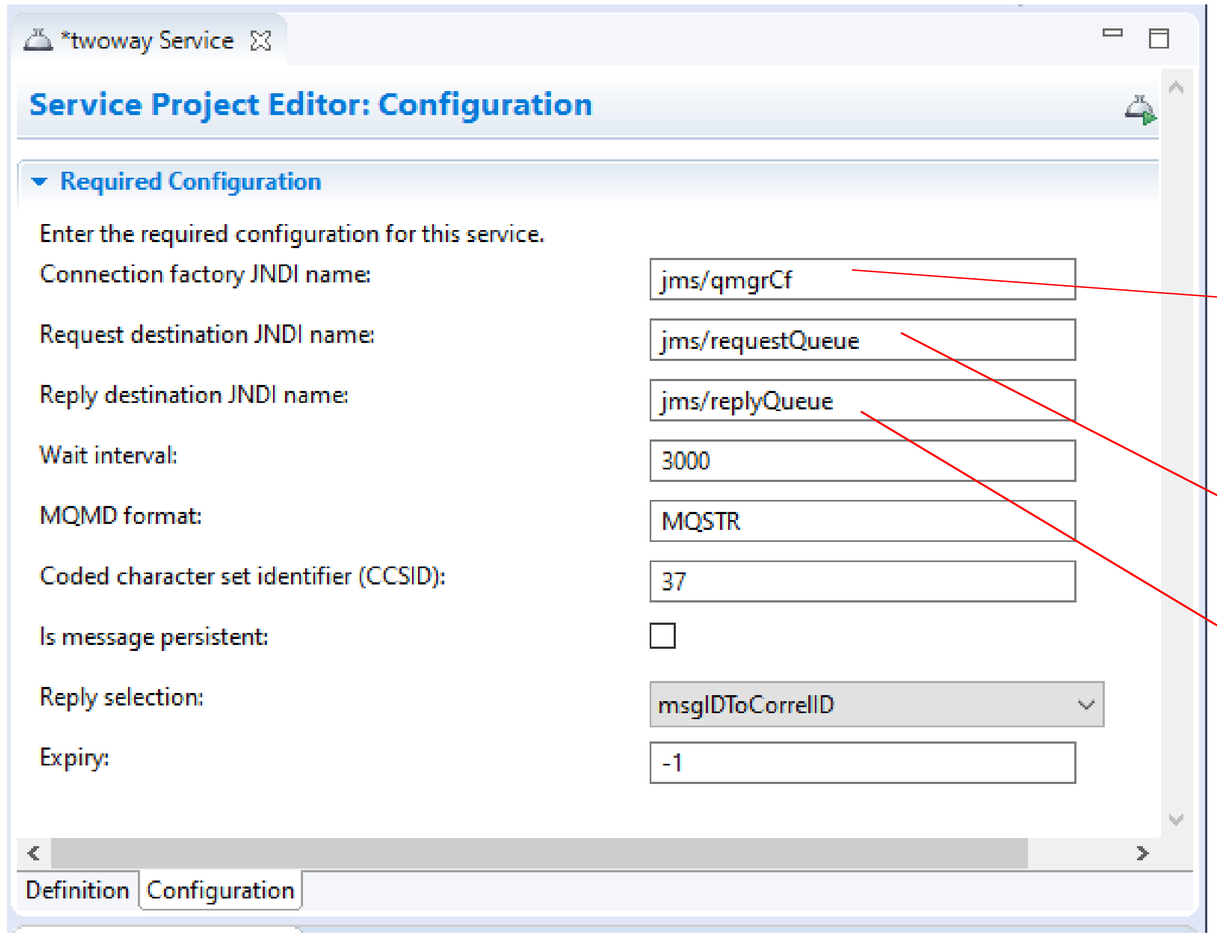
# Using JMS to access MQ (One-Way)

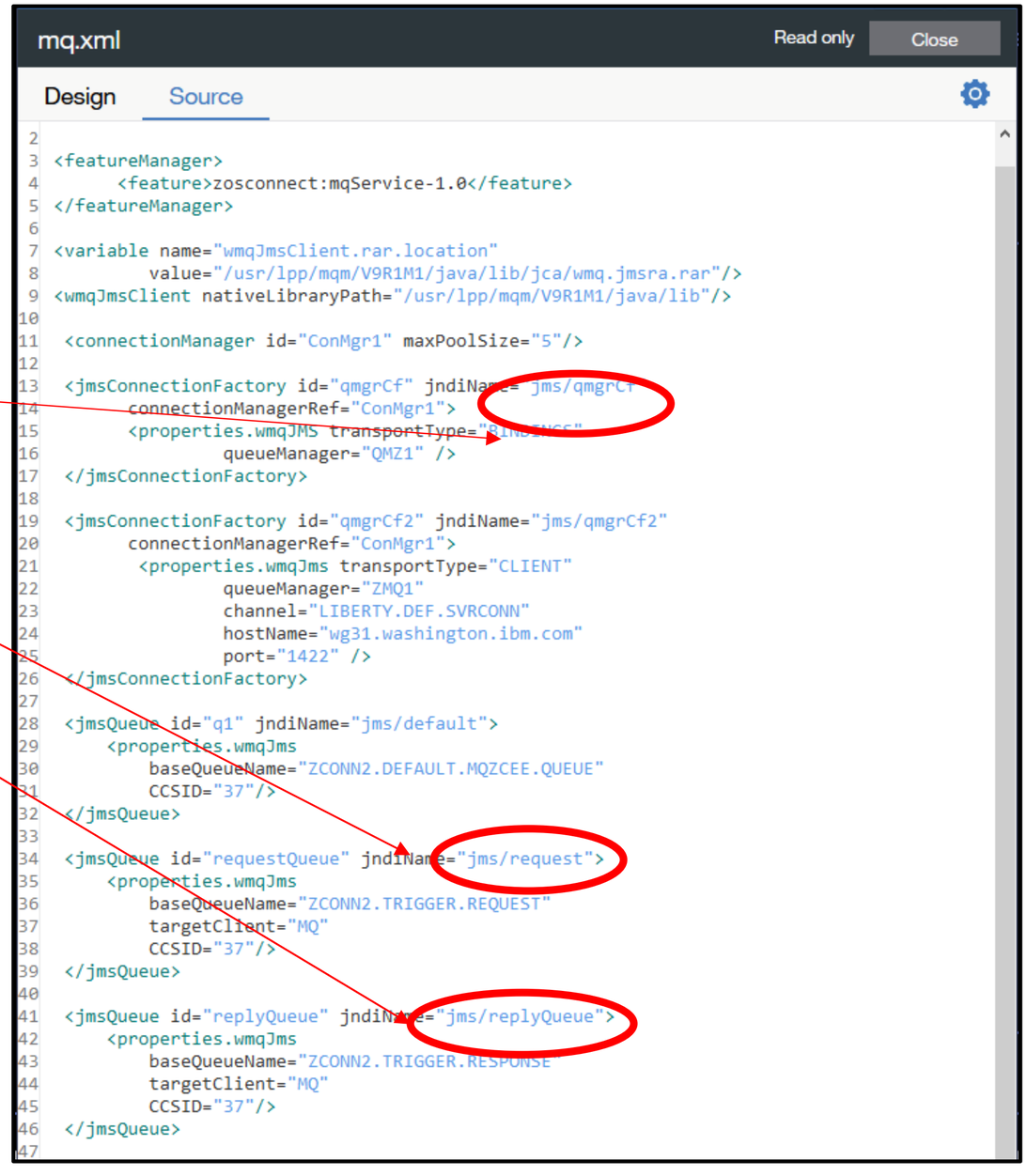



mitchj@us.ibm.com

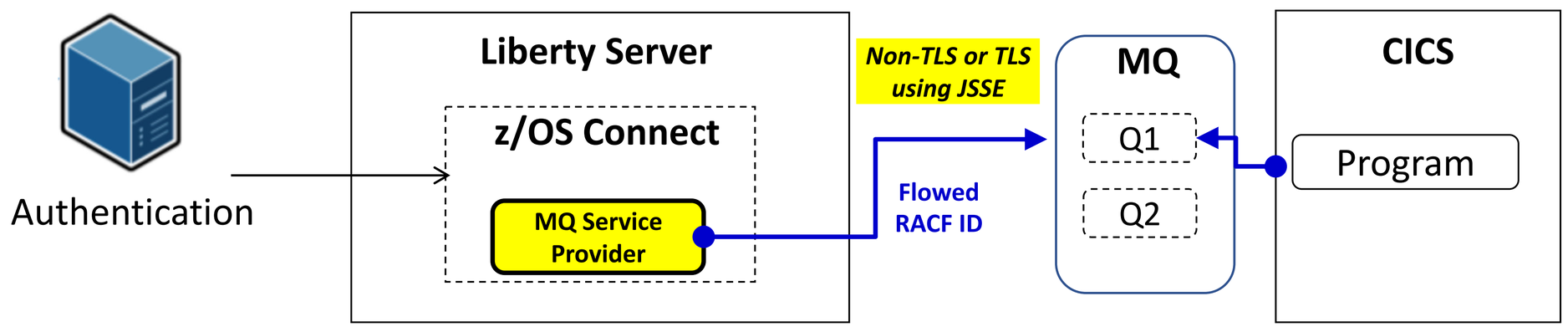
# Using JMS to access MQ (Two-Way)




mitchj@us.ibm.com

# Flowing a user ID with MQ service provider




Set **`useCallerPrincipal=true`** to flow the authenticated RACF user ID, note that this is set at the service.

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

Define identity propagation to MQ

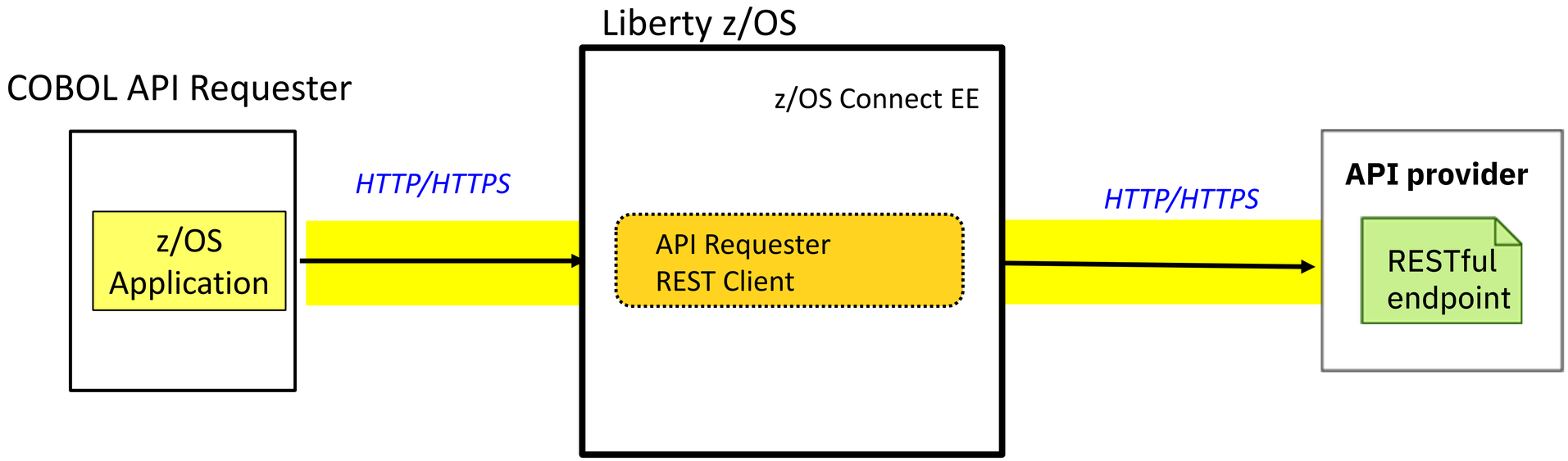
mitchj@us.ibm.com

# Security when accessing non-z/OS systems

# z/OS Connect API Requester Security

mitchj@us.ibm.com

# End to end API requester to API Provider connection overview




MVS Batch, IMS HTTP and Db2 stored procedure connection details provided by:

- Environment Variables (BAQURI, BAQPORT)
  - Via JCL
  - LE Options (CEEROPTS)
  - Programmatically (CEEENV)
- HTTP or HTTPS

CICS HTTP connection details provided by:

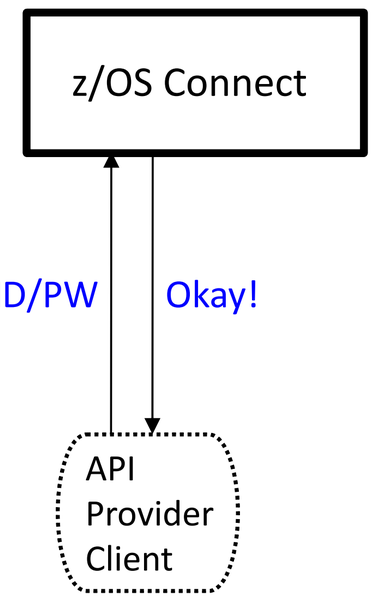
- CICS URIMAP resource (default BAQURIMP)
  - HOST
  - PORT
  - SCHEME (HTTP/HTTPS)

mitchj@us.ibm.com

# **API Requester** – Security from the z/OS application to a Liberty server

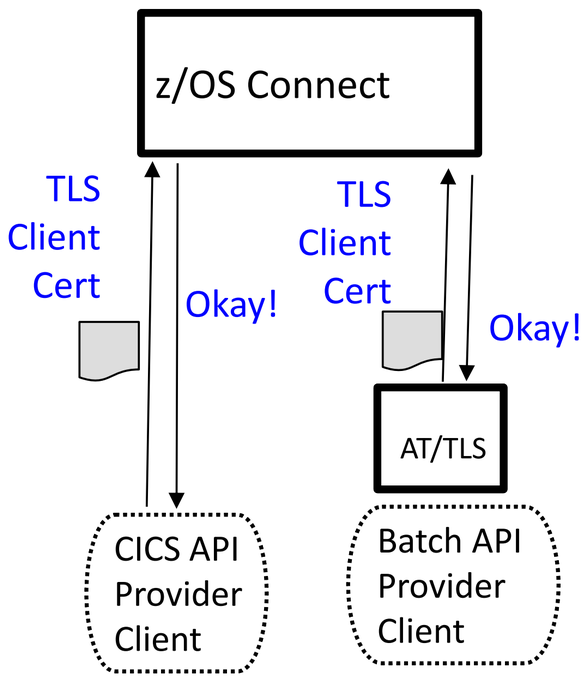
Two options for providing credentials for authentication

Basic Authentication




**Application provides ID/PW or ID/PassTicket**

Client Certificate



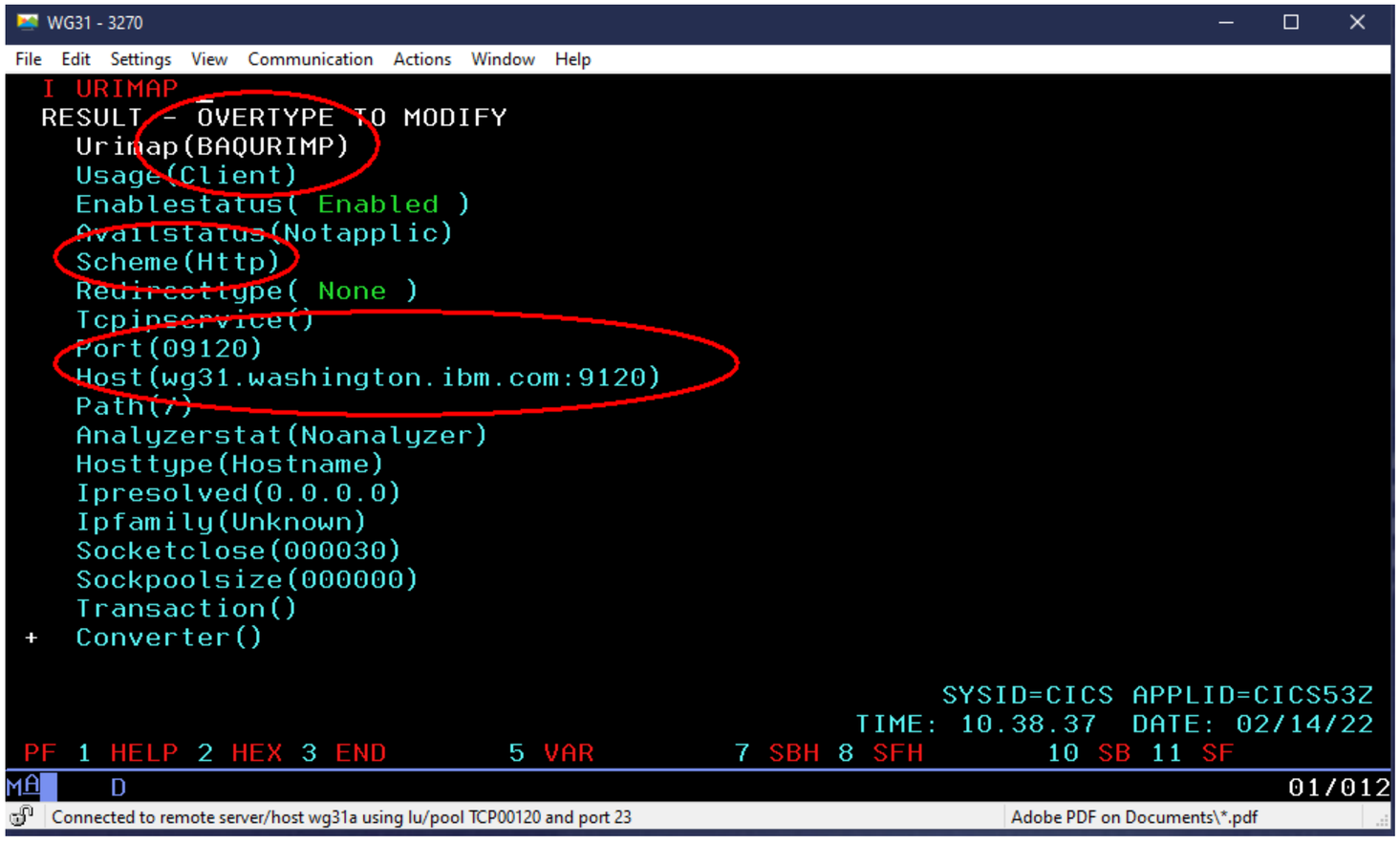

**z/OS Connect requests a client certificate**

**CICS or AT/TLS supplies a client certificate**

mitchj@us.ibm.com

# Configuring connections to the z/OS API requester server

## Default CICS URI MAP*




## LE Environment Variables

```
//DELTAPI EXEC PGM=DELTAPI,PARM='323232'
//STEPLIB  DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB
//         DD DISP=SHR,DSN=ZCEE30.SBAQLIB
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEOPTS  DD *
 POSIX(ON),
 ENVAR("BAQURI=wg31.washington.ibm.com",
 "BAQPORT=9120")
```

* V3.0.37 added support for a CICS application to specify or request a specific URIMAP resource the using BAQ-ZCON-SERVER-URI variable in BAQRINFO

mitchj@us.ibm.com

# A COBOL API Requester using basic authentication

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

- The environment 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 that the z/OS Connect communications stub generates the Authentication header token we saw earlier**

- 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

# Environment variables for non-CICS clients

Use these runtime environment variables when connecting to a z/OS Connect server

**BAQPASSWORD** - Specifies the password, in clear text, for the specified BAQUSERNAME to be authenticated with the z/OS Connect server. The username and password that are used for basic authentication, when SSL mutual authentication is not enabled.

**BAQPORT** - Specifies the port number for the z/OS Connect server.

**BAQTIMEOUT** - An optional 4-byte integer to set a timeout value in seconds for waiting for an API response. Valid range is 1 - 2,678,400 seconds. The default timeout value is 10 seconds.

**BAQURI** - Specifies either an IPv4 or IPV6 address, or a hostname of the host where the z/OS Connect server resides.

**BAQUSERNAME** - Specifies the username for connections if basic authentication is used.

**BAQVERBOSE -** An optional value to turn on verbose messages to assist debugging of runtime and configuration issues. Valid values are **OFF**, **ON**, **ERROR**, **AUDIT** and **ALL**. See URL https://www.ibm.com/docs/en/zos-connect/zosconnect/3.0?topic=car-configuring-other-zos-applications-access-zos-connect-api-calls for more information.

mitchj@us.ibm.com

# Tech/Tip: Generating PassTickets on z/OS

- On z/OS, a COBOL user application can generate a pass tickets by calling RACF service IRRSPK00:

```
 77 COMM-STUB-PGM-NAME          PIC X(8) VALUE 'BAQCSTUB'.
 77 PTKT-STUB-PGM-NAME          PIC X(8) VALUE 'ATSPTKTC'.

*-----------------------------------------------------------------*

*****************************************************************
*    L I N K A G E   S E C T I O N
*****************************************************************
 LINKAGE SECTION.
*****************************************************************
*    P R O C E D U R E S
*****************************************************************
 PROCEDURE DIVISION using PARM-BUFFER.

*-----------------------------------------------------------------*
 MAINLINE SECTION.

*-----------------------------------------------------------------*
* Common code                                                     *
*-----------------------------------------------------------------*
* initialize working storage variables
     INITIALIZE GET-REQUEST.
     INITIALIZE GET-RESPONSE.
     CALL PTKT-STUB-PGM-NAME.
```

**JOHNSON.PASSTCKT.SOURCE(ATSPTKTC)**

```
      *-----------------------------------------------------------------*
      * Build IRRSPK00 parameters                                       *
      *-----------------------------------------------------------------*
           MOVE 0 to ws-length
           MOVE LENGTH OF identity to identity-length.
           INSPECT FUNCTION REVERSE (identity)
                  TALLYING ws-length FOR ALL SPACES.
           SUBTRACT ws-length FROM identity-length.
           MOVE 0 to ws-length
           MOVE LENGTH OF applid to applid-length.
           INSPECT FUNCTION REVERSE (applid)
                  TALLYING ws-length FOR ALL SPACES.
           SUBTRACT ws-length FROM applid-length.
           MOVE 8 to passTicket-length.
           MOVE 'NOTICKET' to passTicket.
           MOVE X'0003' to irr-functionCode.
           MOVE X'00000001' to irr-ticketOptions.
           SET irr-ticketOptions-ptr to ADDRESS OF irr-ticketOptions.
      *-----------------------------------------------------------------*
      * Call RACF service IRRSPK00 to obtain a pass ticket based        *
      *      on identity and applid                                     *
      *-----------------------------------------------------------------*
           PERFORM CALL-RACF.
           IF irr-safrc NOT = zero then
              DISPLAY "SAF_return_code:      " irr-safrc
              DISPLAY "RACF_return_code:     " irr-racfrc
              DISPLAY "RACF_reason_code:     " irr-racfrsn
           End-if
. . . .
      *-----------------------------------------------------------------*
      * Call IRRSPK00 requesting a pass ticket                          *
      *-----------------------------------------------------------------*
       CALL-RACF.
            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

See the full source of the program at URL https://ibm.biz/BdaLJX

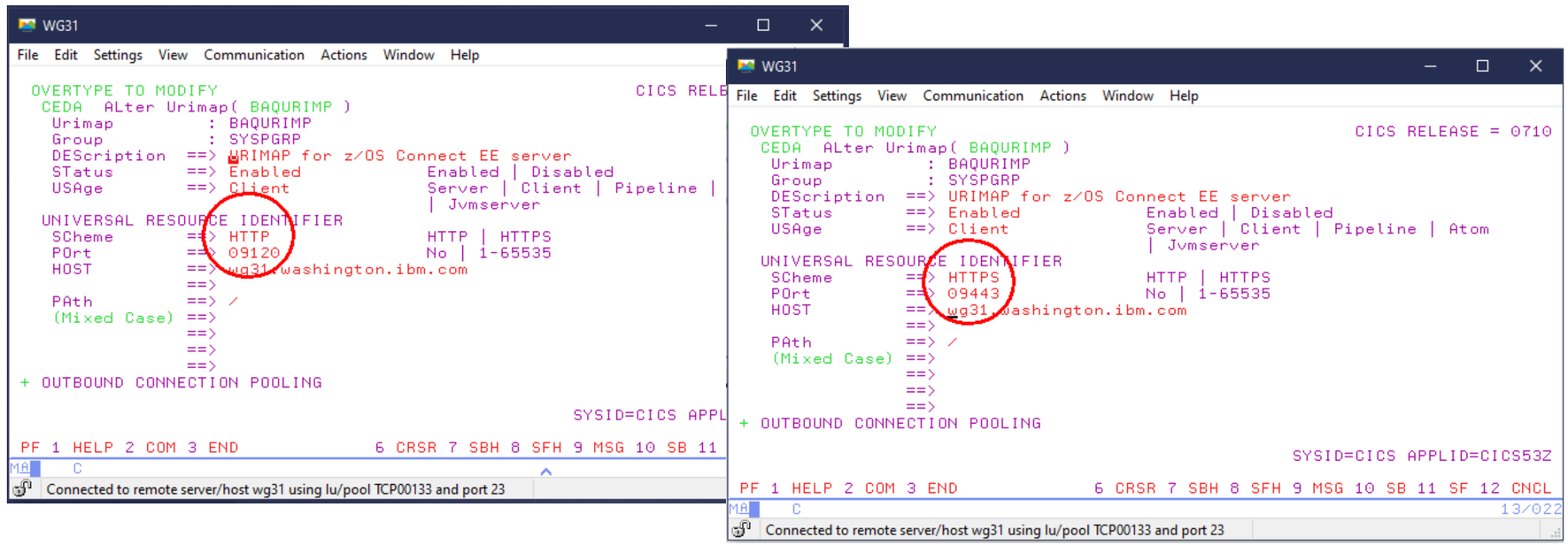
# Tech/Tip:  API Requester - HTTP v HTTPS

MVS Batch and IMS with and without an outbound AT-TLS policy

```
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



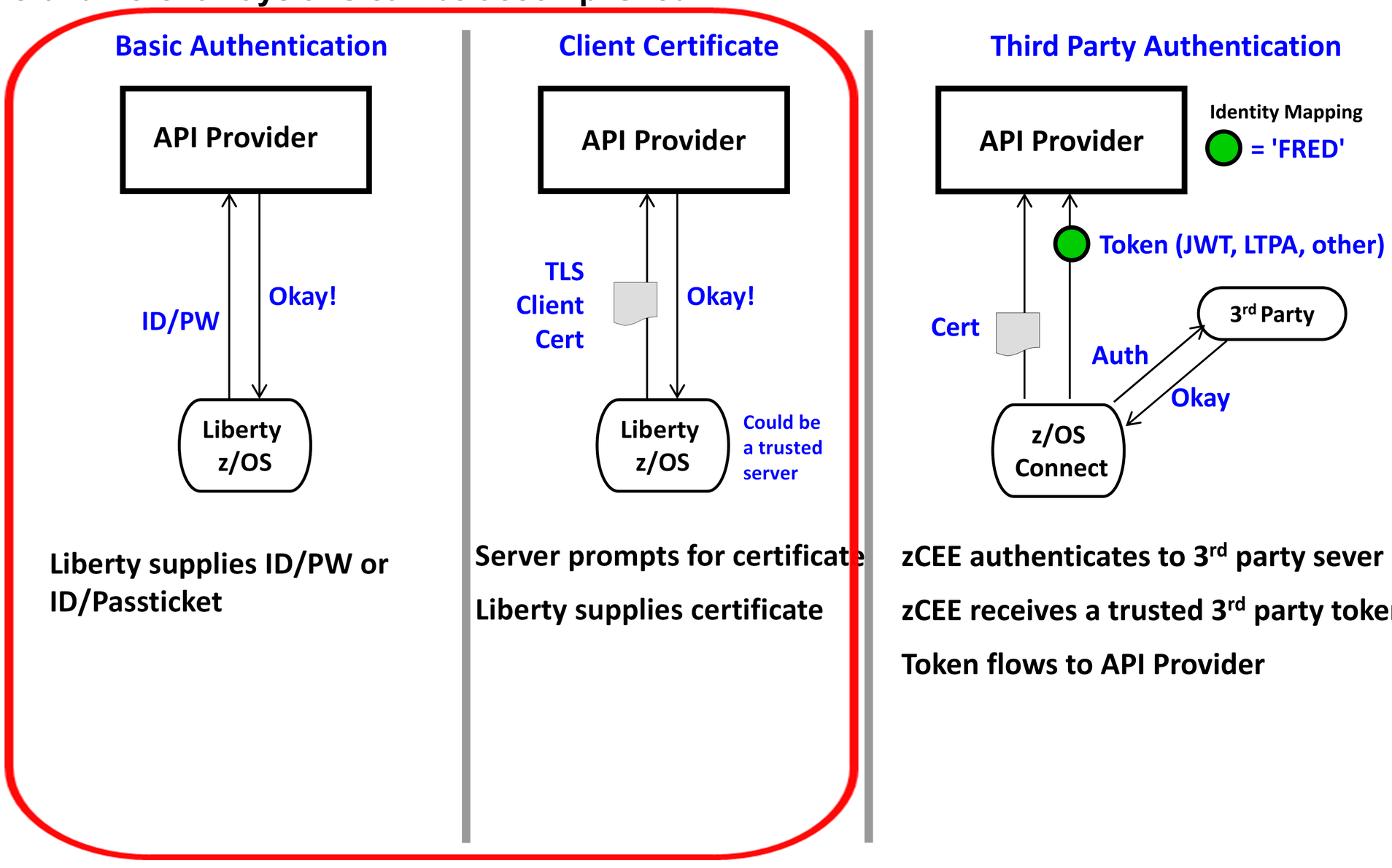

Field BAQ-ZCON-SERVER-URI was added to BAQRINFO in V3.0.37.

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

[1] mitchj@us.ibm.com

# Accessing non-z/OS resources

**Several different ways this can be accomplished:**




mitchj@us.ibm.com

# Configuring Basic and/or TSL support

Basic authentication with HTTP protocol

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

<zosconnect_authData id="myAuthData"
  user="zCEEClient" password="secret"/>
```

TLS with HTTPS protocol

```
<zosconnect_endpointConnection id="cscvincAPI"
       host="https://wg31.washington.ibm.com" port="9443"
       authenticationConfigRef="myAuthData"  1
       sslCertsRef="OutboundSSLSettings" />

<zosconnect_authData id="myAuthData"   1
  user="zCEEClient" password="secret"/>
```

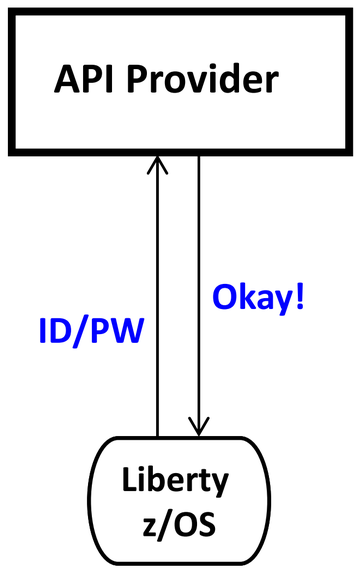
[1] Optional, if mutual authentication is enabled by the server endpoint

mitchj@us.ibm.com

# **API Requester** – Security from the z/OS Connect server to the API provider

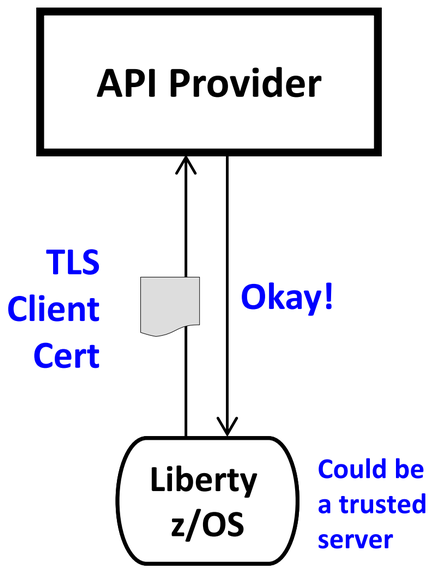
**Several different ways this can be accomplished:**

## Basic Authentication




zCEE supplies ID/PW or ID/Passticket

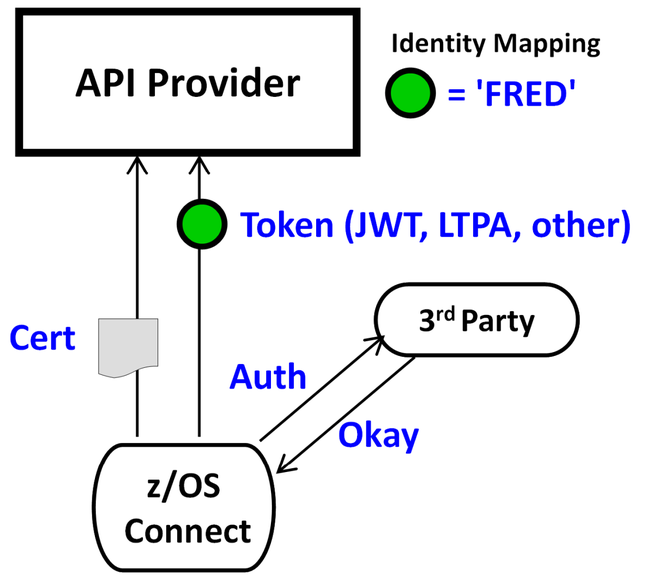
## Client Certificate




Server prompts for certificate

zCEE supplies certificate

## Third Party Authentication




zCEE authenticates to 3rd party sever

zCEE 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

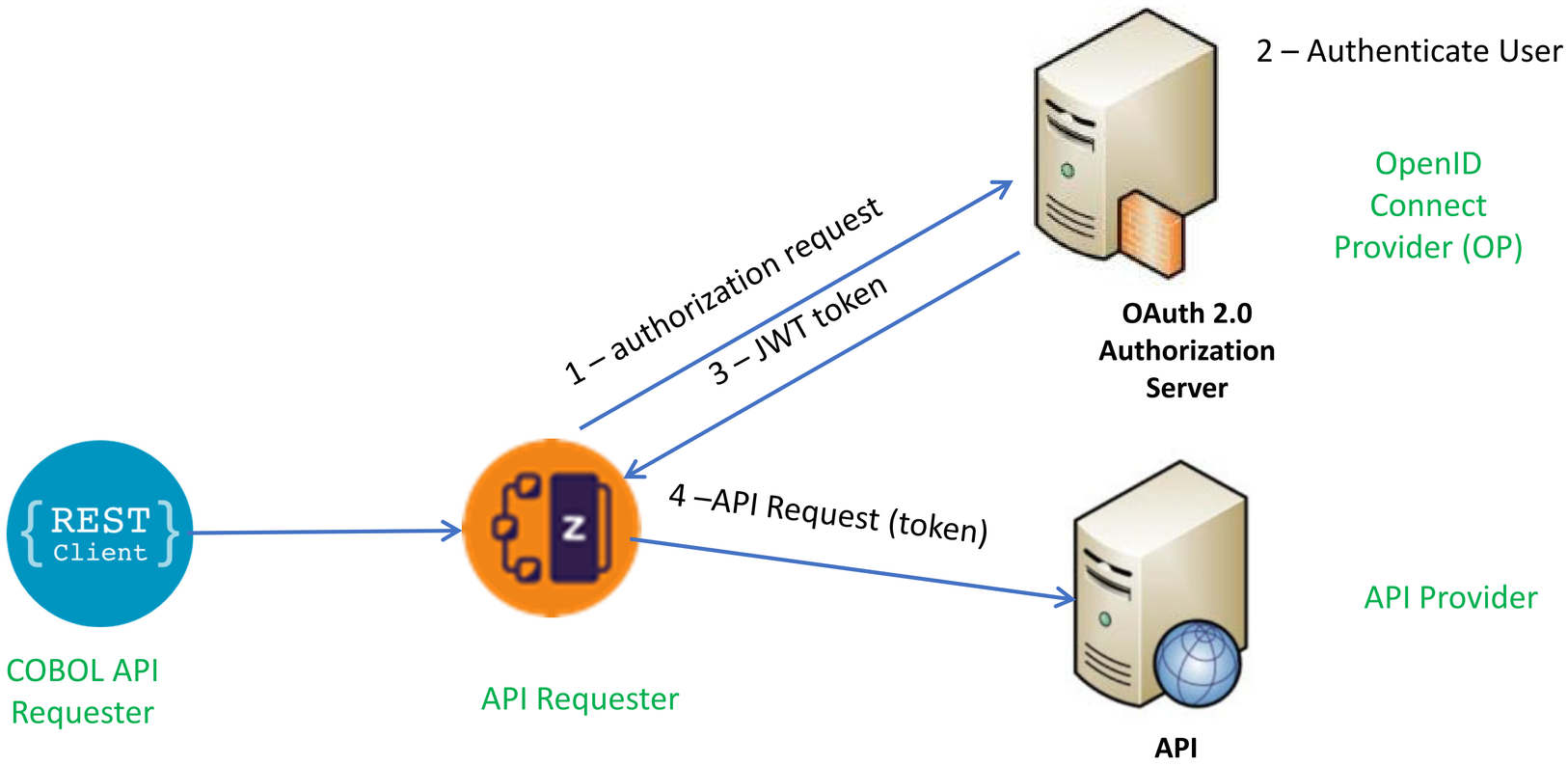
1. Use the OAuth 2.0 support when the request is part of an OAuth 2.0 flow. With OAUTH configured, the token can be an opaque token or a JWT token.

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

# 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.*

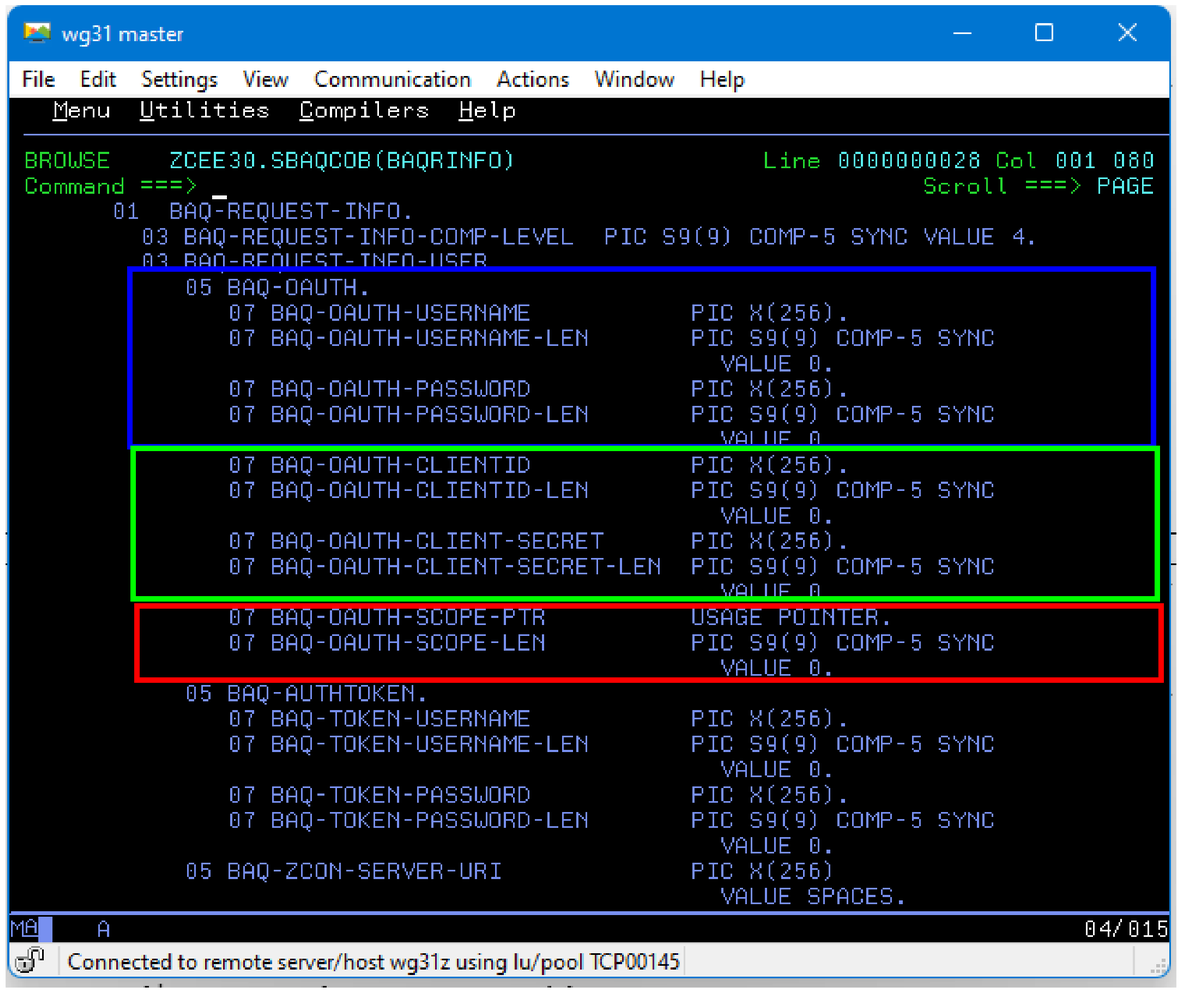
```
<zosconnect_oAuthConfig id="myoAuthConfig"
      grantType="client_credentials"
      authServerRef="myoAuthServer"/>
```

***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




**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.**

**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**

**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

# Obtaining a JWT using request parameters

wg31 master

File Edit Settings View Communication Actions Window Help

```
 Menu  Utilities  Compilers  Help
BROWSE    ZCEE30.SBAQCOB(BAQHCONC)                Line 0000000020 Col 001 080
Command ===>                                        Scroll ===> PAGE
      * Host API Request parameter names
       77 BAQR-OAUTH-USERNAME       PIC X(22)
           VALUE 'BAQHAPI-oAuth-Username'.
       77 BAQR-OAUTH-PASSWORD       PIC X(22)
           VALUE 'BAQHAPI-oAuth-Password'.
       77 BAQR-OAUTH-SCOPE          PIC X(19)
           VALUE 'BAQHAPI-oAuth-Scope'.
       77 BAQR-OAUTH-CLIENT-ID      PIC X(22)
           VALUE 'BAQHAPI-oAuth-ClientId'.
       77 BAQR-OAUTH-CLIENT-SECRET PIC X(26)
           VALUE 'BAQHAPI-oAuth-ClientSecret'.
       77 BAQR-OAUTH-RESOURCE       PIC X(22)
           VALUE 'BAQHAPI-oAuth-Resource'.
       77 BAQR-OAUTH-AUDIENCE       PIC X(22)
           VALUE 'BAQHAPI-oAuth-Audience'.
       77 BAQR-OAUTH-CUSTOM-PARMS   PIC X(25)
           VALUE 'BAQHAPI-oAuth-CustomParms'.
       77 BAQR-JWT-USERNAME         PIC X(22)
           VALUE 'BAQHAPI-Token-Username'.
       77 BAQR-JWT-PASSWORD         PIC X(22)
           VALUE 'BAQHAPI-Token-Password'.

      * Host API ZCON parameter names
       77 BAQZ-TRACE-VERBOSE        PIC X(21)
           VALUE 'BAQHAPI-Trace-Verbose'.
       77 BAQZ-SERVER-URIMAP        PIC X(21)
           VALUE 'BAQHAPI-Server-URIMAP'.
       77 BAQZ-SERVER-HOST          PIC X(19)
MA    A
```

Connected to remote server/host wg31z using lu/pool TCP00111 and port 23

wg31 base

File Edit Settings View Communication Actions Window Help

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       USER1.ZCEE30.SOURCE(GETAPI) - 01.02            Columns 00001 00072
Command ===>                                               Scroll ===> PAGE
****** ******************************** Top of Data ********************************
000001          CBL APOST
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 61 Line(s) not Displayed
000063          01 JWT-USER  PIC X(10) VALUE 'myUsername'.
000064          01 JWT-PSWD  PIC X(10) VALUE 'myPassword'.
000065          01 JWT-SCOPE PIC X(10) VALUE 'myScope'.
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 56 Line(s) not Displayed
000122               *  Send JWT credentials to z/OS Connect
000123
000124                  MOVE BAQR-OAUTH-USERNAME TO
000125                       BAQ-REQ-PARM-NAME OF BAQ-REQ-PARMS(1)
000126                  SET BAQ-REQ-PARM-ADDRESS OF
000127                       BAQ-REQ-PARMS(1) TO ADDRESS OF JWT-USER
000128                  MOVE LENGTH OF JWT-USER TO
000129                     BAQ-REQ-PARM-LENGTH OF BAQ-REQ-PARMS(1)
000130
000131                  MOVE BAQR-OAUTH-PASSWORD TO
000132                       BAQ-REQ-PARM-NAME OF BAQ-REQ-PARMS(2)
000133                  SET BAQ-REQ-PARM-ADDRESS OF
000134                       BAQ-REQ-PARMS(2) TO ADDRESS OF JWT-PSWD
000135                  MOVE LENGTH OF JWT-PSWD TO
000136                       BAQ-REQ-PARM-LENGTH OF BAQ-REQ-PARMS(2)
000137
000138                  MOVE BAQR-OAUTH-SCOPE TO
000139                       BAQ-REQ-PARM-NAME OF BAQ-REQ-PARMS(3)
000140                  SET BAQ-REQ-PARM-ADDRESS OF
000141                       BAQ-REQ-PARMS(3) TO ADDRESS OF JWT-SCOPE
000142                  MOVE LENGTH OF JWT-SCOPE TO
000143                       BAQ-REQ-PARM-LENGTH OF BAQ-REQ-PARMS(3)
000144
000145               * Call the API endpoint using BAQEXEC
000146
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  400 Line(s) not Displayed
****** ***************************** Bottom of Data ********************************
MA    B                                                              37/002
```

Connected to remote server/host wg31z using lu/pool TCP00103 and port 23

# 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>
```

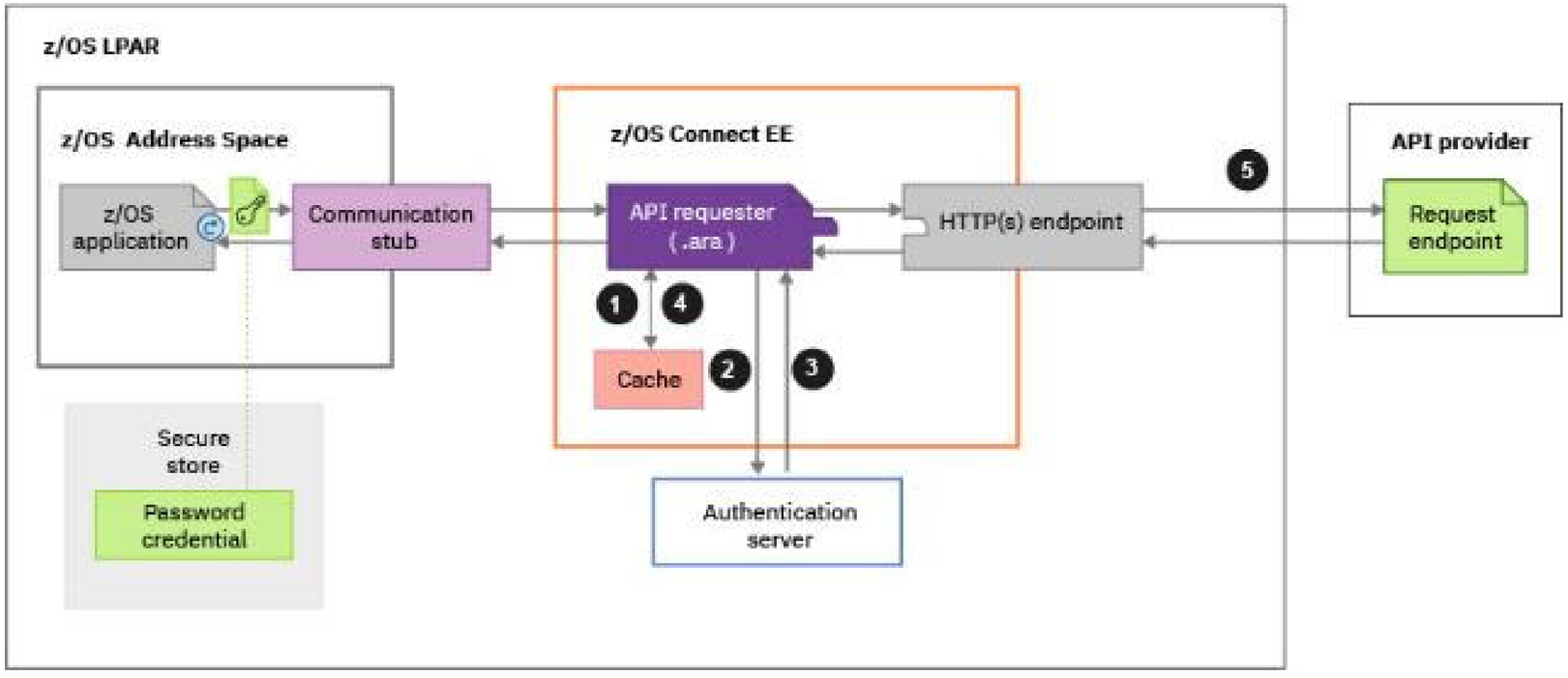
[1] See URL https://www.ibm.com/support/knowledgecenter/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_oidc_token_endpoint.html

[2] These credentials can be specified by the application

mitchj@us.ibm.com

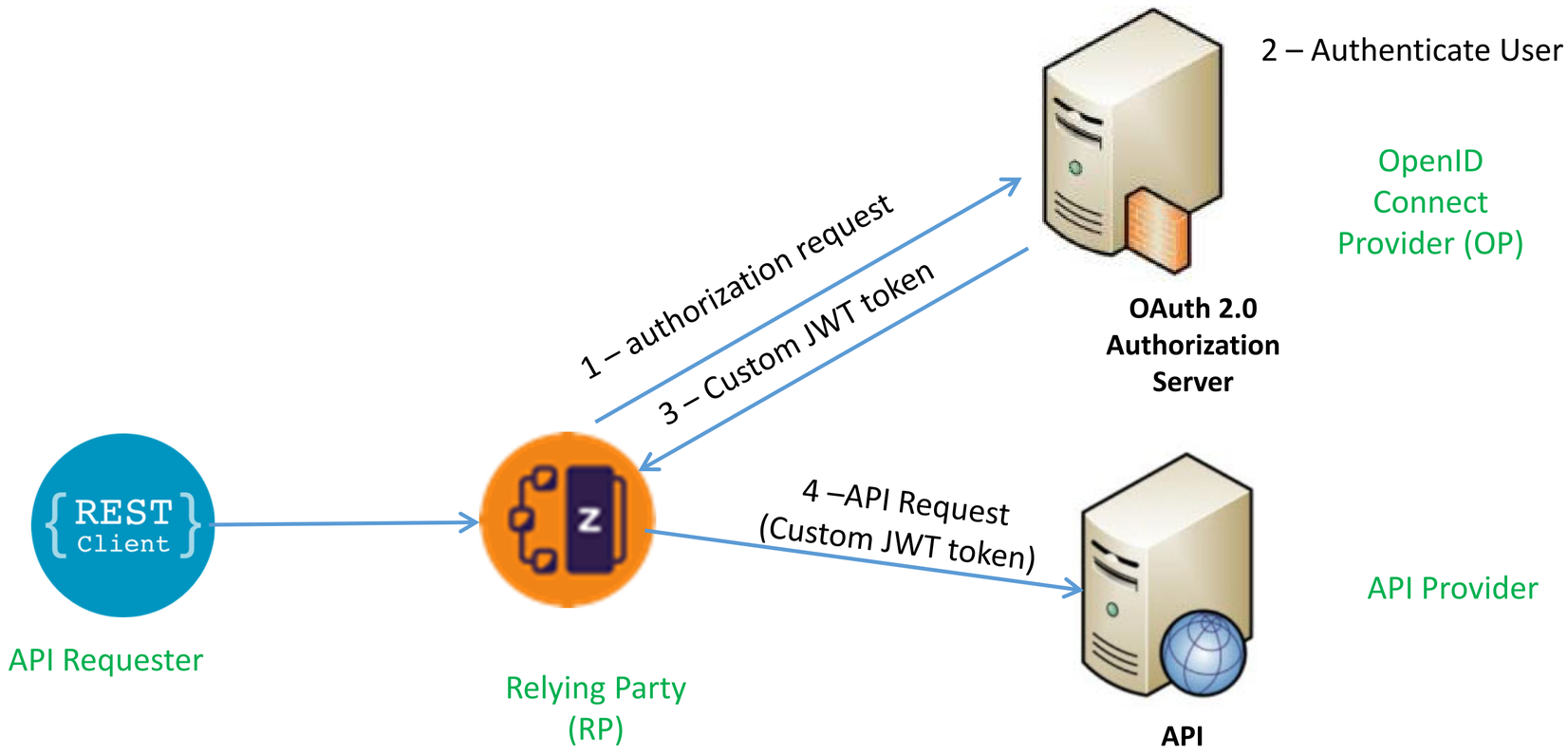
# Calling an API with using a JWT custom flow

- 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 Custom Flow




mitchj@us.ibm.com

# API Requester – JWT Custom flow

wg31 master

File Edit Settings View Communication Actions Window Help

Menu Utilities Compilers Help

```
BROWSE    ZCEE30.SBAQCOB(BAQRINFO)                  Line 0000000028 Col 001 080
Command ===>                                                 Scroll ===> PAGE
      01  BAQ-REQUEST-INFO.
        03 BAQ-REQUEST-INFO-COMP-LEVEL  PIC S9(9) COMP-5 SYNC VALUE 4.
        03 BAQ-REQUEST-INFO-USER.
           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
                                                VALUE 0.
           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.
           05 BAQ-ZCON-SERVER-URI             PIC X(256)
                                                VALUE SPACES.
MA    A                                                        04/015
```

Connected to remote server/host wg31z using lu/pool TCP00145

**BAQRINFO copy book**

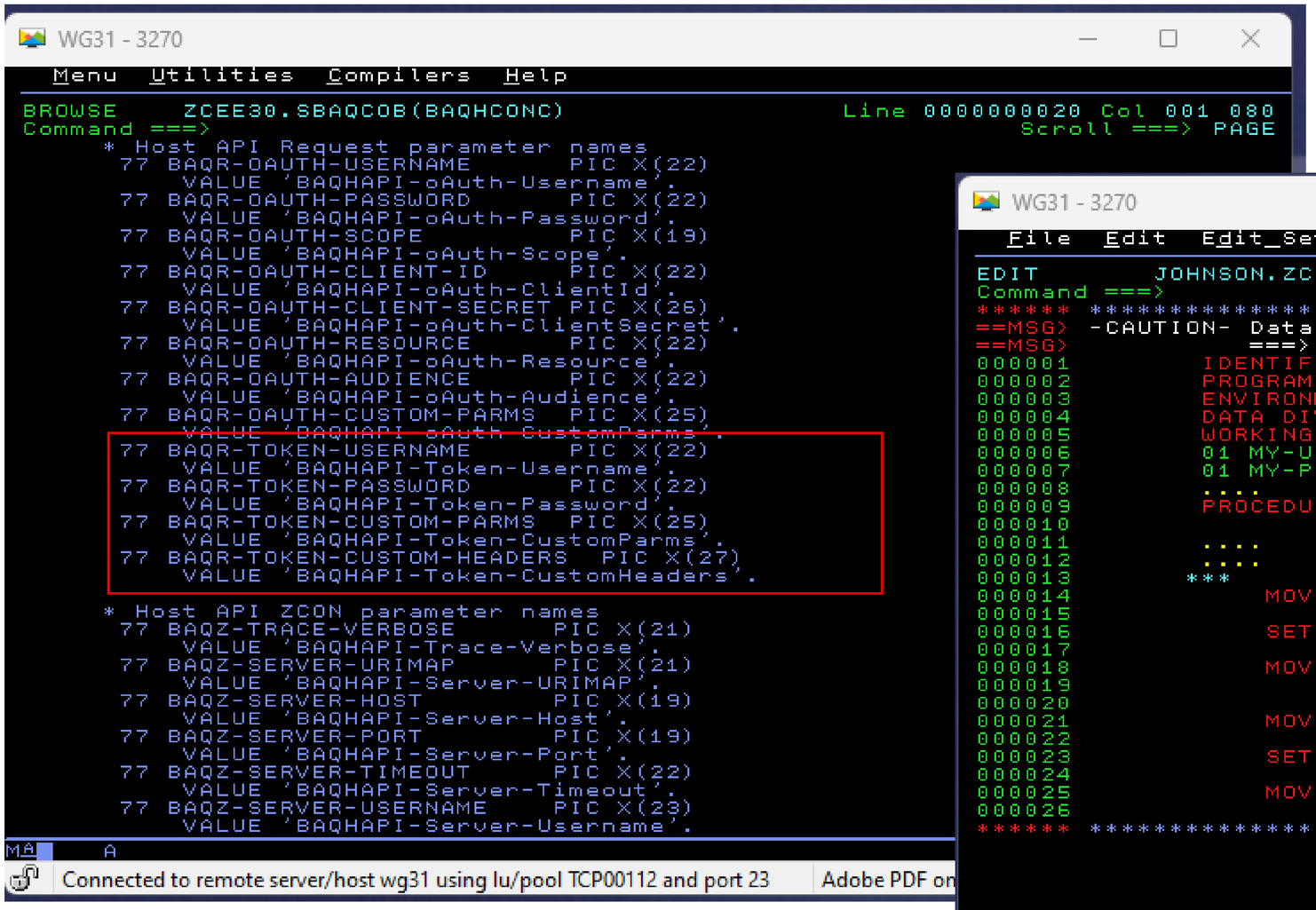
## COBOL application

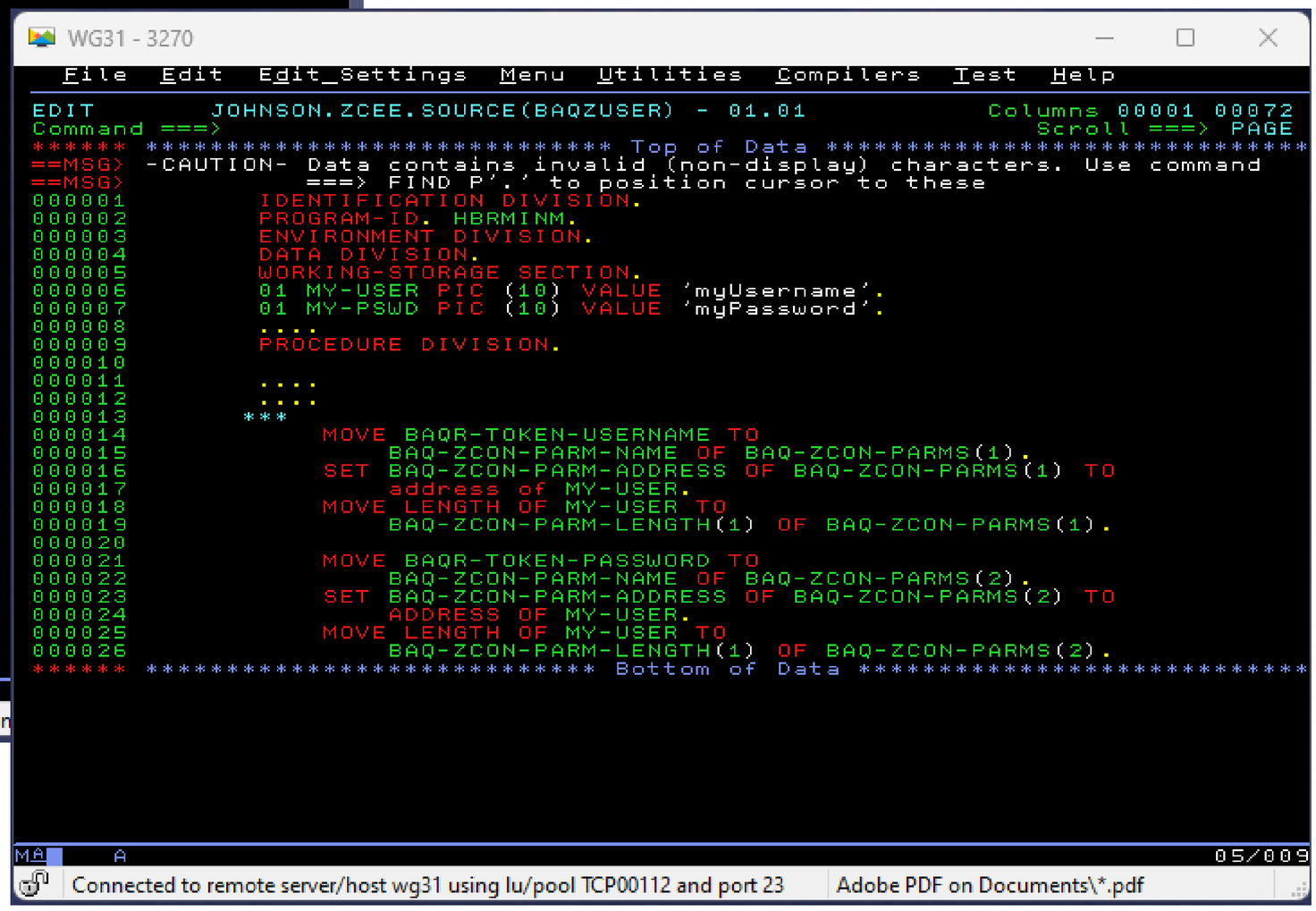
```
MOVE "ATSTOKENUSERNAME" to envVariableName.
PERFORM CALL-CEEENV THRU CALL-CEEENV-END
MOVE VAR(1:valueLength) to BAQ-TOKEN-USERNAME
MOVE valueLength TO BAQ-TOKEN-USERNAME-LEN
MOVE "ATSTOKENPASSWORD" to envVariableName.
PERFORM CALL-CEEENV THRU CALL-CEEENV-END
MOVE VAR(1:valueLength) to BAQ-TOKEN-PASSWORD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

# API Requester – JWT Custom flow




mitchj@us.ibm.com

# Configuring JWT Custom flow

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

<zosconnect_authToken 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

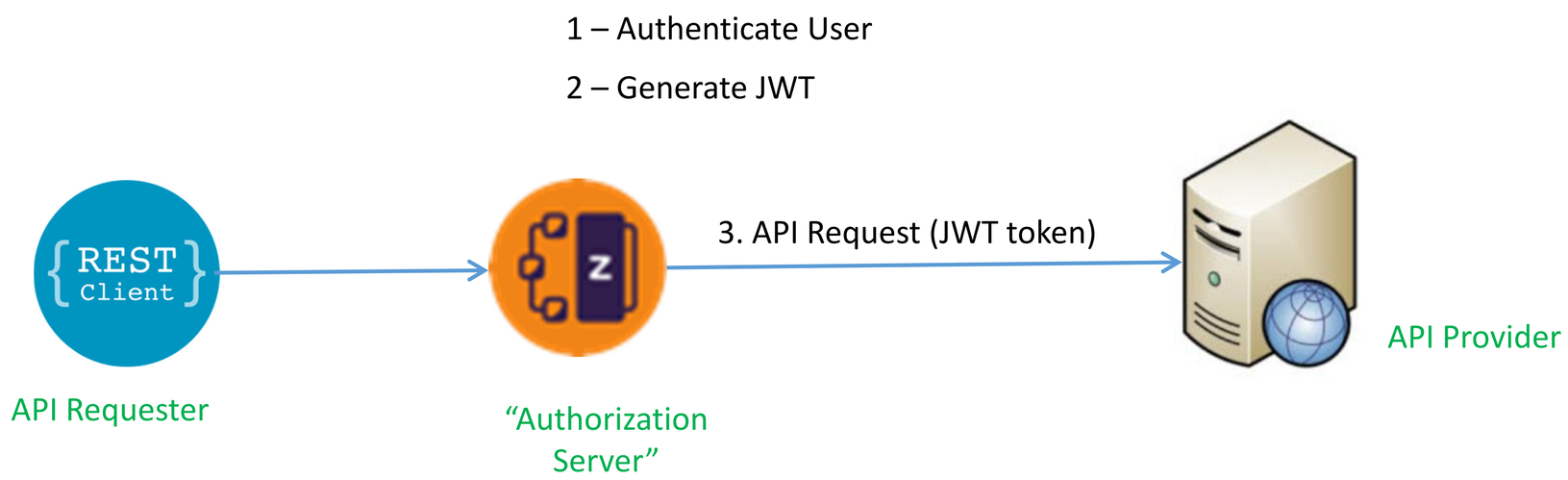
```
<tokenRequest credentialLocation="body"
   requestMethod="POST“    // Use XML escaped characters in requestBody
  requestBody="{&quot;apiuser&quot;:&quot;${userid}&quot;,&quot;apipassword&quot;:&quot;${password}&quot;}" />
```

Response Token

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

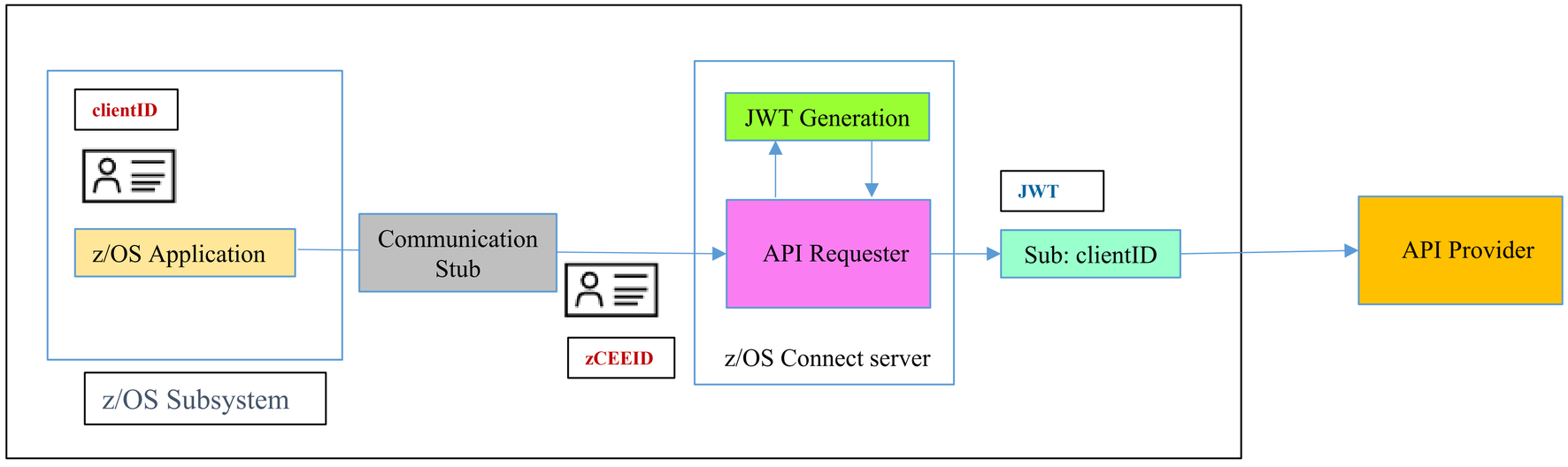
mitchj@us.ibm.com

# 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

# JWT generation requires setting a program control extended attribute

**As root or superuser, set the *libifaedjreg64.so* program control extended attribute bit**

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

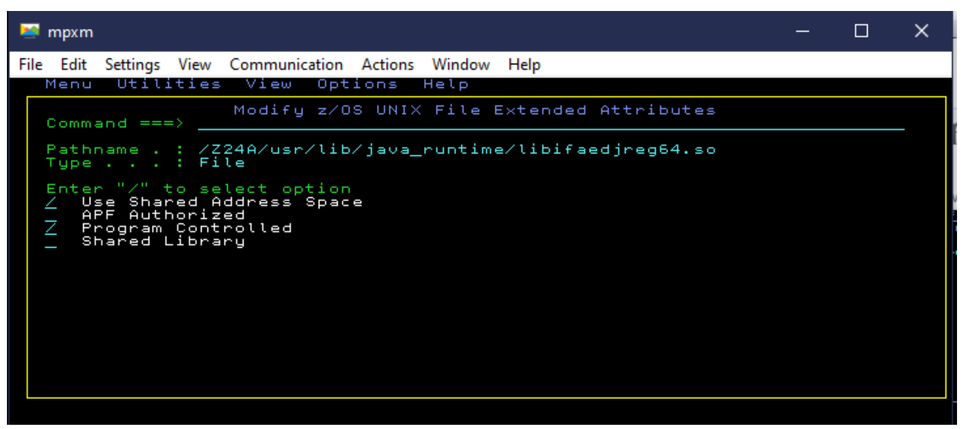
  ***PERMIT BPX.SERVER CLASS(FACILITY) ID(LIBSERV) ACCESS(READ)***
  ***SETROPTS RACLIST(FACILITY) REFRESH***

- ***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**

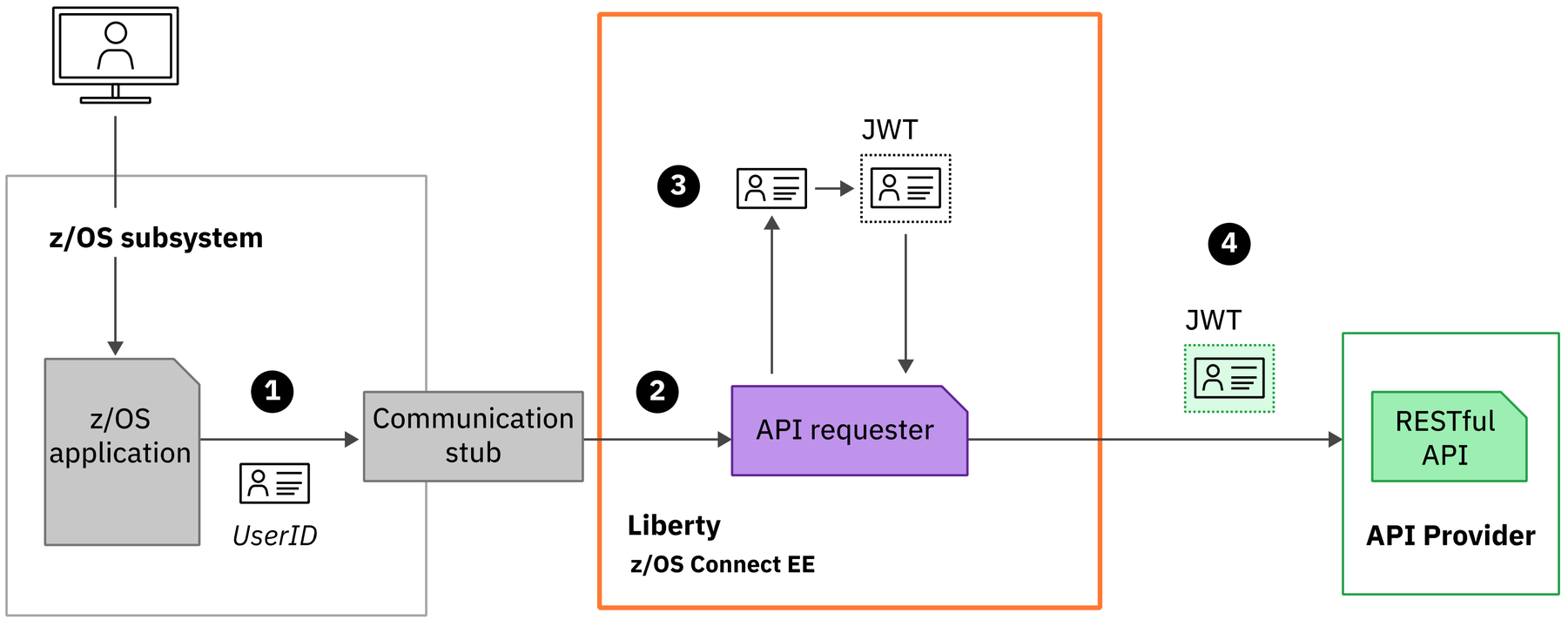
- ***Enable the program control bit for Java shared object ifaedjreg64***

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




mitchj@us.ibm.com

# JWT Generation




1. Communication stub extracts the ID from the application environment
2. z/OS Connect generates a JWT token containing the z/OS application asserted user ID
3. The JWT is used to authorise the request to the API endpoint

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

# server XML Configuration

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

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

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

Configure the Liberty jwtBuilder element in server.xml.

Configure the zosconnect_authTokenLocal element, specifying any additional private claims required and the name of the header used to send the JWT to the endpoint.

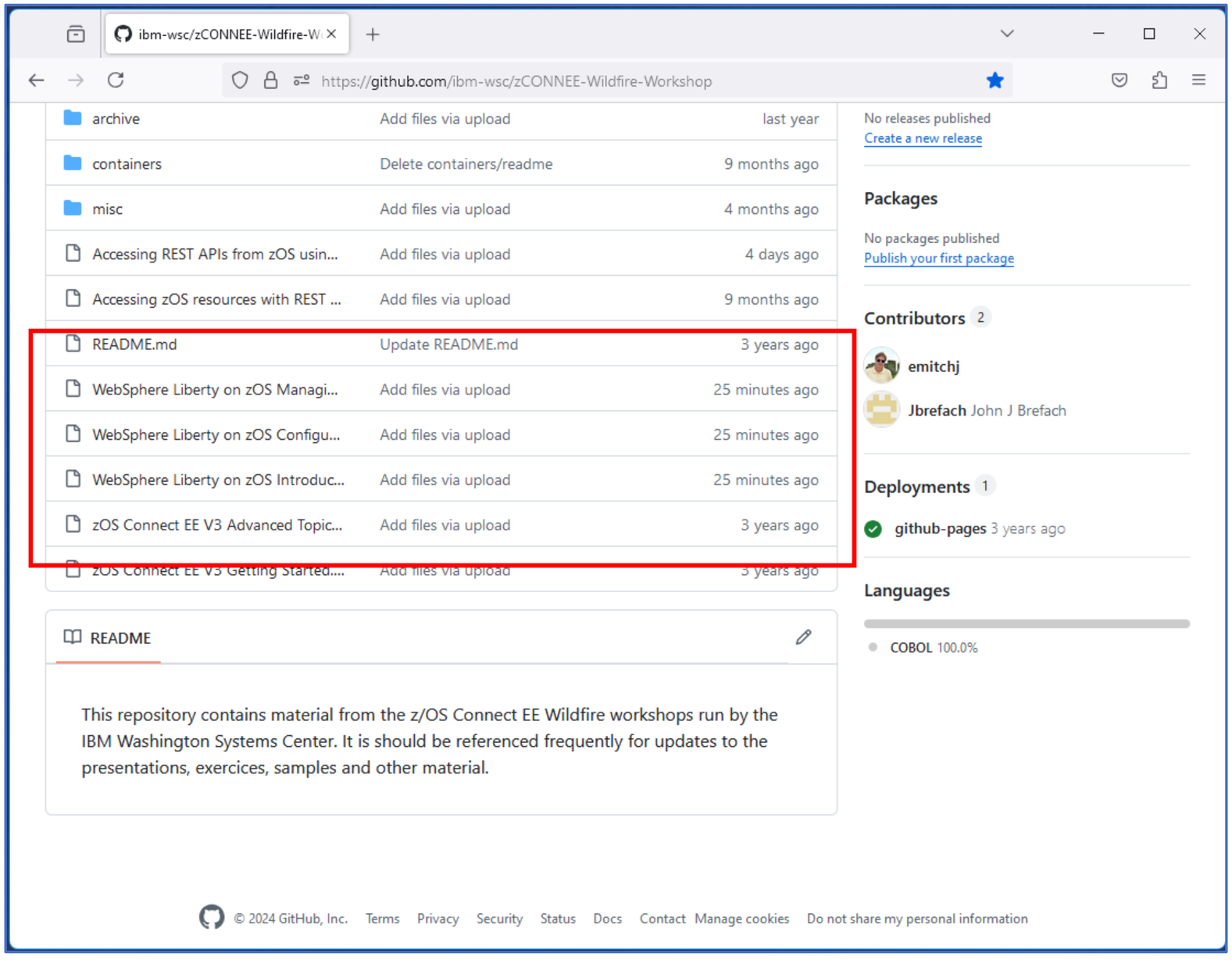
header default value is Authorization

Finally, reference the JWT configuration from the zosconnect_endpointConnection element.

mitchj@us.ibm.com

# Liberty Wildfire Github Site

https://ibm.biz/BdPRGD




mitchj@us.ibm.com

Thank you for listening and your questions.

IBM Washington Systems Center

The WSC Requests your Feedback!

Please scan the QR Code or go to this link:
https://ibm.biz/wsc-survey

For Event Code, Enter: **ZOSCON** + the current date **(DDMMYY)**

Thank you for listening and your questions.

mitchj@us.ibm.com

# Miscellaneous Odds and Ends

mitchj@us.ibm.com

# Tech-Tip: RACF resources for using PassTickets

- Aa PTKTDATA resource is defined 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 Liberty, the APPLDATA value is the value of the *profilePrefix* in the *safCredentials* configuraton element. For example, to use PassTickets to access Liberty server the resources below need to be defined, and appropriate access granted.

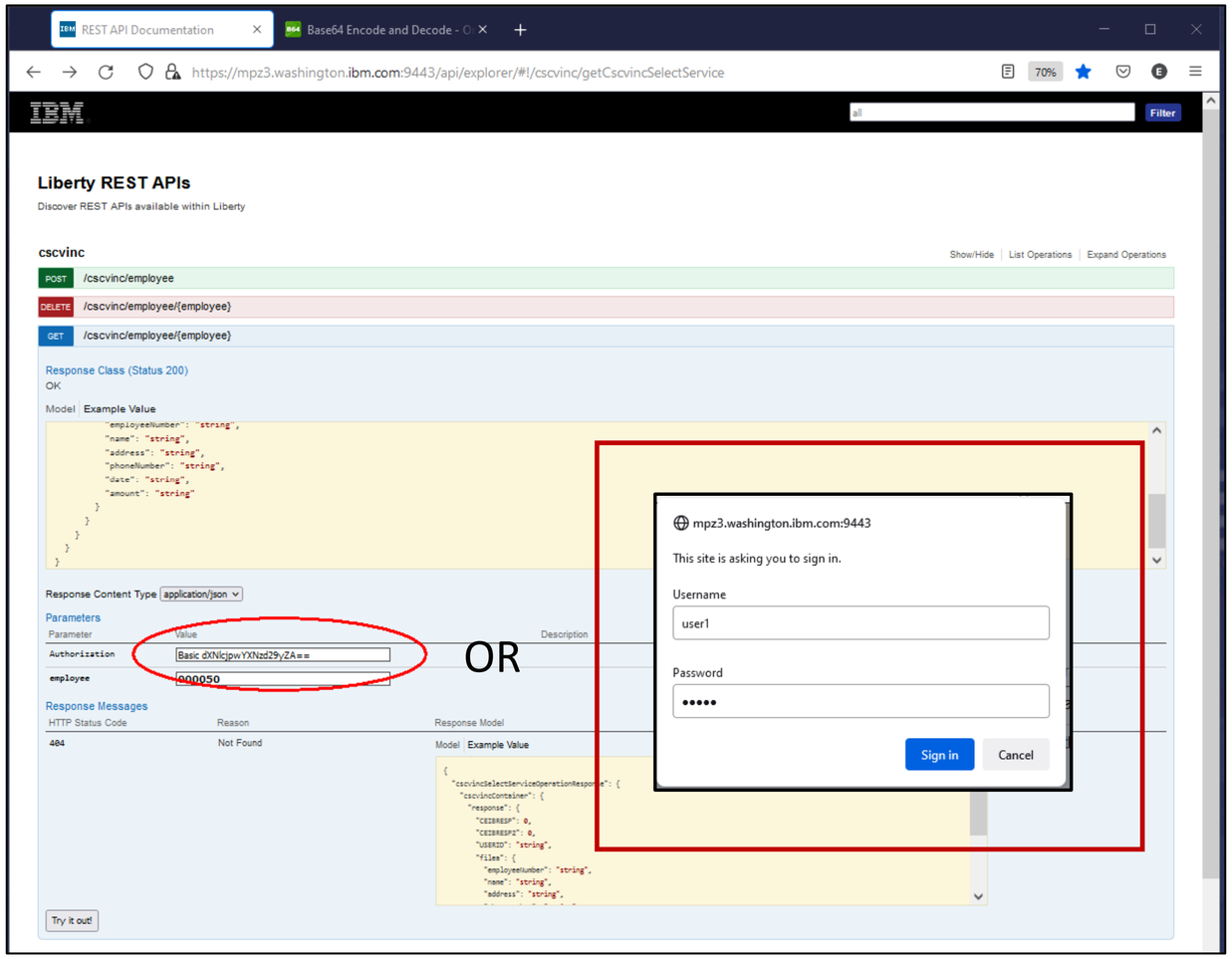
```
<safRegistry id="saf" />
   <safAuthorization racRouteLog="ASIS" />
   <safCredentials unauthenticatedUser="WSGUEST"
      profilePrefix="BBGZDFLT" />
```

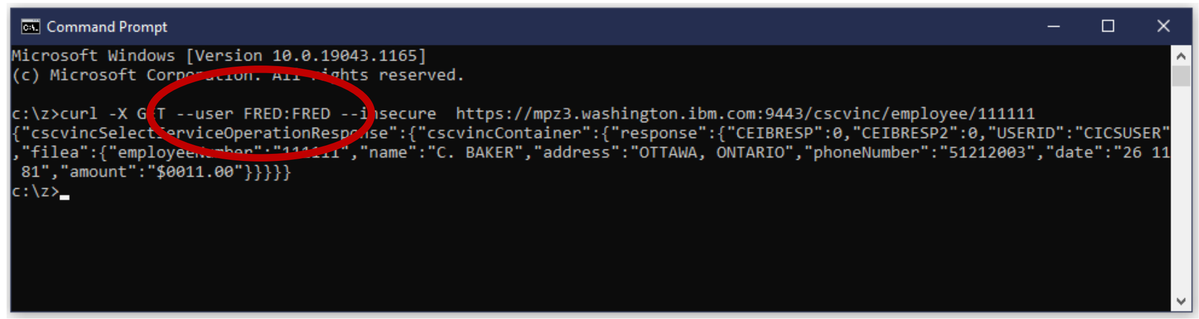
```
RDEFINE PTKTDATA BBGZDFLT SSIGNON(0123456789ABCDEF))
          APPLDATA('NO REPLAY PROTECTION') UACC(NONE)
RDEFINE PTKTDATA IRRPTAUTH.BBGZDFLT.* UACC(NONE)
PERMIT IRRPTAUTH.BBGZDFLT.* ID(LIBSERV) CLASS(PTKTDATA) ACCESS(UPDATE)
PERMIT IRRPTAUTH.BBGZDFLT.USER1 ID(USER1) CLASS(PTKTDATA) ACCESS(UPDATE)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

mitchj@us.ibm.com

# There are multiple ways to provide an identity and password

- When sending a request to a Liberty server, 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.
  - Examples using the API Explorer feature , cURL, and a Java client.





https://www.base64encode.org/

mitchj@us.ibm.com

# Tech/Tip:  Details of the flow with mutual authentication

1. A Client sends a request to server for a protected session in a ***ClientHello*** message. Included in the request is the TLS capabilities of the client (e.g., TLS 1.2 or 1.3) and a list of supported ciphers in preference order.
2. The server selects the TLS version and selects cipher from the list sent by the client and returns this information in a ***ServerHello*** message.
3. The server's certificate information (including the public key) is sent to the client in a ***Certificate*** message.
4. The server sends cryptographic information for the client to use for encrypting a pre-master key in a **Server key exchange** message.
5. **For mutual authentication, the server sends a *CertificateRequest* message requesting a client's personal certificate.**
6. The server concludes by sending a ***ServerHelloDone*** message.
7. The client verifies the server's certificate with its trust store.
8. **If mutual authentication is requested, the client sends its personal certificate in a *Certificate* message**
9. The client then uses the server's public key to generate and encrypt a 48 byte "premaster secret" message which is sent to the server in a ***ClientKeyExchange*** message (**asymmetric encryption**)
10. **When mutual authentication is requested, a digitally signature (hashed) of the concatenation of all previous handshake messages is encrypted with the client's private key sent in a *CertificateVerify* message.**
11. The ***Change Cipher*** message is used to change the from cipher used during the handshake so all subsequent messages will be encrypted using a different cipher.
12. The server uses its private key to decrypt the "premaster secret" message (only the private key can be used to decrypt the message).
13. **If mutual authentication is requested, the server verifies the client's personal certificate with its key ring and uses the client's public key to decrypt and verify the message sent in the *CertificateVerify* message.**
14. Both the Client and Server use the "premaster secret" to compute a 'master secret', also know as "shared secret" or "session key" (**symmetric encryption**)
15. Client and server will use this "shared secret" or "session key" to encrypts messages sent between the endpoints.

At a regular interval, the two endpoints will use the asymmetric keys to generate a new symmetric key

mitchj@us.ibm.com

# Tech/Tip: Using a cURL trace to show the flow with mutual authentication

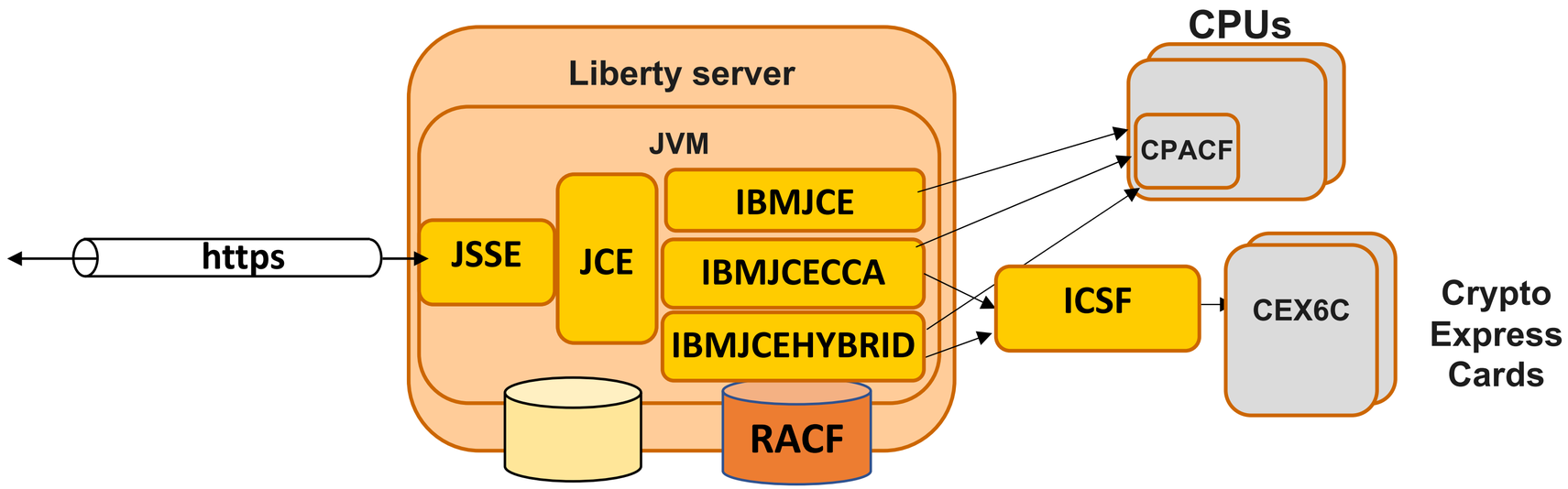
```
* successfully set certificate verify locations:
* TLSv1.3 (OUT), TLS handshake, Client hello (01):
* TLSv1.3 (IN), TLS handshake, Server hello (02):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Request CERT (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.
```

```
enum {
hello_request(0),
client_hello(1),
server_hello(2),
certificate(11),
server_key_exchange (12),
certificate_request(13),
server_hello_done(14),
certificate_verify(15),
client_key_exchange(16),
finished(20),
 (255) }
 HandshakeType;
```

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

# Liberty and using Java Secure Socket Extension (JSSE)

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

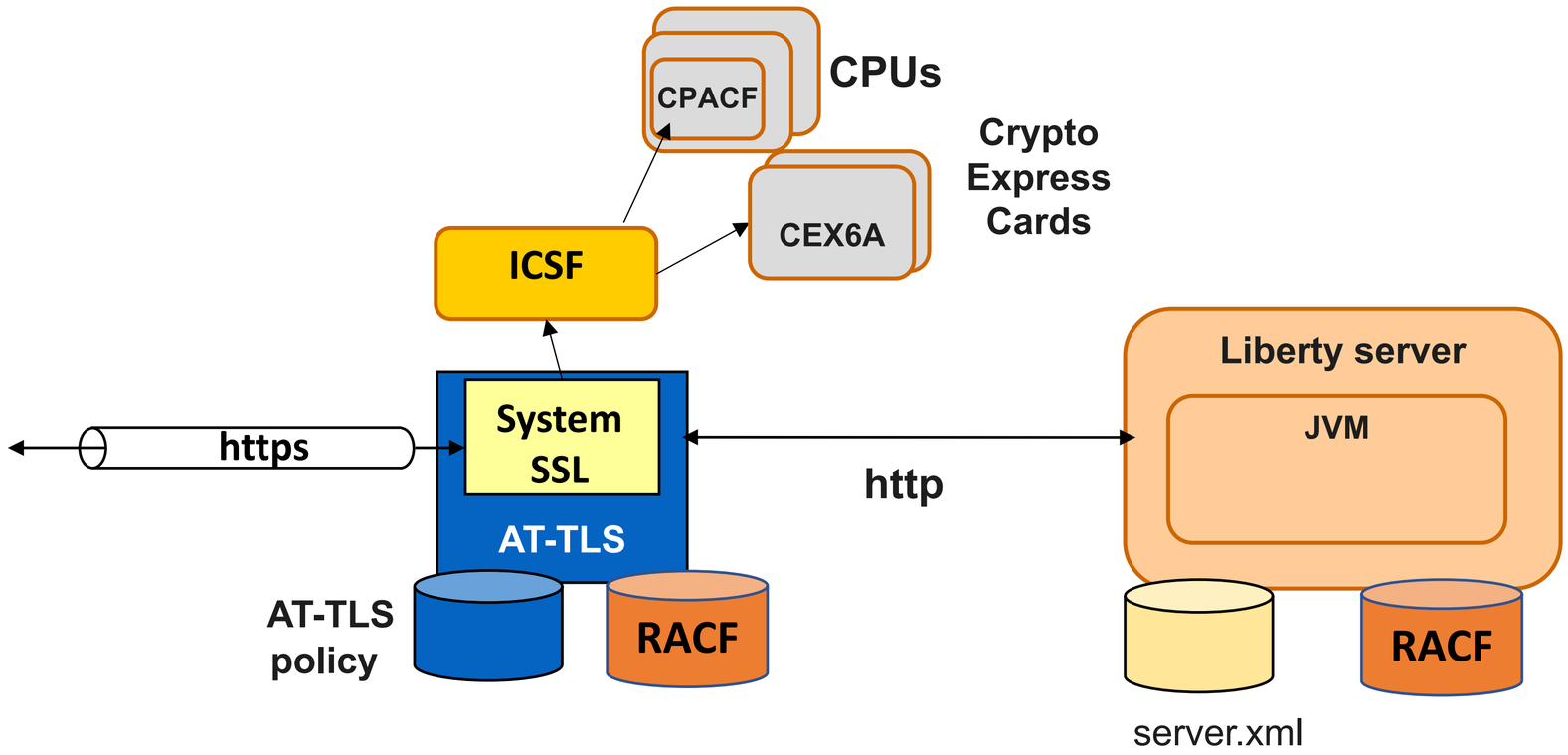



- z/OS Connect EE support for TLS is based on **Liberty** server support
- **Java Secure Socket Extension** (JSSE) API provides framework and Java implementation of  TLS protocols used by Liberty HTTPS support
- **Java Cryptography Extension** (JCE) is standard extension to the Java Platform that provides 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

# Liberty and using AT-TLS

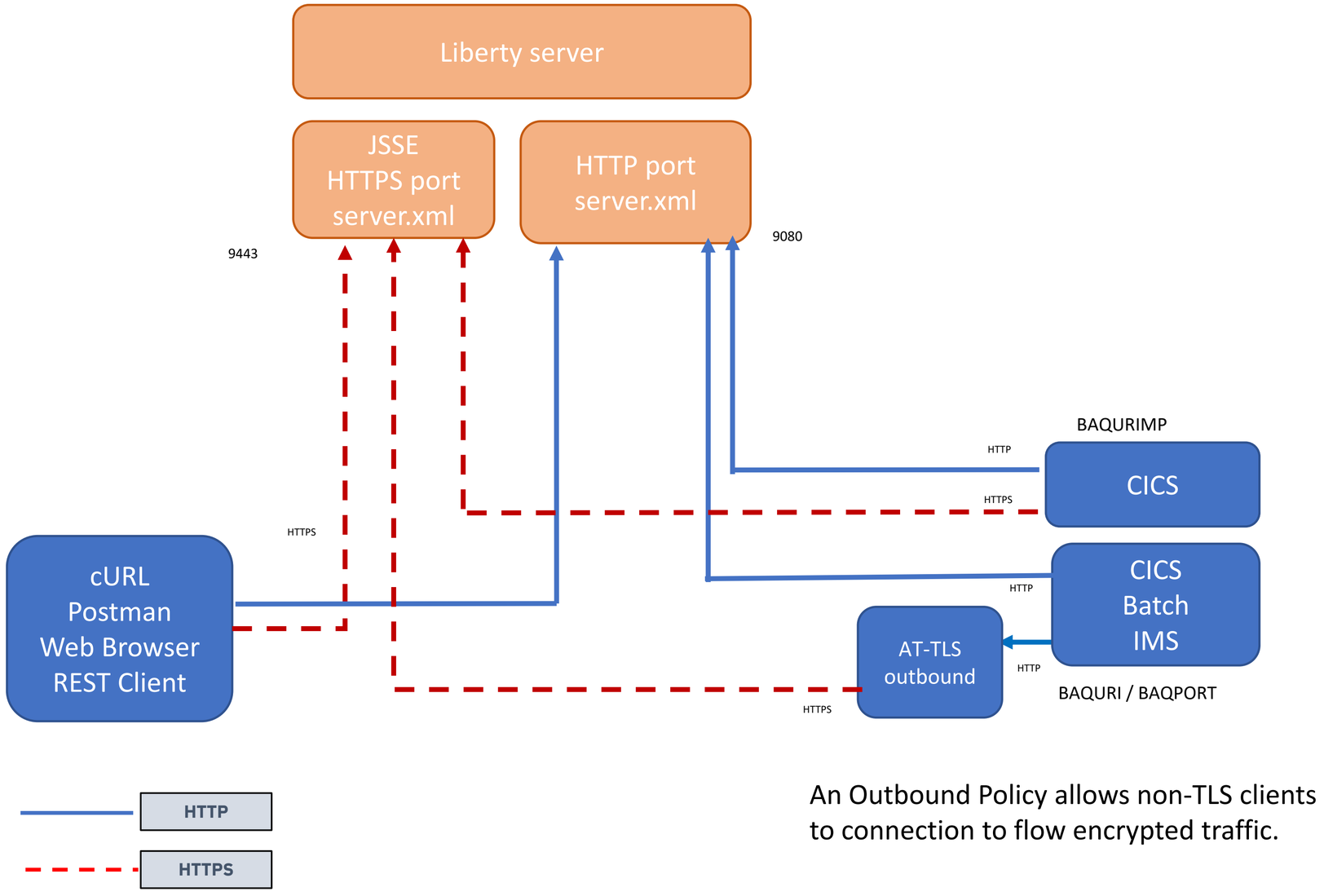
**The server XML configuration uses <u>no</u> HTTPS protocol, key rings or other JSSE attributes**




- **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

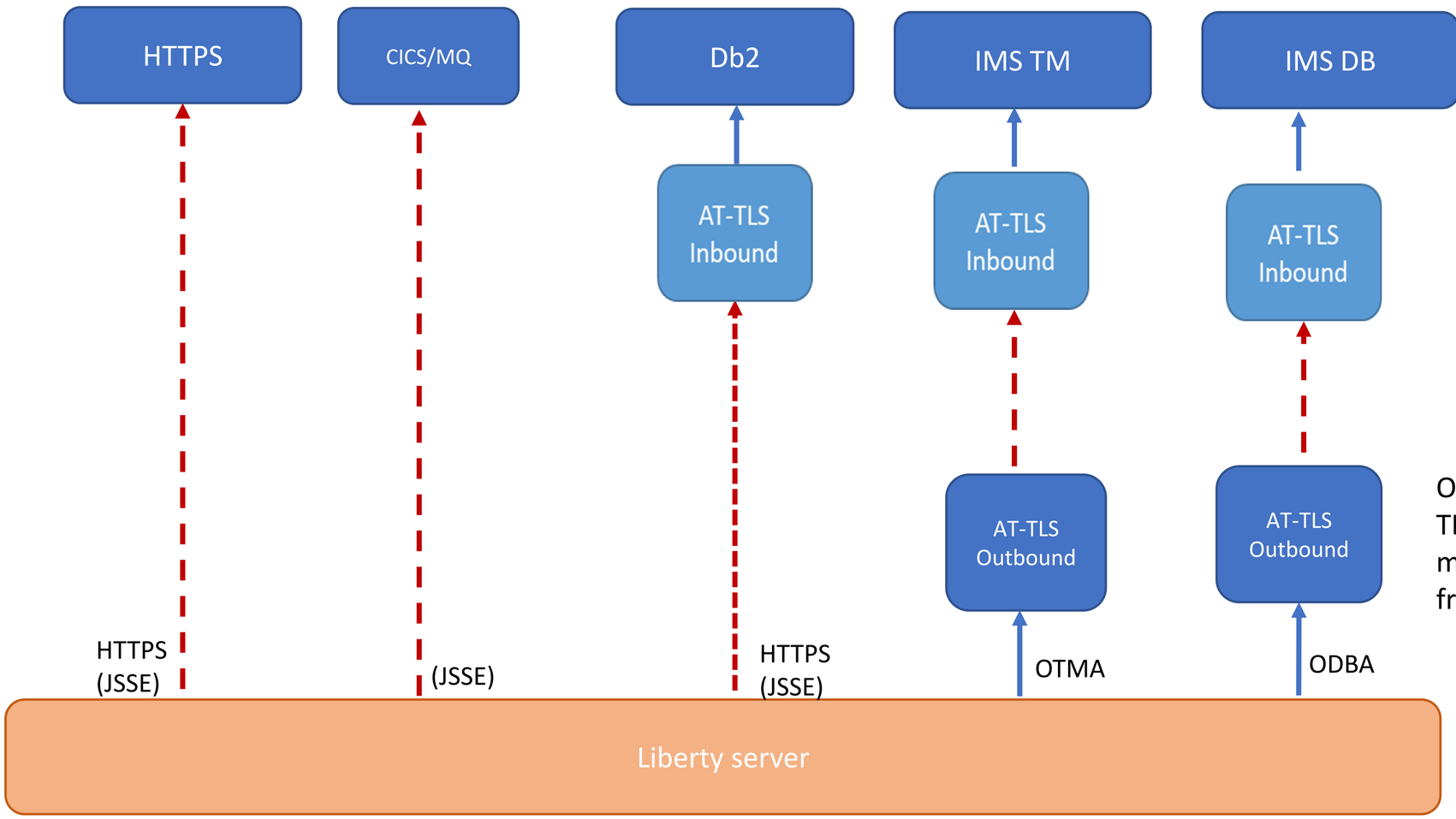
# Tech-Tip: TLS client encryption to a Liberty server scenarios




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

[1] mitchj@us.ibm.com

# Tech-Tip: TLS encryptions from a Liberty server (HTTPS/native to HTTPS/TLS/OTMA/ODBA)




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

Outbound Policies provide TLS support for non-JSSE managed request outbound from Liberty

mitchj@us.ibm.com

# Tech-Tip: Enabling hardware cryptography

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

# 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.

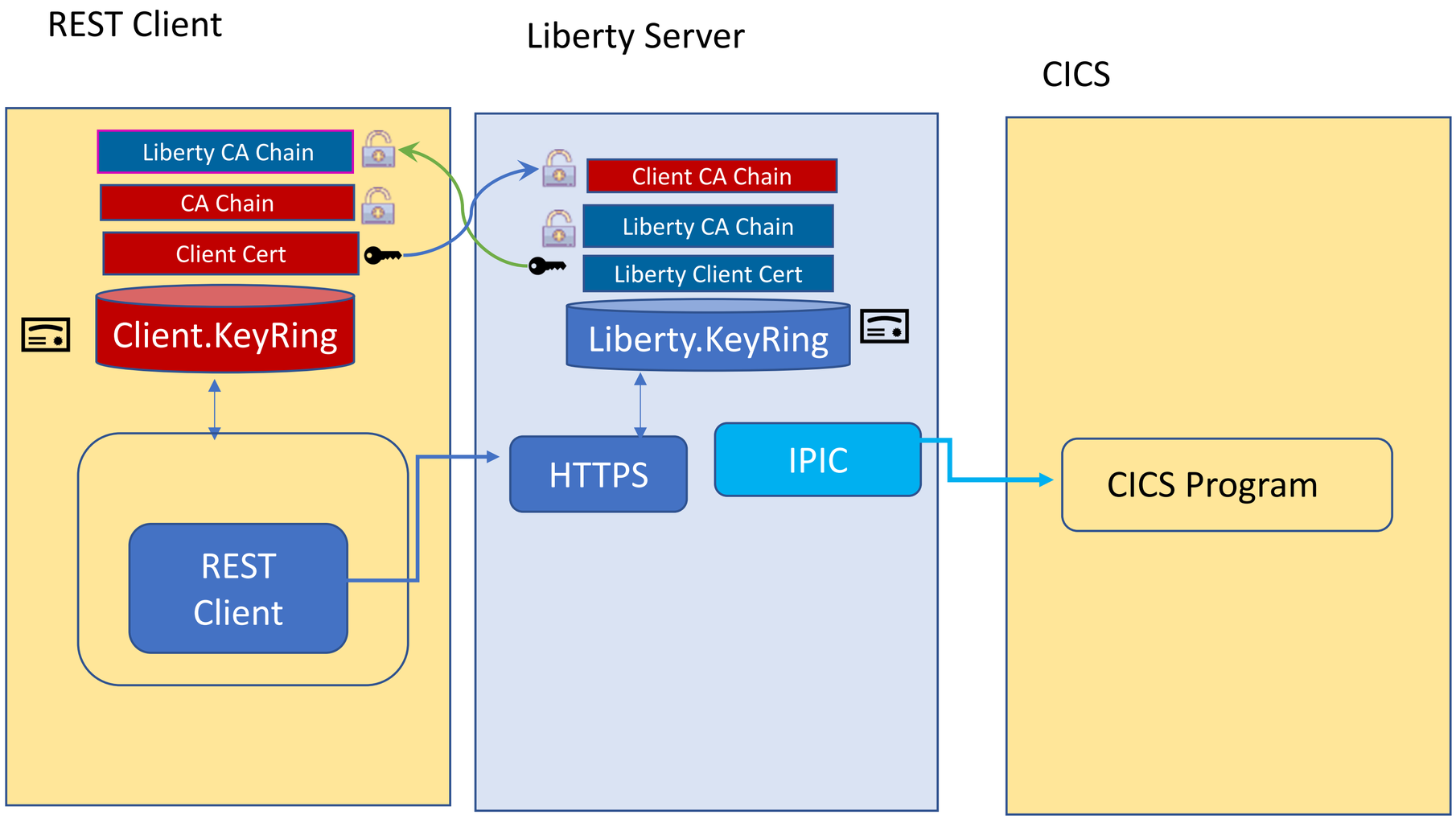
*Table 1. CipherSpecs supported by IBM MQ and their equivalent CipherSuites*

| 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/docs/en/ibm-mq/9.3?topic=java-tls-cipherspecs-ciphersuites-in-mq-classes

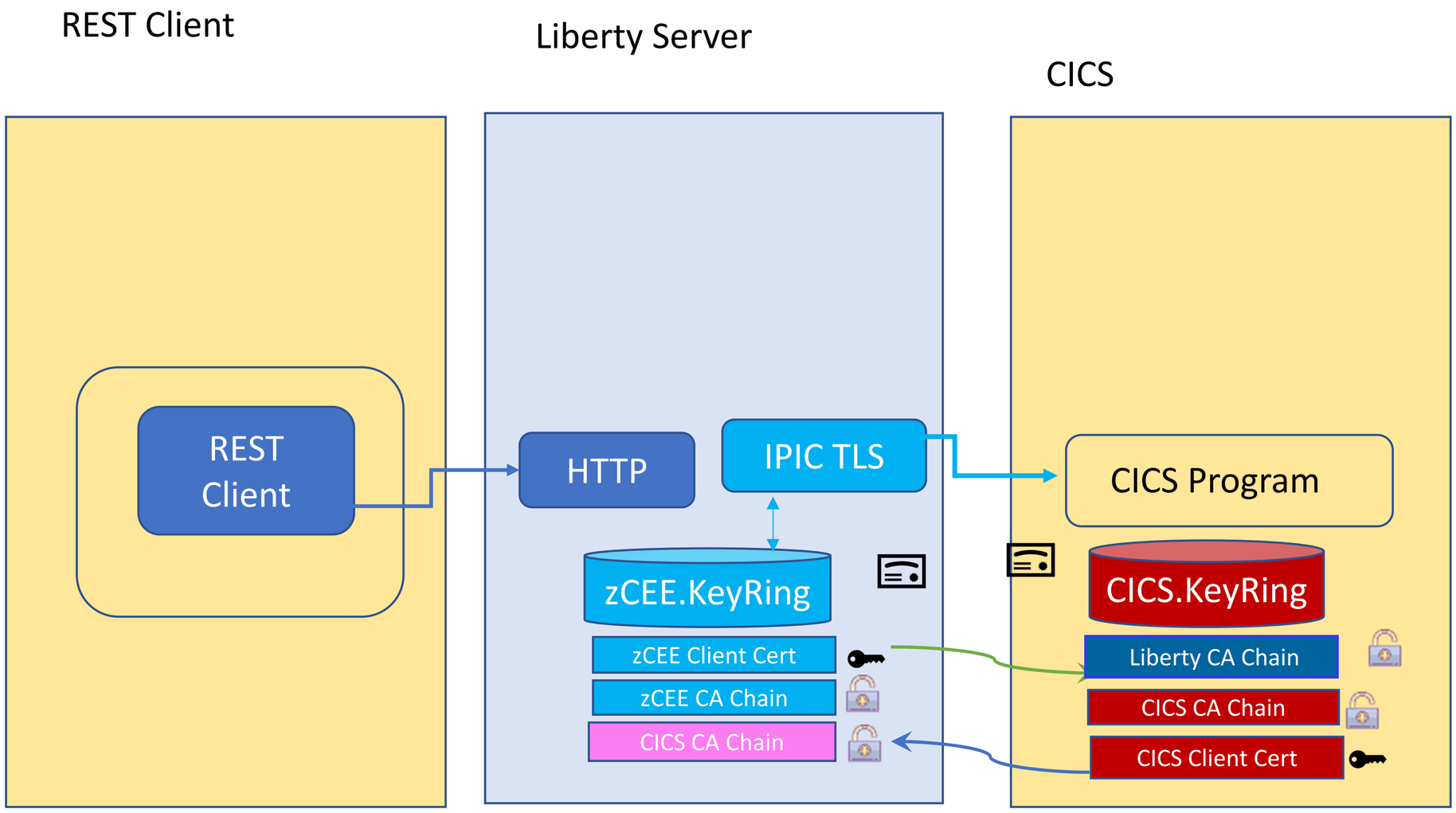
mitchj@us.ibm.com

# TLS handshake between the client and the Liberty server




mitchj@us.ibm.com

# TLS handshake between the Liberty server and the target endpoint

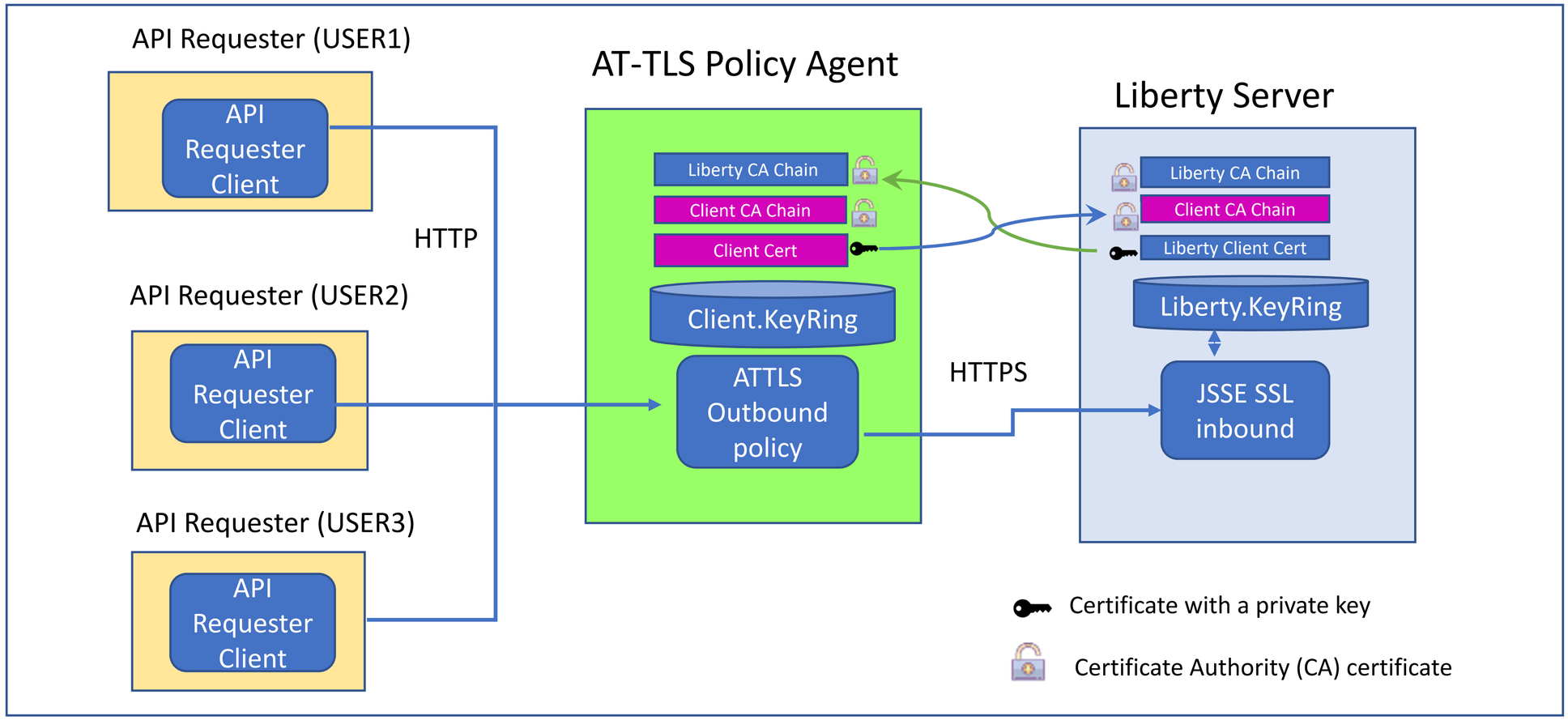



```
KEYRING=CICS.KeyRing,
ENCRYPTION=STRONG,
MAXSSLTCBS=8,
SSLDELAY=600
```

mitchj@us.ibm.com

# AT-TLS - outbound policy handshake scenario

Use of a common key ring name for multiple client identities



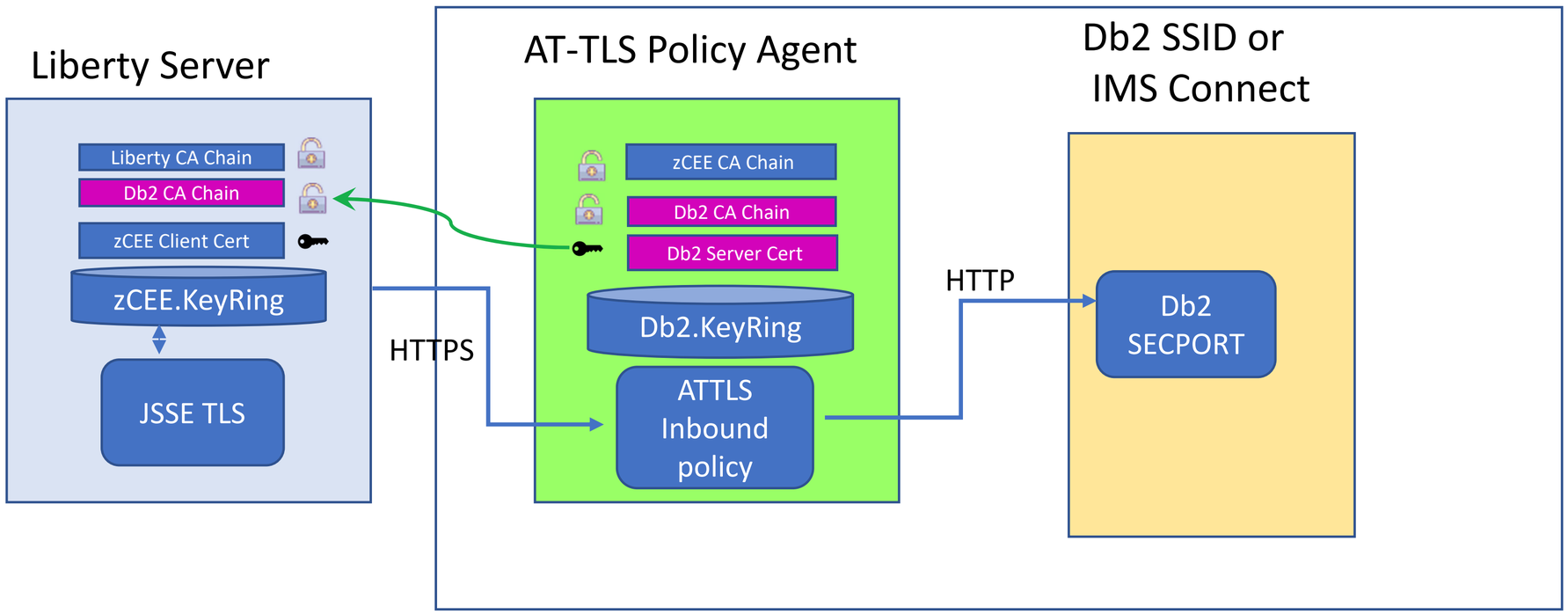

- Each user owns a keyring with the name Liberty.KeyRing.
- Each key ring has a different default client certificate for mutual authentication purposes.

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

mitchj@us.ibm.com

# AT-TLS - inbound policy handshake scenario (Db2 and IMS)

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




Note that DB2 is AT-TLS aware
IMS is AT-TLS unware

Certificate with a private key

Certificate Authority (CA) certificate

mitchj@us.ibm.com

# Tech/Tip: Let's explore a flow using a Liberty OpendID Provider as an example

This Liberty server configuration provides a good example of the workings of an authorization server.

```
<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>
</oauthProvider>
```

**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

# z/OS Connect Security server XML Authentication Configuration (OpenAPI 2)

- requireAuth - requires the client to provide credentials

```
<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>
```

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.

**The requireAuth attribute controls whether an inbound request must provide credentials using one of the three authentication methods, e.g., basic, client certificate, or third-party token.**

**Note that there are no equivalent configuration elements for an z/OS Connect OpenAPI 3 server.**

mitchj@us.ibm.com

# z/OS Connect Security server XML TLS Security Configuration (OpenAPI 2)

- requireSecure - requires the client to connect using HTTPS

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

Globally, requires that client connections use HTTPS, unless overridden on the specific resource definitions.

Requires that client connections use HTTPS (true) or HTTP(false) in order to access the API.

Requires that client connections use HTTPS (true) or HTTP(false) to directly access the service. This attribute is ignored when the service is invoked from an API, then only the API requireSecure attribute is relevant.

Requires that client connections use HTTPS (true) or HTTP(false) ito access the PI requesters. If the requireSecure attribute is not set, the global setting on the zosconnect_zosConnectManager element is used instead, unless the requireSecure attribute is overridden on the specific API requester.

**The requireSecure attribute controls whether an inbound must be using HTTPS(true) or if HTTP(false) is allowed.**

**Note that there are no equivalent configuration elements for an z/OS Connect OpenAPI 3 server.**

mitchj@us.ibm.com

# Tech-Tip: 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

# Tech-Tip: 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

# Tech/Tip: Generating a JWT using Liberty's as an example OPID provider

The Liberty server authorization server's XML configuration

```
<!--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="distributed_User1" 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('distributed_User1'))
     REGISTRY(NAME('*')) WITHLABEL('zCEE JWT distributedUser1')
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')
```

[1] mitchj@us.ibm.com

# Tech-Tip: 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    )
```

Number of IPCONN resources equals the number of zCEE server times the number of unique connection references

[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 connection processing for high availability load balancing*

If the *reconnectInterval* attribute is set, at the specified time interval, a check is made to see if a new connection attempt should be attempted

A new connection is established if the current connection properties are not the preferred connection properties:

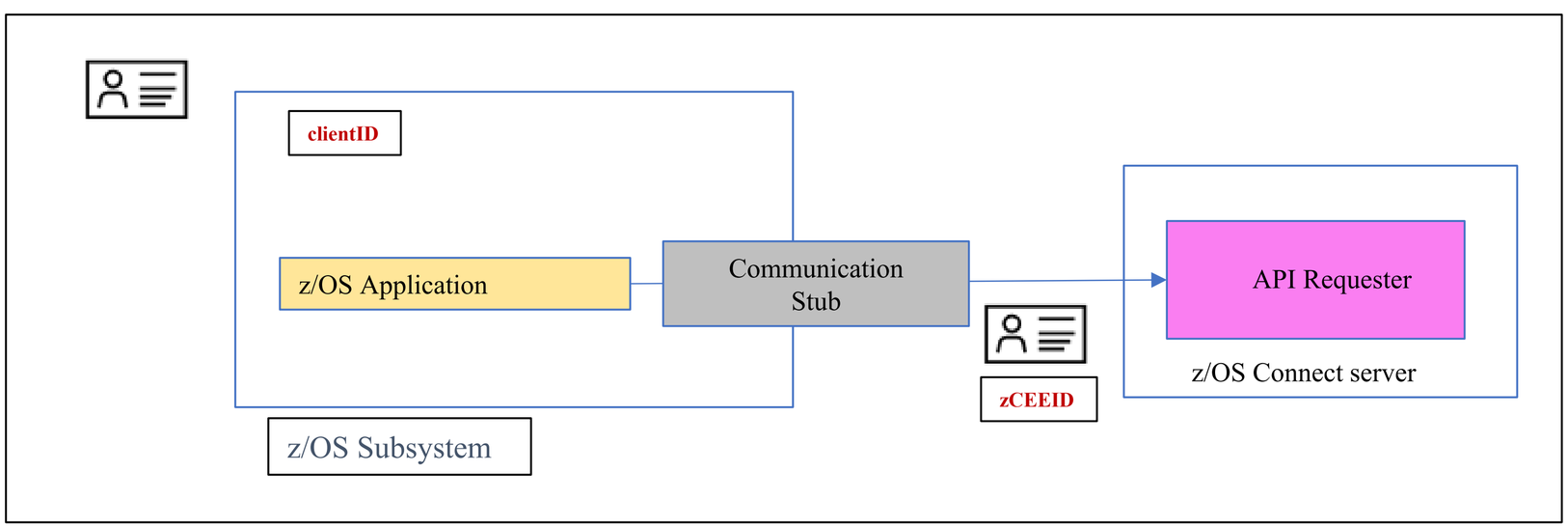
- If *reconnectInterval, preferredSpecificHost* and *preferredSpecificPort* are not set,
  - New connection attempts are disabled (this is the default behavior).
- If *reconnectInterval* is set and *preferredSpecificHost* and *preferredSpecificPort* are not set,
  - A new connection is attempted at the interval specified by the *reconnectInterval* time. Use this to enable regular connection rebalancing.
- If *reconnectInterval* and *preferredSpecificPort* are set and *preferrredSpecificHost* is not set,
  - A new connection is attempted at the expiration time interval and if the current connected port in use does not match the preferred port
  - Relevant when shared port is for a single LPAR
  - Specific CICS region is preferred
- If *reconnectInterval* and *preferredSpecificHost* are set and *preferredSpecificPort* is not set
  - A new connection is attempted at the expiration time interval and if the current host in use does not match the preferred port
  - Relevant when shared port is across Sysplex
  - Any CICS region on a specific LPAR is preferred
- If *reconnectInterval*, *preferredSpecificHost* and *preferredSpecificPort* are all set
  - A new connection is attempted at the expiration time interval time and if both the current host and port in use do not match the preferred host and port
  - Relevant when shared port is on a single LPAR or across a Sysplex
  - Specific CICS region is preferred.

When the reconnection attempt results in a new connection to a CICS region, new requests are sent over the new connection. Previous connections will continue and when all requests have completed processing, the previous or old connection will be closed.

mitchj@us.ibm.com

*Available at V3.0.56 and later

# 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. For MVS batch, IMS and Db2 stored procedures, the *zCEEID* is the provided by the environment variable ***BAQUSERNAME***. For CICS, the value for *zCEEID* is usually provided by the identity mapped to the CICS client certificate.

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

- For CICS, the CICS task identity
- For IMS, the transaction owner
- For batch, the job card USERID

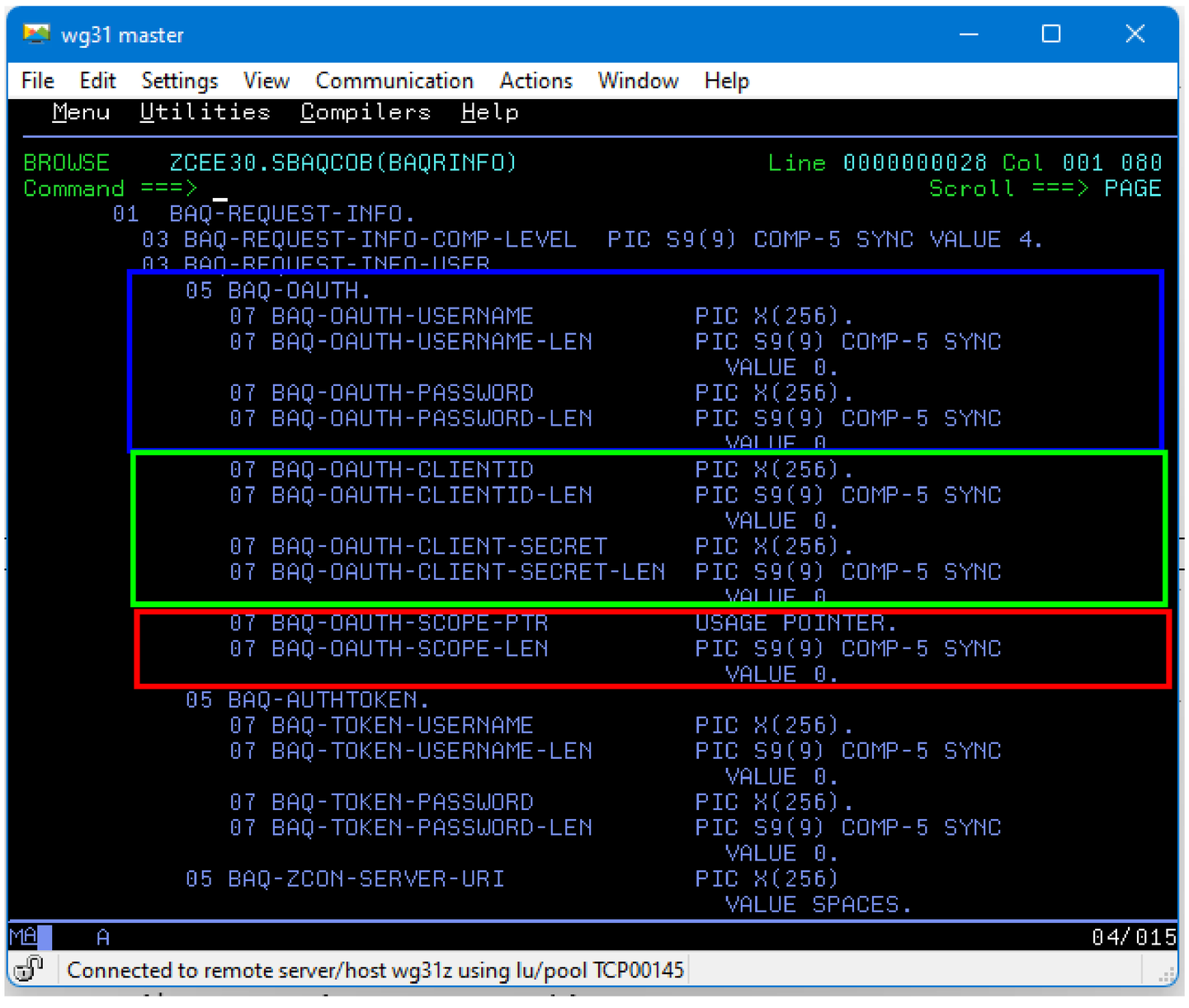
| requireAuth | idAssertion | Actions performed by z/OS Connect |
|---|---|---|
| true | OFF | Identity assertion is disabled. The zCEE server authenticates *zCEEID* and checks whether *zCEEID* has the authority to invoke an API requester. |
| | ASSERT_SURROGATE | Identity assertion is enabled. The zCEE 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_zosConnectManager
            requireAuth="true|false"
            requireSecure="true|false"/>

<zosconnect_apiRequesters idAssertion="OFF">

<zosconnect_apiRequester name="cscvinc_1.0.0"
            requireAuth="true|false"
            requireSecure="true|false"/>
            idAssertion="ASSERT_ONLY"> *

<zosconnect_apiRequester name="db2employee_1.0.0“
            requireAuth="true|false"
            requireSecure="true|false"/>
            idAssertion="ASSERT_SURROGATE"> *

</zosconnect_apiRequesters>
```

* Added in V3.0.45

mitchj@us.ibm.com

# Configuring OAuth support – BAQRINFO copy book




**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.**

**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**

**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

# Obtaining a JWT using request parameters

wg31 master

File Edit Settings View Communication Actions Window Help

```
 Menu  Utilities  Compilers  Help
BROWSE    ZCEE30.SBAQCOB(BAQHCONC)                Line 0000000020 Col 001 080
Command ===>                                        Scroll ===> PAGE
      * Host API Request parameter names
       77 BAQR-OAUTH-USERNAME      PIC X(22)
           VALUE 'BAQHAPI-oAuth-Username'.
       77 BAQR-OAUTH-PASSWORD      PIC X(22)
           VALUE 'BAQHAPI-oAuth-Password'.
       77 BAQR-OAUTH-SCOPE         PIC X(19)
           VALUE 'BAQHAPI-oAuth-Scope'.
       77 BAQR-OAUTH-CLIENT-ID     PIC X(22)
           VALUE 'BAQHAPI-oAuth-ClientId'.
       77 BAQR-OAUTH-CLIENT-SECRET PIC X(26)
           VALUE 'BAQHAPI-oAuth-ClientSecret'.
       77 BAQR-OAUTH-RESOURCE      PIC X(22)
           VALUE 'BAQHAPI-oAuth-Resource'.
       77 BAQR-OAUTH-AUDIENCE      PIC X(22)
           VALUE 'BAQHAPI-oAuth-Audience'.
       77 BAQR-OAUTH-CUSTOM-PARMS  PIC X(25)
           VALUE 'BAQHAPI-oAuth-CustomParms'.
       77 BAQR-JWT-USERNAME        PIC X(22)
           VALUE 'BAQHAPI-Token-Username'.
       77 BAQR-JWT-PASSWORD        PIC X(22)
           VALUE 'BAQHAPI-Token-Password'.

      * Host API ZCON parameter names
       77 BAQZ-TRACE-VERBOSE       PIC X(21)
           VALUE 'BAQHAPI-Trace-Verbose'.
       77 BAQZ-SERVER-URIMAP       PIC X(21)
           VALUE 'BAQHAPI-Server-URIMAP'.
       77 BAQZ-SERVER-HOST         PIC X(19)
MA     A
```

Connected to remote server/host wg31z using lu/pool TCP00111 and port 23

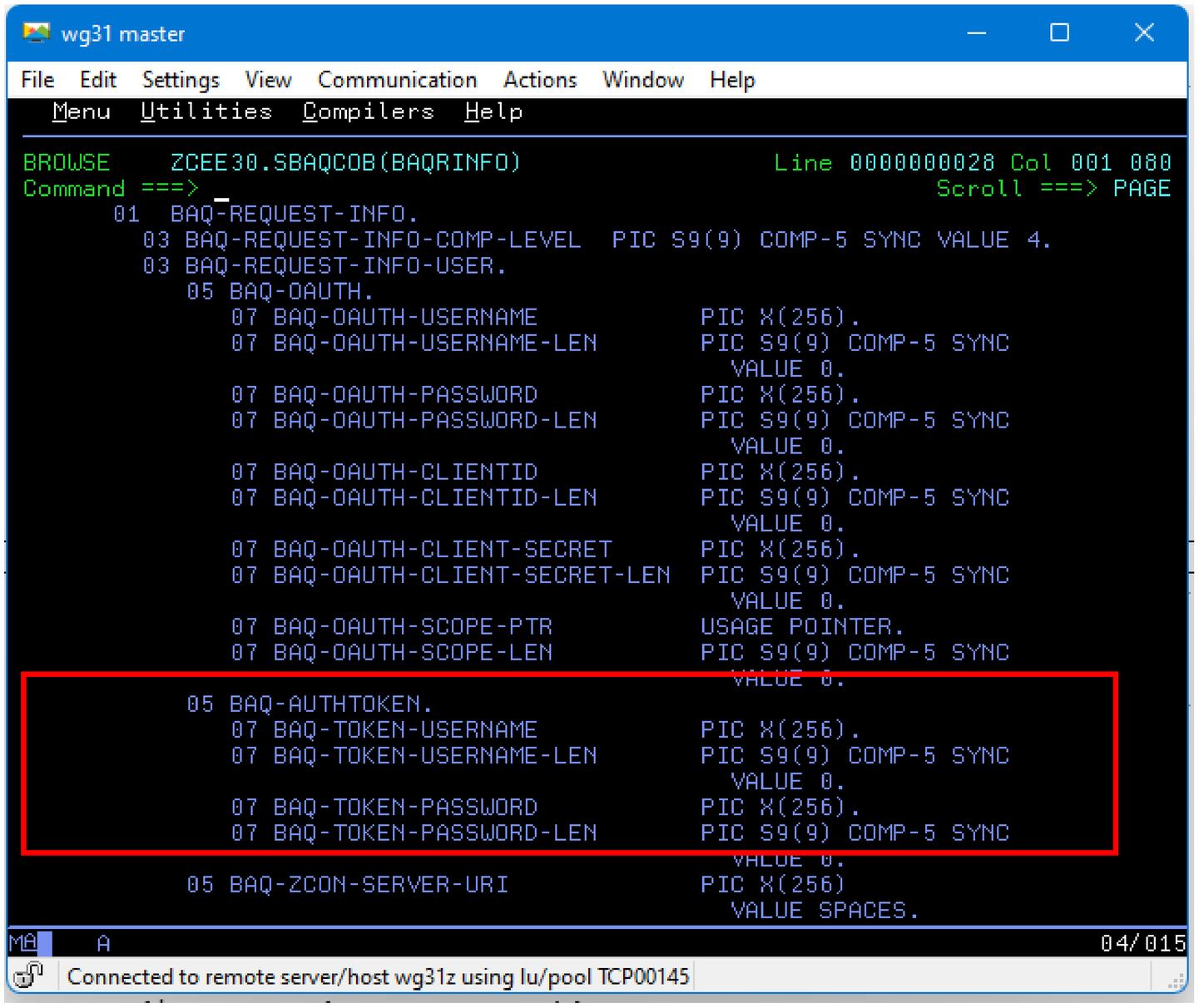
wg31 master

File Edit Settings View Communication Actions Window Help

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       USER1.ZCEE30.SOURCE(GETAPI) - 01.02          Columns 00001 00072
Command ===>                                             Scroll ===> PAGE
****** ******************************** Top of Data ********************************
000001          CBL APOST
000002
000003             * Authentication server credentials
000004          01 JWT-USER PIC X(10) VALUE 'myUsername'.
000005          01 JWT-PSWD PIC X(10) VALUE 'myPassword'.
000006
000007
000008
000009             *  Send JWT credentials to z/OS Connect
000010               MOVE BAQR-TOKEN-USERNAME TO
000011                    BAQ-REQ-PARM-NAME OF BAQ-REQ-PARMS(1)
000012               SET BAQ-REQ-PARM-ADDRESS OF
000013                   BAQ-REQ-PARMS(1) TO ADDRESS OF JWT-USER
000014               MOVE LENGTH OF JWT-USER TO
000015                  BAQ-REQ-PARM-LENGTH OF BAQ-REQ-PARMS(1)
000016               MOVE BAQR-TOKEN-PASSWORD TO
000017                    BAQ-REQ-PARM-NAME OF BAQ-REQ-PARMS(2)
000018               SET BAQ-REQ-PARM-ADDRESS OF
000019                   BAQ-REQ-PARMS(2) TO ADDRESS OF JWT-PSWD
000020               MOVE LENGTH OF JWT-PSWD TO
000021                   BAQ-REQ-PARM-LENGTH OF BAQ-REQ-PARMS(2)
000022
000023             * Call the API endpoint using BAQEXEC
000024
000025
000026
000027
MA     A                                                          28/019
```

Connected to remote server/host wg31z using lu/pool TCP00111 and port 23

# API Requester – JWT Custom flow




**BAQRINFO copy book**

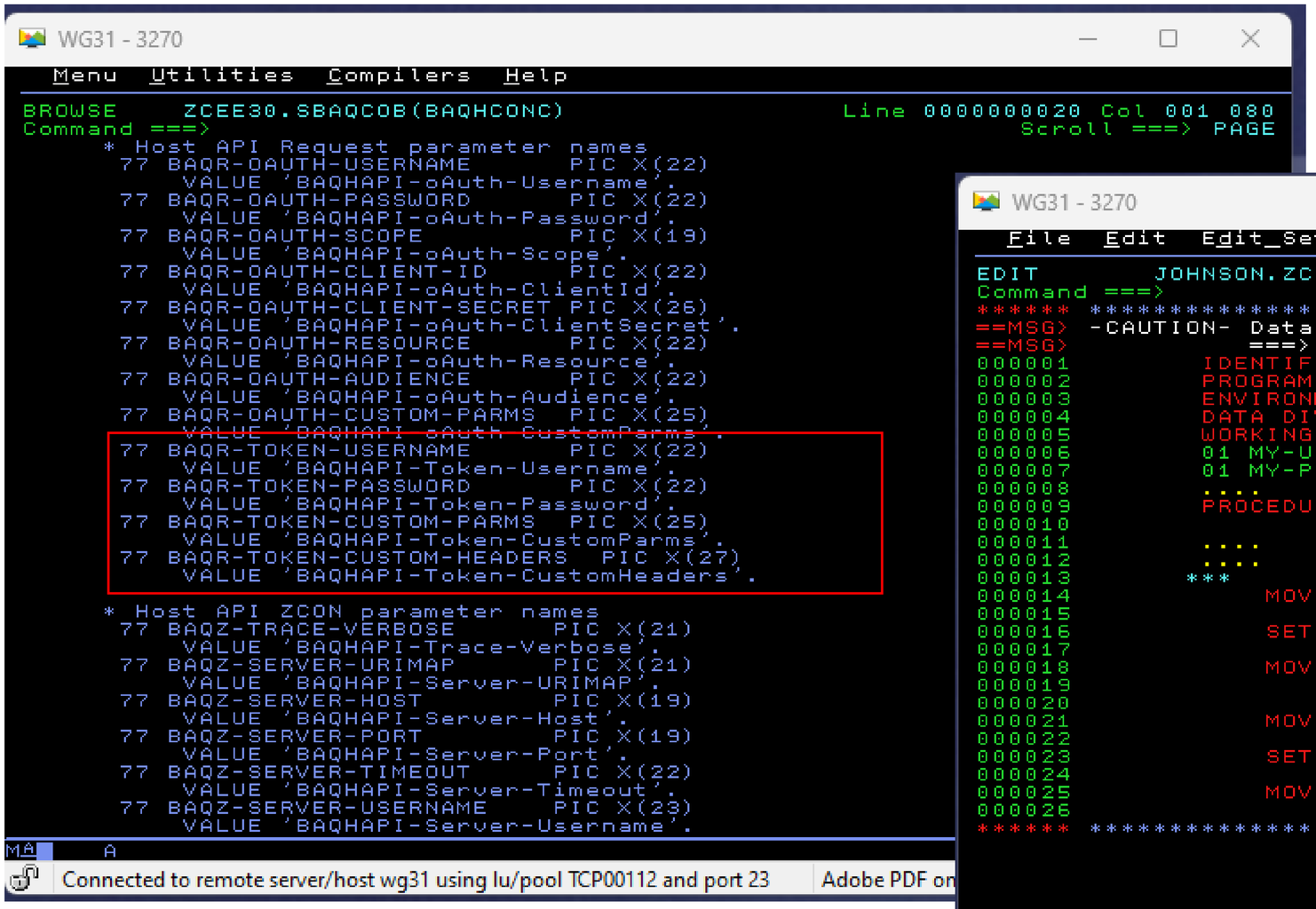
## COBOL application

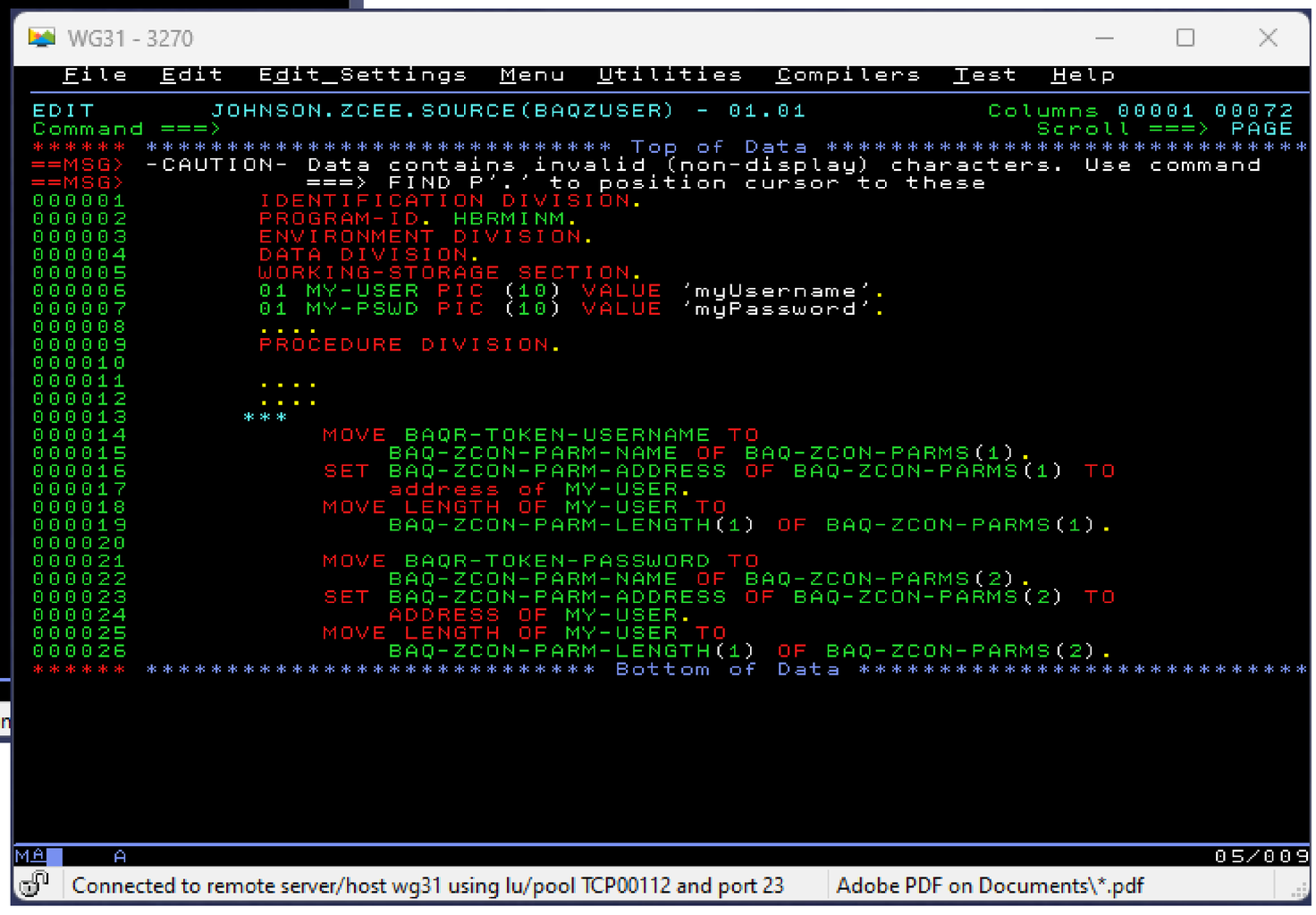
```
MOVE "ATSTOKENUSERNAME" to envVariableName.
PERFORM CALL-CEEENV THRU CALL-CEEENV-END
MOVE VAR(1:valueLength) to BAQ-TOKEN-USERNAME
MOVE valueLength TO BAQ-TOKEN-USERNAME-LEN
MOVE "ATSTOKENPASSWORD" to envVariableName.
PERFORM CALL-CEEENV THRU CALL-CEEENV-END
MOVE VAR(1:valueLength) to BAQ-TOKEN-PASSWORD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

# API Requester – JWT Custom flow



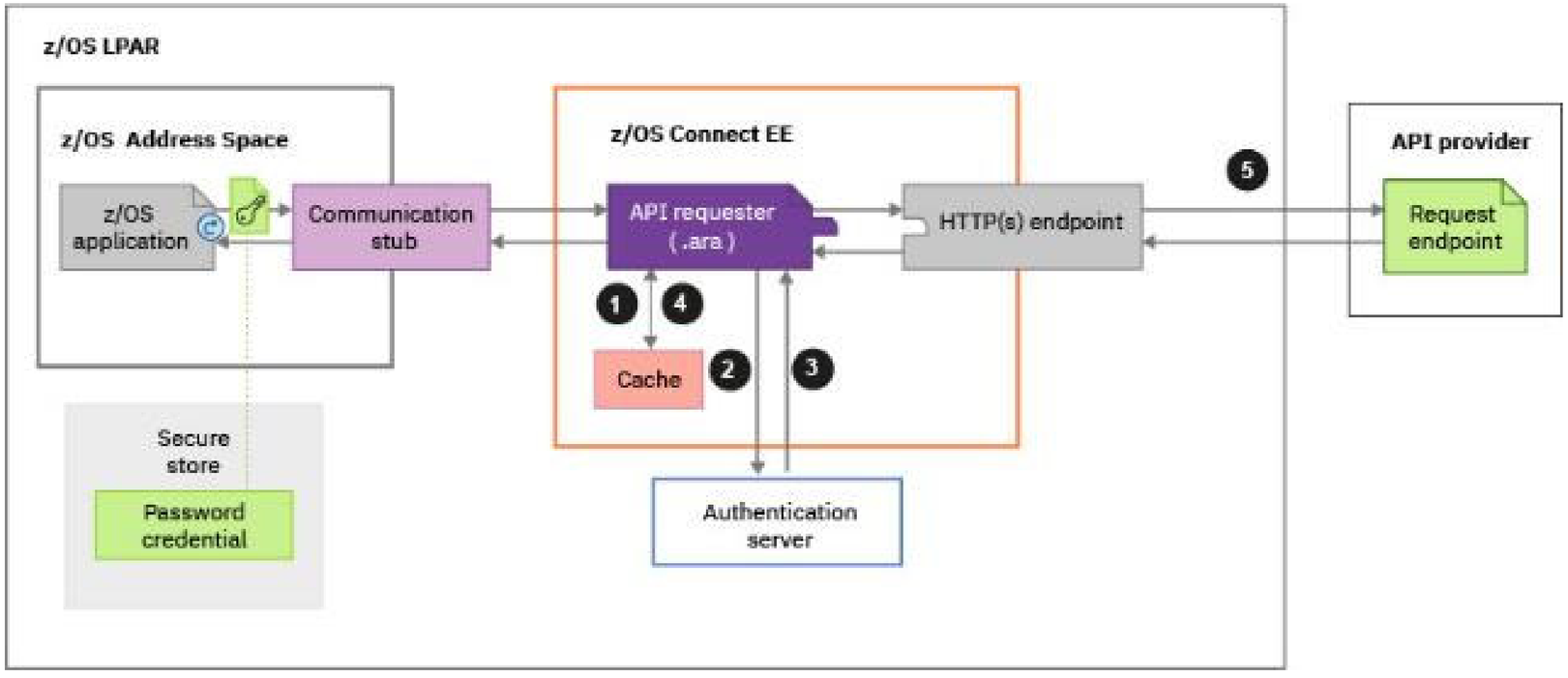





mitchj@us.ibm.com

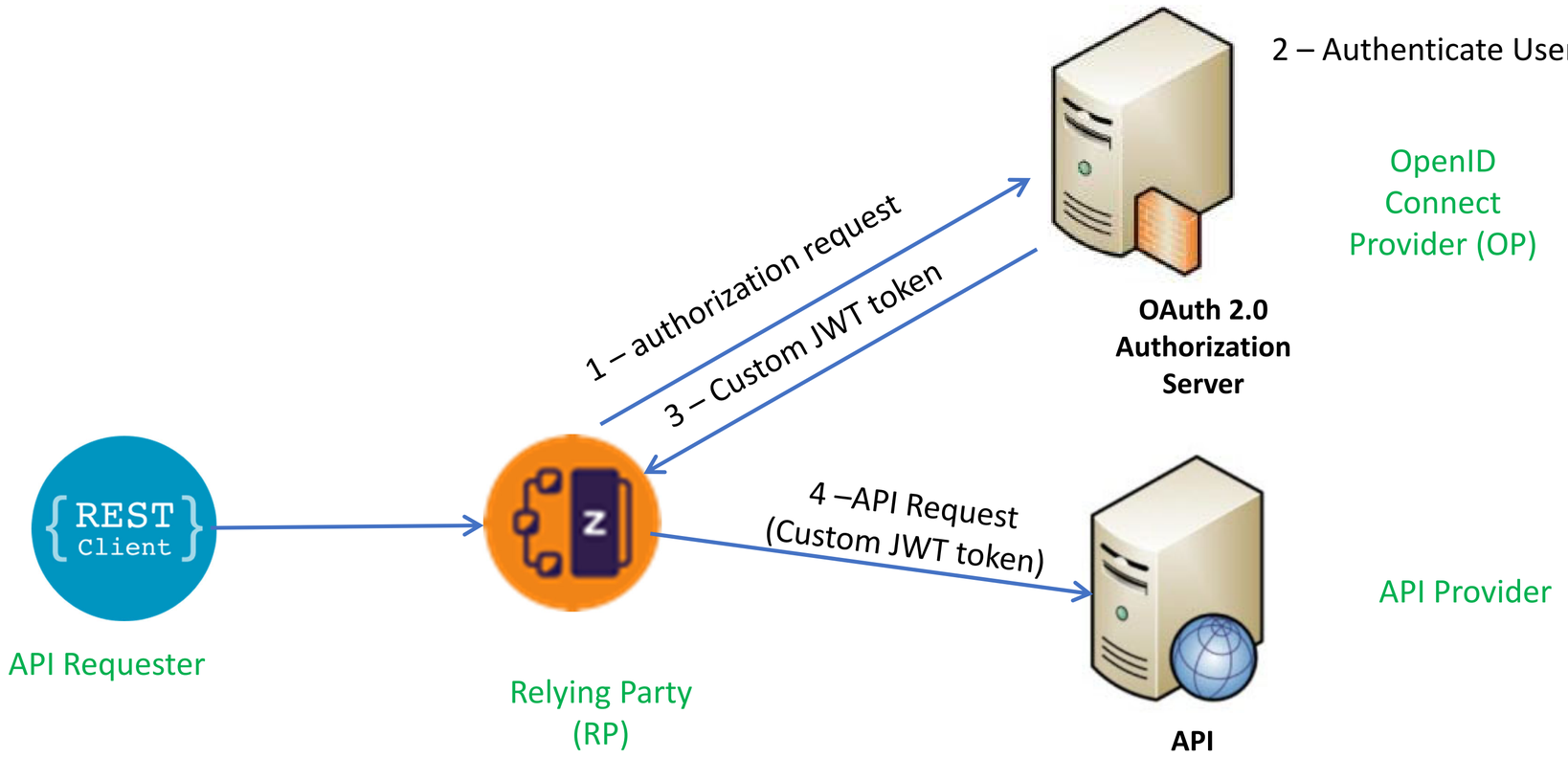
# Calling an API with using a JWT custom flow

- 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 Custom Flow




mitchj@us.ibm.com

# Configuring JWT Custom flow

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

<zosconnect_authToken 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

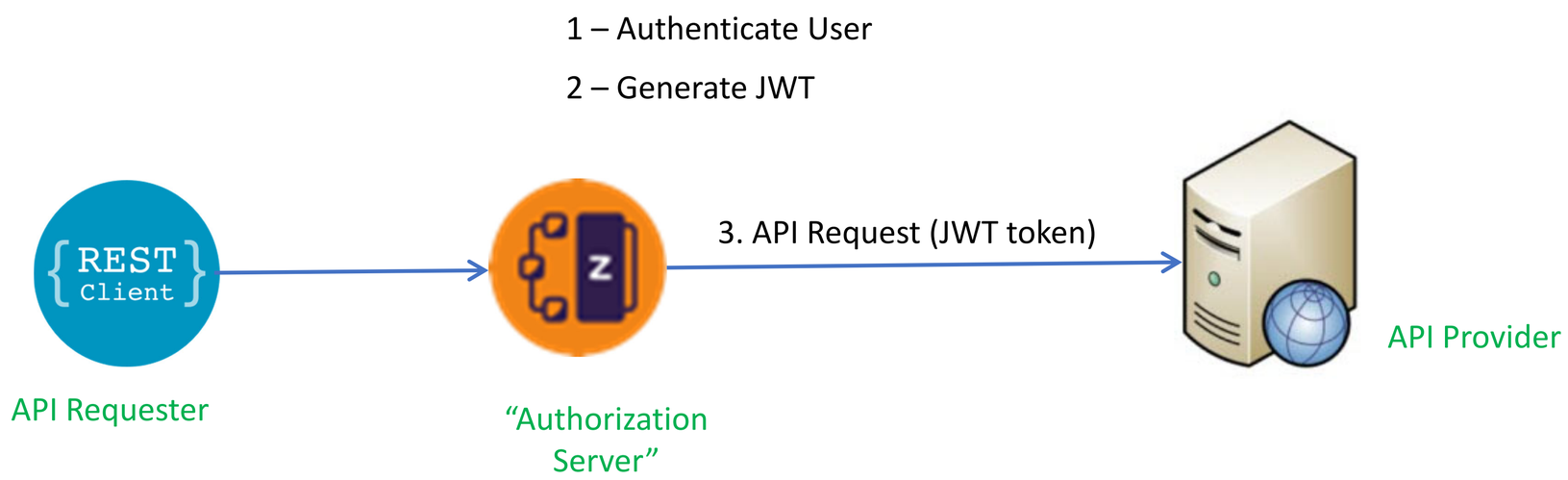
```
<tokenRequest credentialLocation="body"
    requestMethod="POST"
   // Use XML escaped characters in requestBody
   requestBody="{&amp;quot;apiuser&amp;quot;:&amp;quot;${userid}&amp;quot;,&amp;quot;apipassword&amp;quot;:&amp;quot;${password}&amp;quot;}" />
```

Response Token

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

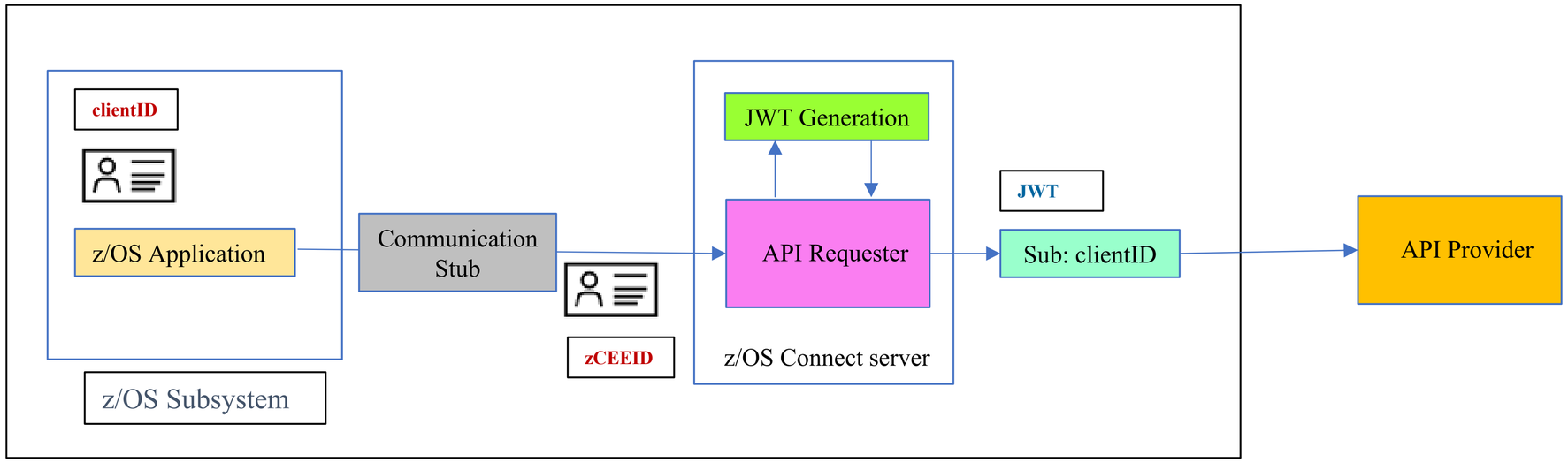
mitchj@us.ibm.com

# z/OS Connect JWT Generation – V3.0.43




mitchj@us.ibm.com

# API Requester – JWT Generation



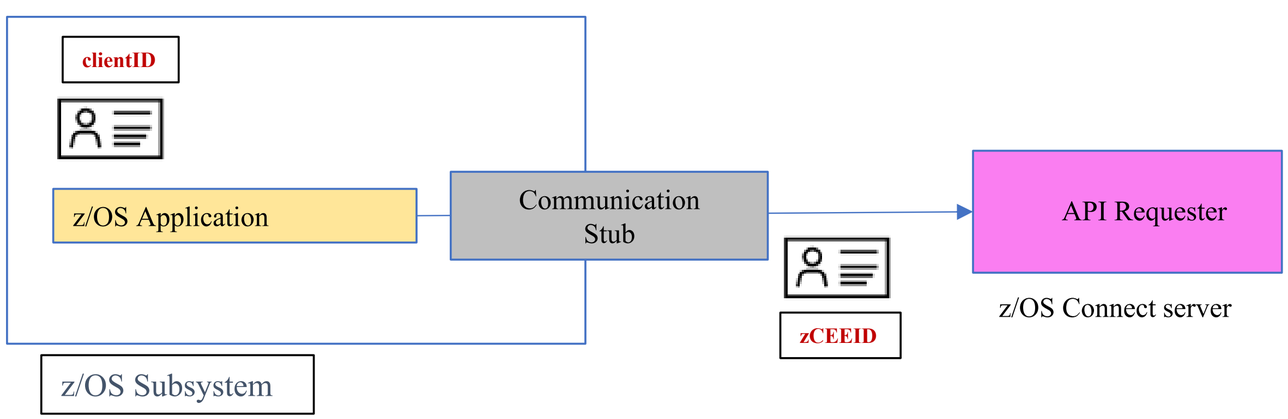

*zCEEID* – The identity that is used for authenticating connectivity the z/OS subsystem to the zCEE server. It is configured using basic authentication or for CICS, TLS client authentication.
*clientID* – the identity under which the z/OS application is executing.

- 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 |

[1] mitchj@us.ibm.com

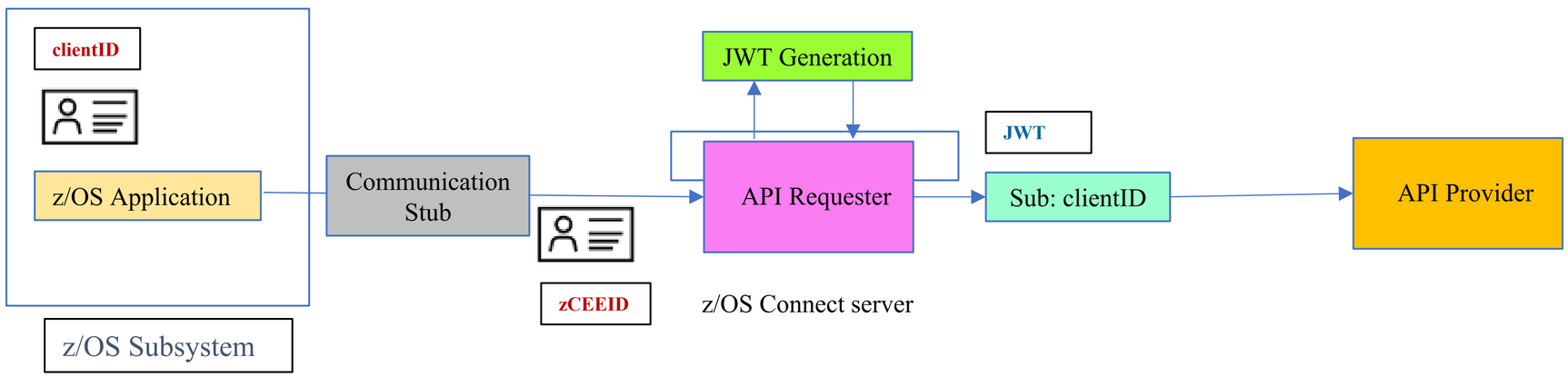
# API Requester - authentication with identity assertion and JWT generation




*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.

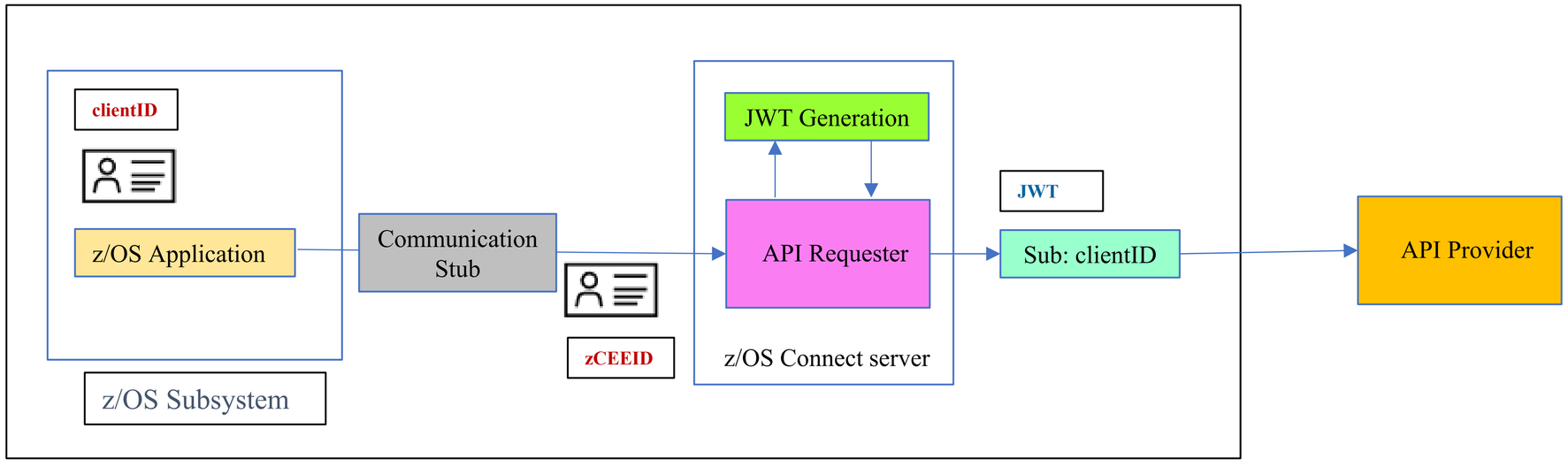
- For CICS, the task owner
- For IMS, the transaction owner
- For batch, the job owner




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

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

# 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

# server XML Configuration

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

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

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

Configure the Liberty jwtBuilder element in server.xml.

Configure the zosconnect_authTokenLocal element, specifying any additional private claims required and the name of the header used to send the JWT to the endpoint.

header default value is Authorization

Finally, reference the JWT configuration from the zosconnect_endpointConnection element.

mitchj@us.ibm.com