



# WebSphere Liberty Profile on z/OS

## Introduction to Liberty Security

Mitch Johnson  
[mitchj@us.ibm.com](mailto: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. . .



## WebSphere Liberty Profile on z/OS

Configuration/Management Best Practices

With a focus on Liberty servers for CICS, MQ, z/OSMF and z/OS Connect

Mitch Johnson  
[mitchj@us.ibm.com](mailto:mitchj@us.ibm.com)  
Washington Systems Center

[mitchj@us.ibm.com](mailto:mitchj@us.ibm.com)

<https://ibm.biz/BdahrR>



## WebSphere Liberty Profile on z/OS

Introduction to Liberty Security



© 2017, 2025 IBM Corporation  
Slide 1

[jauni](mailto:jauni)



## WebSphere Liberty Profile on z/OS

Managing, Monitoring and Problem Determination

Mitch Johnson  
[mitchj@us.ibm.com](mailto:mitchj@us.ibm.com)  
Washington Systems Center

[mitchj@us.ibm.com](mailto:mitchj@us.ibm.com)

<https://ibm.biz/BdahrR>



24 IBM Corporation  
Slide 1

[mitchj@us.ibm.com](mailto:mitchj@us.ibm.com)

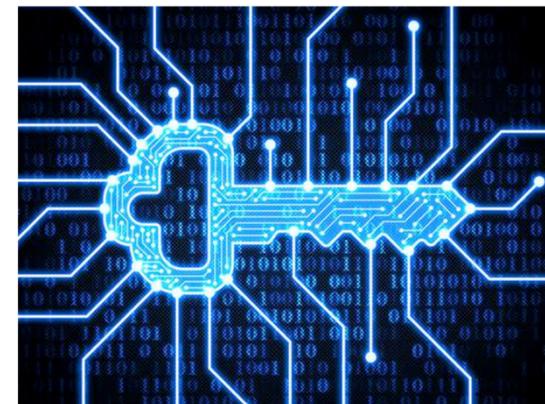
<https://www.ibm.com/support/pages/mainframe-system-education-wildfire-workshops>

© 2018, 2024 IBM Corporation  
Page 3

# 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.



## 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.

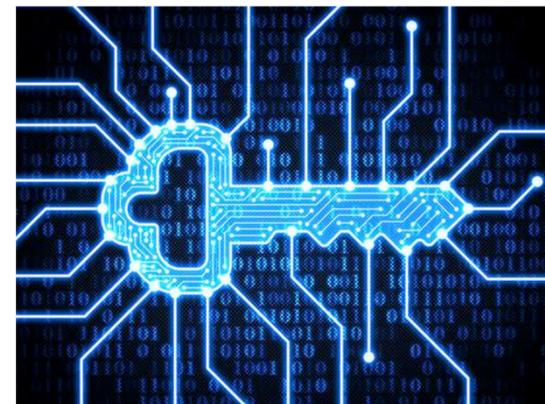


# Client Security Options

# 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.

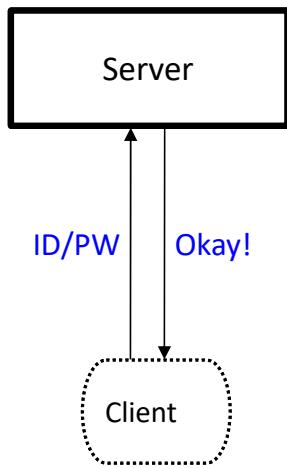




# Liberty Client Authentication Options

Several different ways this can be accomplished:

## Basic Authentication

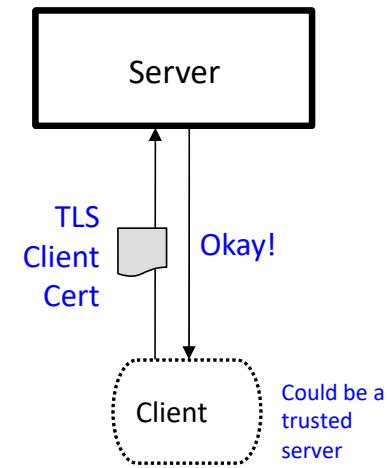


**Client supplies ID/PW or ID/PassTicket**

**Server checks registry:**

- Basic (server.xml)
- SAF

## Client Digital Certificate



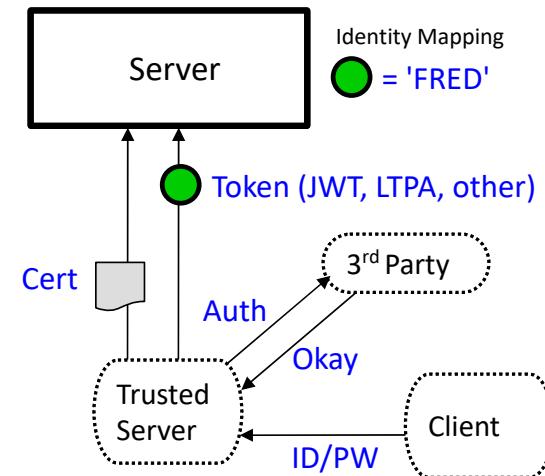
**Client supplies client personal certificate**

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

**Registry options:**

- SAF

## Third Party Authentication



**Client authenticates to 3<sup>rd</sup> party sever**

**Client receives a trusted 3<sup>rd</sup> party token**

**Token flows to server and is mapped to an identity**

**Registry options:**

- We may not need to know these details.



## 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.

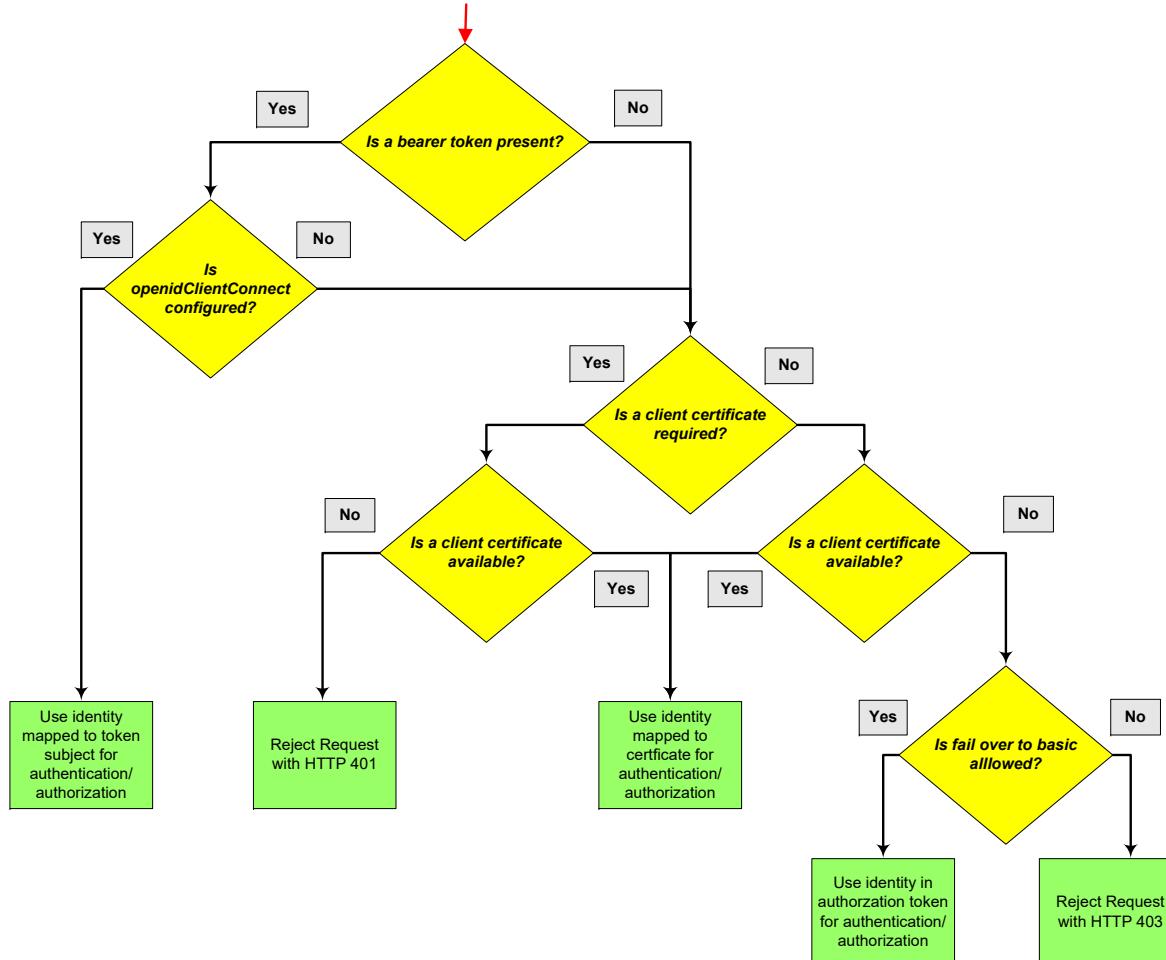
Requires that users specify security credentials to be authenticated in order to access all API requesters. If the requireAuth attribute is not set, the global setting on the zosconnect\_zosConnectManager element is used instead, unless the requireAuth attribute is overridden on the specific API requester.

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.



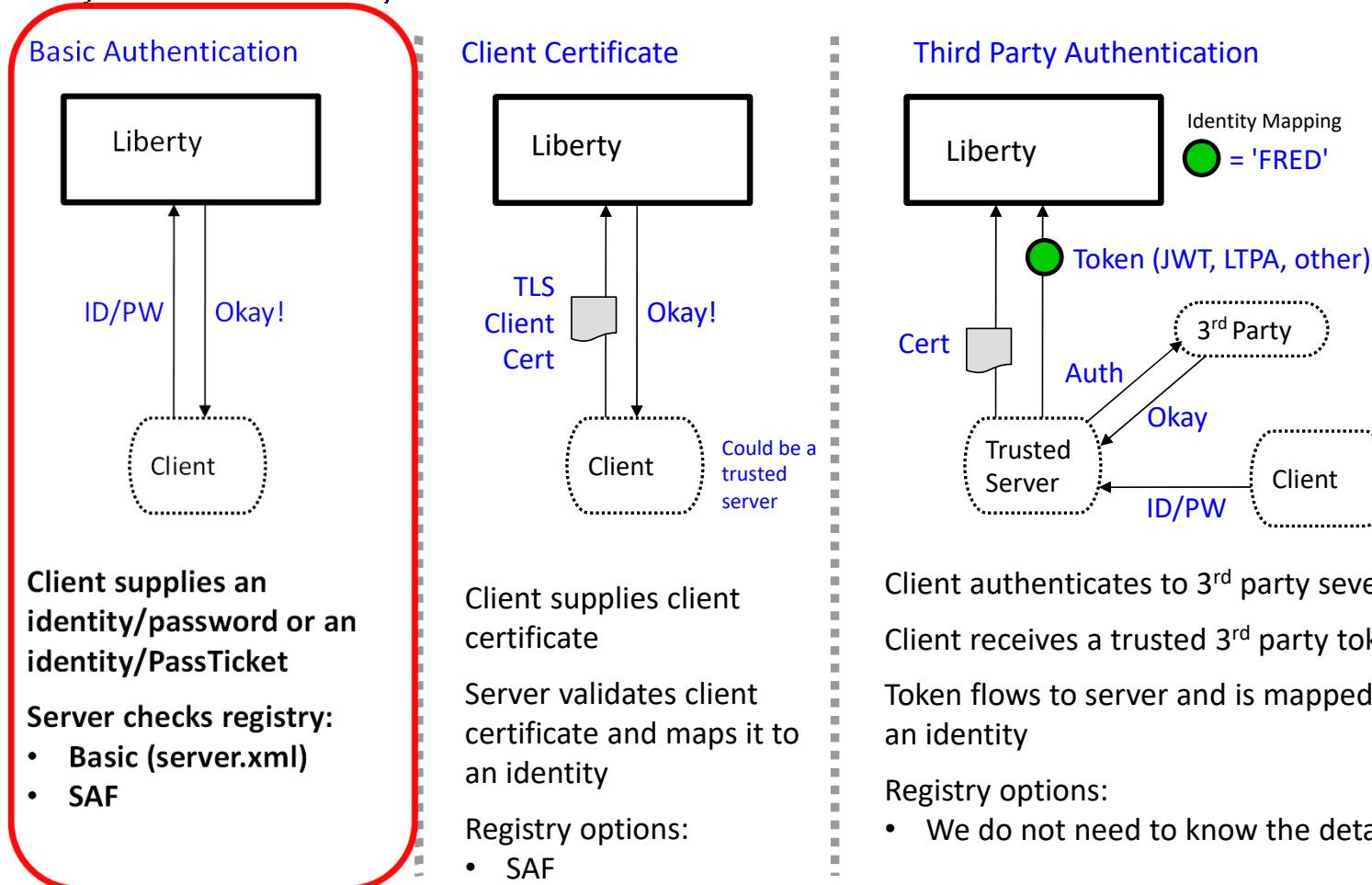
# Precedence order for determining an authorization identity





# Authentication - Basic Authentication

Several different ways this can be accomplished:





# 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** token with the identity and password encoded or formatted using Base64.

- An example with Postman:

The screenshot shows the Postman interface for a GET request to <https://mpz3.washington.ibm.com:9443/cscvinc/employee/111111...>. The 'Auth' tab is selected. Under 'Type', 'Basic Auth' is chosen. A note says: 'The authorization header will be automatically generated when you send the request.' Below it, 'Username' is set to 'Fred' and 'Password' is set to '....'. There is a 'Show Password' checkbox.

The screenshot shows the 'Headers' tab in Postman with 8 items. The 'Authorization' header is circled in red. Its value is listed as 'Basic RnJIZDpmcmVk'. Other headers shown include Postman-Token, Host, User-Agent, Accept, Accept-Encoding, and Connection.

KEY	VALUE
Authorization	Basic RnJIZDpmcmVk
Postman-Token	<calculated when request is sent>
Host	<calculated when request is sent>
User-Agent	PostmanRuntime/7.29.0
Accept	*
Accept-Encoding	gzip, deflate, br
Connection	keep-alive



# 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* token with the identity and password encoded or formatted using Base64.
  - Examples using the API Explorer feature , cURL, and a Java client.

The screenshot shows the IBM API Explorer interface for the 'cscvinc' service. On the left, there's a list of operations: POST /cscvinc/employee, DELETE /cscvinc/employee/{employee}, and GET /cscvinc/employee/{employee}. The 'GET' operation is selected. In the center, there's a 'Response Class (Status 200)' section with a JSON example. Below it, a 'Response Content Type' dropdown is set to 'application/json'. A 'Parameters' section shows an 'Authorization' parameter with the value 'Basic QWxhZGRpbjpvcGVuIHNlc2FyZGVy'. To the right of this, there's a red box highlighting a sign-in dialog box. The dialog box has the URL 'mpz3.washington.ibm.com:9443' at the top, followed by the message 'This site is asking you to sign in.' It has two input fields: 'Username' with 'user1' and 'Password' with '\*\*\*\*\*'. At the bottom are 'Sign in' and 'Cancel' buttons. A large red circle highlights the 'Authorization' parameter in the 'Parameters' section.

```
c:\>curl -X GET --user FRED:FRED --insecure https://mpz3.washington.ibm.com:9443/cscvinc/employee/111111
{ "cscvincSelectServiceOperationResponse": { "cscvincContainer": { "response": { "CE1BRESP": 0, "CE1BRESP2": 0, "USERID": "CICUSER", "filea": { "employeeNumber": "111111", "name": "C. BAKER", "address": "OTTAWA, ONTARIO", "phoneNumber": "51212003", "date": "26 11 81", "amount": "$0011.00" } } } }}
```

```
URL url = new URL("https://vg31.washington.ibm.com:9453/db2/department?dept1=C01&dept2=C01");
System.out.println("URL: " + url);
HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
conn.setRequestMethod("GET");
conn.setRequestProperty("Content-Type", "application/json");
byte[] bytesEncoded = Base64.encodeBase64("Fred:fredpwd".getBytes());
conn.addRequestProperty("Authorization", new String(bytesEncoded));

try {
    if (conn.getResponseCode() != 200) {
        throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());
    }
    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader((conn.getInputStream())));
    String output;
    StringBuilder stringBuffer = new StringBuilder();
    while ((output = bufferedReader.readLine()) != null) {
        stringBuffer.append(output);
    }
    JSONObject json = new JSONObject(stringBuffer.toString());
    JSONArray jsonArray = json.getJSONArray("ResultSet 1 Output");
    JSONObject jsonEntry = new JSONObject();
    for (int index = 0; index < jsonArray.length(); index++) {
        jsonEntry = jsonArray.getJSONObject(index);
        if (jsonEntry.has("employeeNumber")){
            ...
        }
    }
}
```

## Tech-Tip: A RACF PassTicket is an alternative to a password

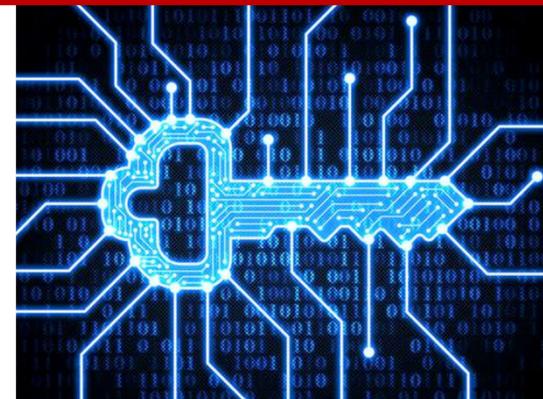
- ❑ A PassTicket is generated by or for a client by using a secured sign-on key (whose value is masked or encrypted) to encrypt a valid *RACF identity* combined with the *application name* of the targeted resource. Also embedded in the PassTicket is a time stamp (based on the current Universal Coordinated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

# 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.

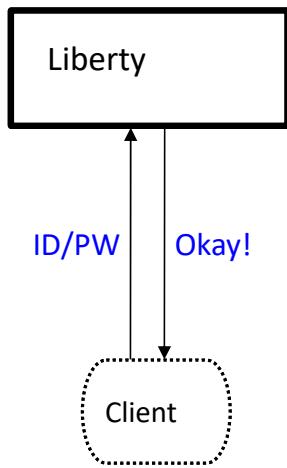




# Authentication - TLS Mutual Authentication

Several different ways this can be accomplished:

## Basic Authentication

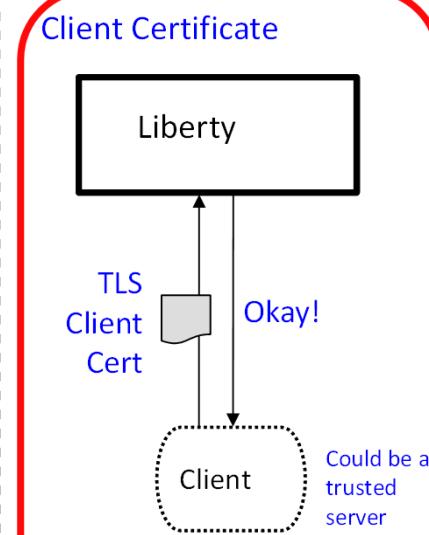


Server prompts for ID/PW

Client supplies ID/PW or ID/PassTicket

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

## Client Certificate

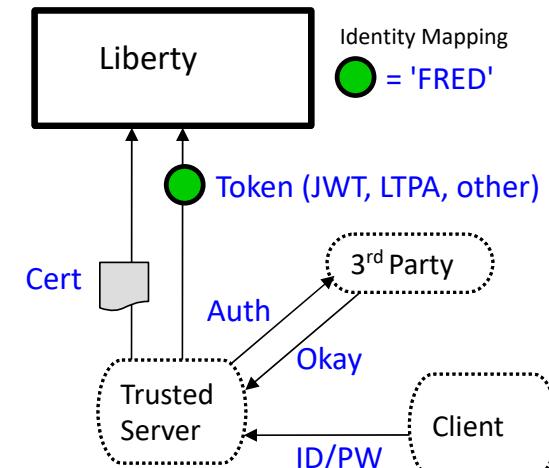


**Server prompts for client certificate information**

**Client supplies personal certificate information**

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

## Third Party Authentication



Client authenticates to 3<sup>rd</sup> party sever

Client receives a trusted 3<sup>rd</sup> party token

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

Registry options:

- We may not need to know these details.



# 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>

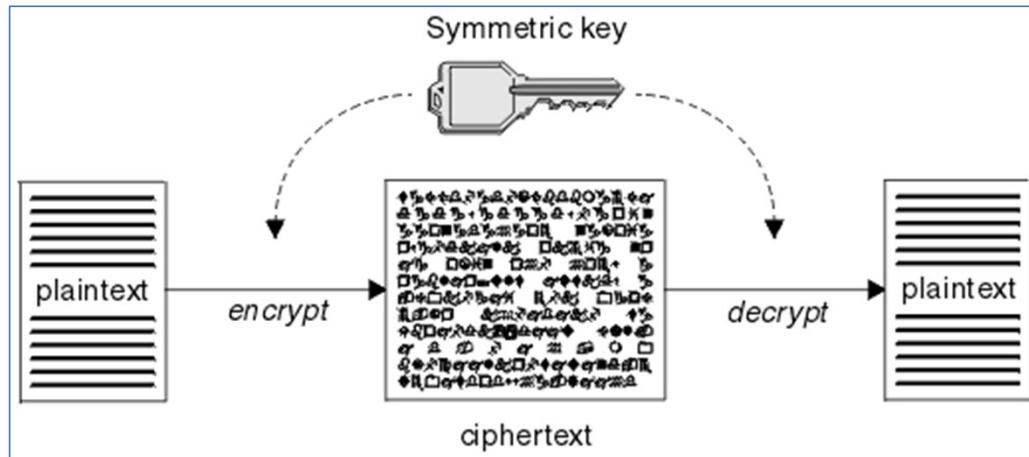
<zosconnect_services>
    <service id="selectByEmployee"
        name="selectEmployee"
        requireAuth="true"
        requireSecure="true|false"/>
</zosconnect_services>

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

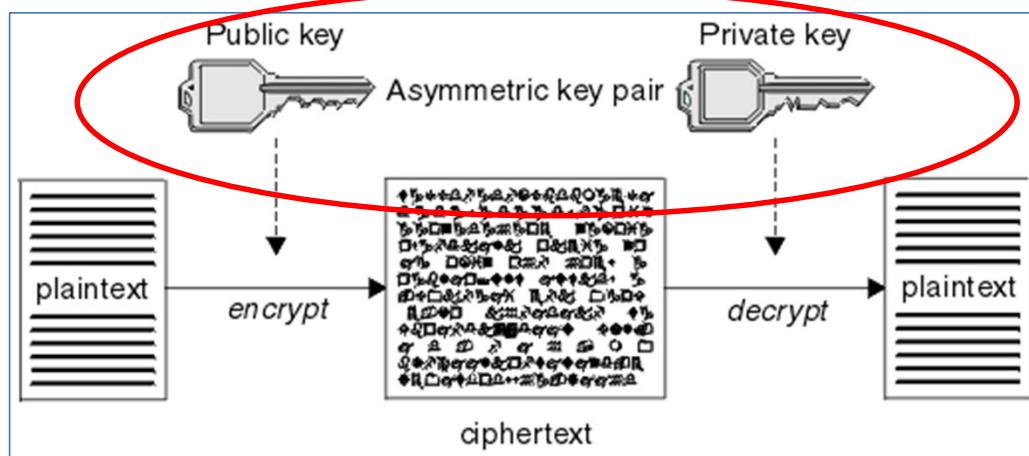
safkeyring:///KeyRing v safkeyring://owner/KeyRing

**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.

## 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.

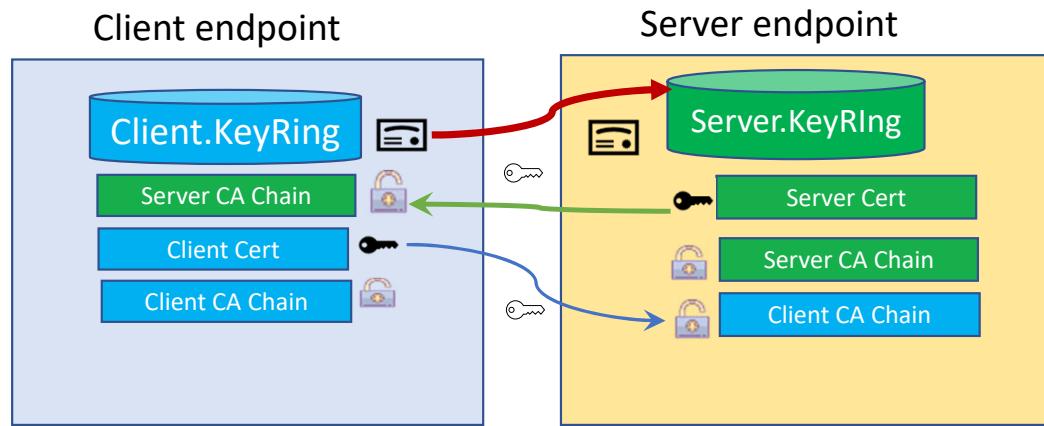
If an endpoint can successfully decrypt a message message encrypted received with a private key, the endpoint sending the message has successfully asserted its validity by proving it has the private used to encrypt the message.

# 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)

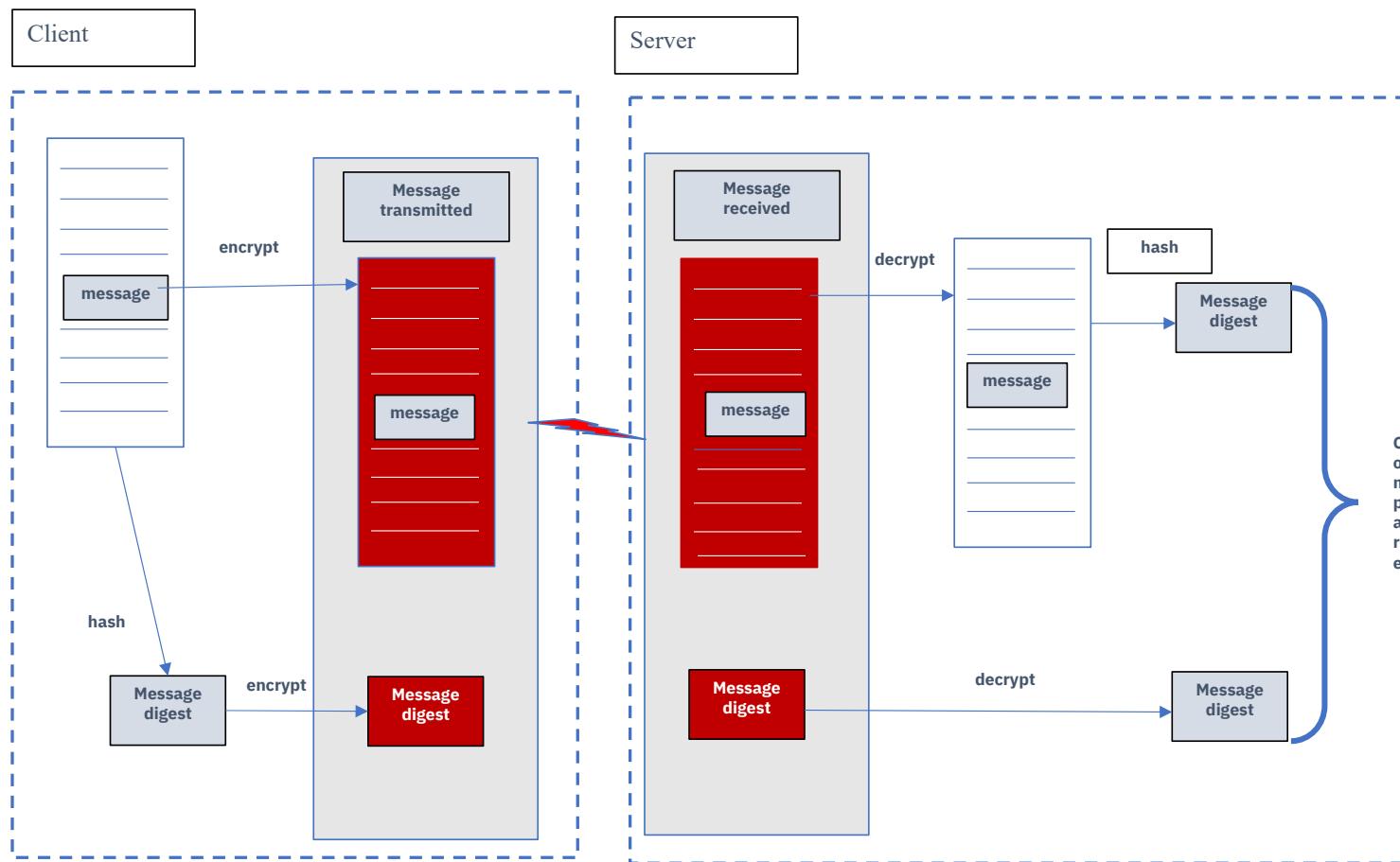


- 🔑 A certificate with a private key\* is required when an endpoint needs to prove its identity
- 🔑 Certificate's public key
- 🔒 Certificate Authority (CA) certificate chain<sup>#</sup>

\*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.

# 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.

# 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') ALTNAMES(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>

# 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

## 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.

## 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.

# 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 <a href="#">IBM SDK Policy files</a> topic in the <a href="#">Security Reference for IBM SDK, Java Technology Edition</a> 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

© 2017, 2022 IBM Corporation  
Slide 25

# 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 <a href="#">IBM SDK Policy files</a> topic in the <a href="#">Security Reference for IBM SDK, Java Technology Edition</a> 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

© 2017, 2022 IBM Corporation  
Slide 26

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



```
Digital certificate information for user ATSSERV:  
  
Label: RPServer-Server  
Certificate ID: 2QfB4+Lixdn12dfihZmlhZlg4oWZpYWZ  
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 resolve this issue.

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

**Key information for [fred]**

fred

<b>Key Size:</b>	2048
<b>Certificate Properties:</b>	
Version:	X509 V3
Serial Number:	02
Issued to:	CN=Fred D. Client, OU=LIBERTY, O=IBM
Issued by:	CN=CA for Liberty, OU=LIBERTY
Validity:	Valid from February 3, 2021 to December 31, 2022
Fingerprint:	SHA1: 5C:07:59:0C:FC:01:77:60:F1:E4:8C:3D:F7:DE:CB:1E:0C:F6:DB:D1 SHA256: 65:6A:28:B4:1B:D6:D0:FD:3D:6B:04:1E:C9:42:7E:7A:F3:0E:6B:E3:E0:58:15:40:C7:80:4D:3F:67:3B:CD:18 HPKP: bcbFMr+RjuMVhrc5xyKYZzNSWiZpTUzbk03MDE+A94=
Signature Algorithm:	SHA256withRSA (1.2.840.113549.1.1.11)
Subject Alternative Names:	Email Address: <input type="text"/>
IP Address:	<input type="text"/>
DNS Name:	<input type="text"/>

**View Details...**

**OK**

**Command Prompt**

```
c:\Users\workstation\.zosexplorer\.metadata\.plugins\com.ibm.cics.core.comm>keytool -list -keystore explorer_keystore.jks -alias fred -v -storepass changeit
Alias name: fred
Creation date: Mar 8, 2021
Entry type: keyEntry
Certificate chain length: 2
Certificate[1]:
Owner: CN=Fred D. Client, OU=LIBERTY, O=IBM
Issuer: CN=CA for Liberty
Serial number: 2
Valid from: 2/3/21 11:08:00 AM
Certificate fingerprints:
    MD5: 89:5B:87:
    SHA1: 5C:07:59:
    SHA256: 65:6A:2
Signature algor:
Version: 3

Extensions:
#1: ObjectId: 2.16.840.1
0000: 16 30 47 65 6e 65
0010: 68 65 20 53 65 63
0020: 65 72 20 66 6f 72
0030: 46 29

#2: ObjectId: 2.5.29.35
AuthorityKeyIdentifier [
0000: 6e a0 26 b5 6b d5
0010: a4 3a 2f e7
]
]

#3: ObjectId: 2.5.29.14
SubjectKeyIdentifier [
KeyIdentifier [

```

**Keychain Access**

Johnson, Edward M (Mitch)

Johnson, Edward M (Mitch)

Issued by: IBM CA  
Expires: Saturday, September 4, 2021 at 7:59:59 PM Eastern Daylight Time  
This certificate is valid  
Trust

**certmgr - [Certificates - Current User\Personal\Certificates]**

File Action View Help

Certificates - Current User

- Personal
  - Certificates
- Trusted Root Certification Authority
- Enterprise Trust
- Intermediate Certification Authority
- Active Directory User Object
- Trusted Publishers
- Untrusted Certificates
- Third-Party Root Certification Authority
- Trusted People
- Client Authentication Issuers
- Other People
- Local NonRemovable Certificate Store
- MSIEHistoryJournal
- Certificate Enrollment Requests
- Smart Card Trusted Roots

Issued To      Issued By

Edward M. Johnson	IBM Wireless Intermediate CA
CA for Liberty	CA for Liberty
mitchj	IBM Wireless PC CA
mitchj@us.ibm.com	MS-Organization-P2P-Access [20...
S-1-12-1-874953245-130398356...	S-1-12-1-874953245-130398356...

Personal store contains 15 certificates.

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

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

Public certificate (site or certificate authority or certificate request)

```
CERTAUTH.PEM - Notepad
-----BEGIN CERTIFICATE-----
MIIDWjCCAj6gAwIBAgIBADANBgkqhkiG9w0BAQsFADArMRAwDgYDVQQLEwdMSUJF
U1RZRMcwFQDVQDQEwSDQS8mb3IgTG1iZXJ0eTAEfW0yMDExMTUwNDAwMDBaFw0y
MzAxMDExMzU5NT1aMCsxEDA0BqNVBsATB0xJQkVSFkxZaVBgjNVBAMTDkNBIGZv
c1BMAwJ1cnRS5MIIBJAANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEASC7jIPa9
WuIPN1pCIA4twmgZM0R9naCitAnjFyN10RqbHZxKBueg+2mjE6axmRSmj7pI
kLSEK8fZ683avEQTZVTeSiow+9C0vMc6bCVok/F0FhzmTNrRmhk16jMXFeE8
RJwVxhtVgUY24J0AxG0oPI7cYoeBYaZ39t+hng+4Ip03JK111xSJUyRngJ0W2YT
ya8uJb57nhx30fHPVdpxa5h5dVmV9YtM/w6644/JBKMo03jbWwnQ9FlonCsdSc+
ob+au04NHV6dIC9z-/Dq95p1uXMYwPx8j1dhrzAIR3bnJFpy02xTTckREGfq27
gYl0YEcdb+biNQIDAQAB04GEMIGBM8D8GCWCGSAGG+EIBDQyFjBH2W51cmF0ZWQ
YnkgdGh1FN1y3VyaXR5F1n1cnZlci8mb3Ige19PUyAoUKFDRIkwDgYDVROPAQH/
BAQDAgEGMA8GA1UdEwEB/wFOFMABAf8wHQYDVROBBYEFJZVpVWPqeQWhYY2v1+
T1uzksHEMA0GCSqGSIb3DQEBCwUA41BAQQAEPHQs75ZyP8DWnyX5IQnLDTFU36bX
4Rgttx3XV1+H/jSh01Wfvuxp7iguD30v3+7dtFUR5Diup6pkmaE9v8fbqA6FD
N6EABN6vI8VQUAV9cAW95Pp0mpCwre0akJDeRywttxiX9aGUKE1Vgk1K4tHv/8
Rfw+Fb8pzQHAQPOTJ8s270eGlW0DtDwqfDb8SCfmJdLUQvixLezPdHc+CX3t+0
iJTSziLLcBtC6Ainycw/U4/0x19m0AvjKrgym14rfC1fj8G23hk6vxhJBxCRB
BD9wVZ9ZIuiA0NugkpzGR03HwcbyY0idkAF0ZkeVh0t1SMRzbG6q55b1
-----END CERTIFICATE-----
```

Personal certificate with private key

```
USER1.P12 - Notepad
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
```

```
WG31# - 3270
File Edit Settings View Communication Actions Window Help
Menu Utilities Compilers Help
Browse USER1.CERTAUTH.PEM Line 0000000000 Col .001 080
***** BEGIN CERTIFICATE ***** Top of Data *****
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
```

```
WG31# - 3270
File Edit Settings View Communication Actions Window Help
Menu Utilities Compilers Help
Browse USER1.CICS.P12 Line 0000000000 Col .001 080
Command ==>
***** BEGIN CERTIFICATE ***** Top of Data *****
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
```

# 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
```



# 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.



# Tech/Tip: Combining TLS mutual and basic authentication

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

```
<httpEndpoint id="defaultHttpEndpoint"  
host="*"  
httpPort="9080"  
httpsPort="9443" />  
  
<sslDefault sslRef="DefaultSSLSettings"  
outboundSSLRef="DefaultSSLSettings" />  
  
<ssl id="DefaultSSLSettings"  
keyStoreRef="CellDefaultKeyStore"  
trustStoreRef="CellDefaultKeyStore"  
clientAuthenticationSupported="true"  
clientAuthentication="true"/>  
  
<keyStore id="CellDefaultKeyStore"  
location="safkeyring:///Liberty.KeyRing"  
password="password" type="JCERACFKS"  
fileBased="false" readOnly="true" />
```

```
<httpEndpoint id="AdminHttpEndpoint"  
host="*"  
httpPort="-1"  
httpsPort="9445"  
sslOptionsRef="mySSLOptions"/>  
  
<ssLOptions id="mySSLOptions"  
sslRef="BatchSSLSettings" />  
  
<ssl id="BatchSSLSettings"  
keyStoreRef="CellDefaultKeyStore"  
trustStoreRef="CellDefaultKeyStore"  
clientAuthenticationSupported="true"  
clientAuthentication="false"/>
```

<https://www.rocketsoftware.com/products/rocket-open-source/rocket-open-appdev-z>



# 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" />

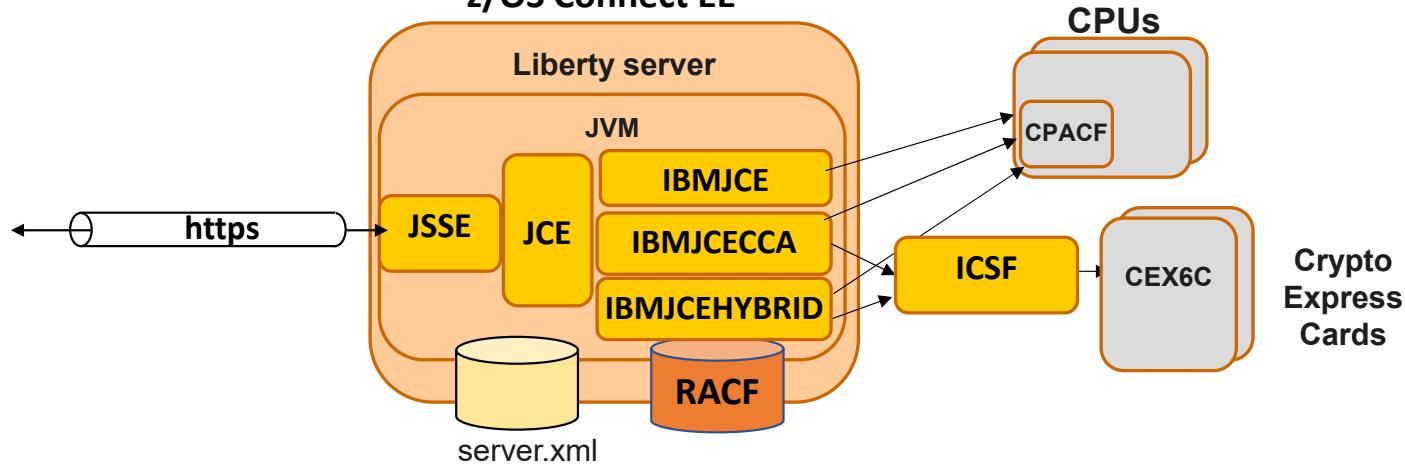
<zosconnect_authorizationServer sslCertsRef="SSL repertoire"/>
<zosconnect_cicsIpicConnection sslCertsRef="cicsTLSSettings"/>
<zosconnect_db2Connection sslCertsRef="db2TLSSettings" *>
<zosconnect_endpointConnect sslCertsRef="SSL repertoire"/>
<zosconnect_zosConnectRestClient sslCertsRef="SSL repertoire"/>
<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**



# Using JSSE with Liberty

The server XML configuration defines the HTTPS ports, key rings, and other JSSE attributes  
z/OS Connect EE

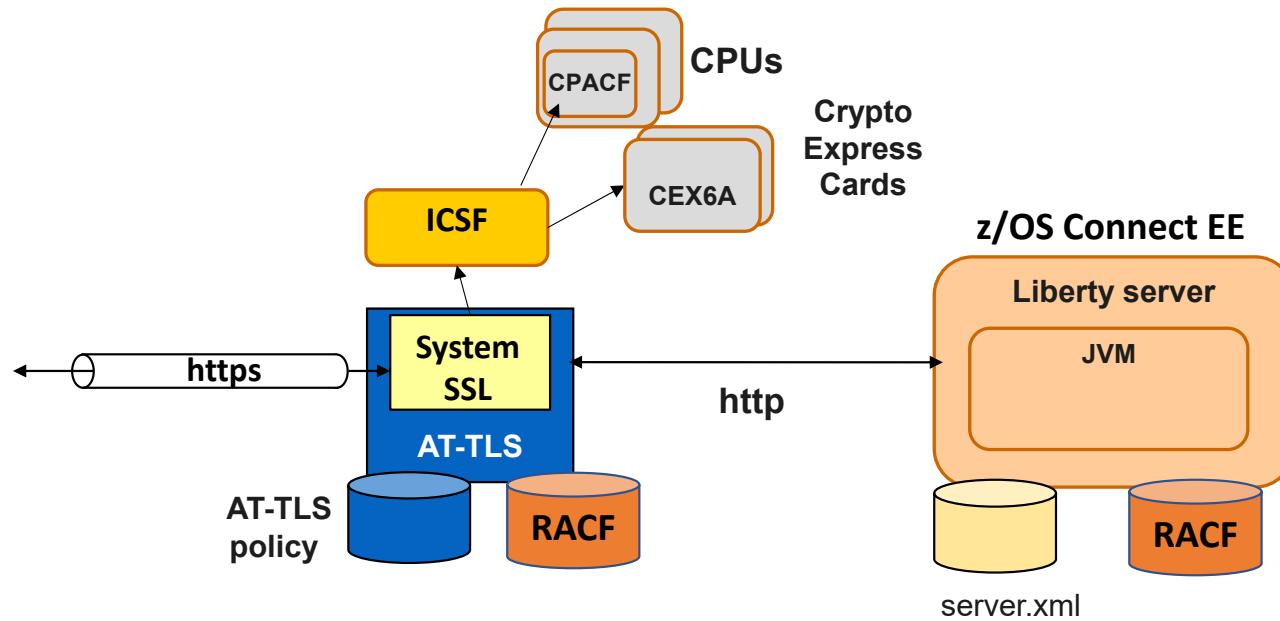


- z/OS Connect EE support for TLS is based on **Liberty** server support
- **Java Secure Socket Extension (JSSE)** API provides framework and Java implementation of TLS protocols used by Liberty HTTPS support
- **Java Cryptography Extension (JCE)** is standard extension to the Java Platform that provides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



# Using AT-TLS with Liberty

The server XML configuration uses no 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

**Let's explore using TLS for  
encryption and data integrity  
using samples in various scenarios**



## Using this Liberty JSSE server XML configuration



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

Liberty's outbound 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 CICSSSTC:			
Ring: >CICS.KeyRing<			
Certificate Label Name	Cert Owner	USAGE	DEFAULT
CICS CA	CERTAUTH	CERTAUTH	NO
CICS Client Cert	ID(CICSSSTC)	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

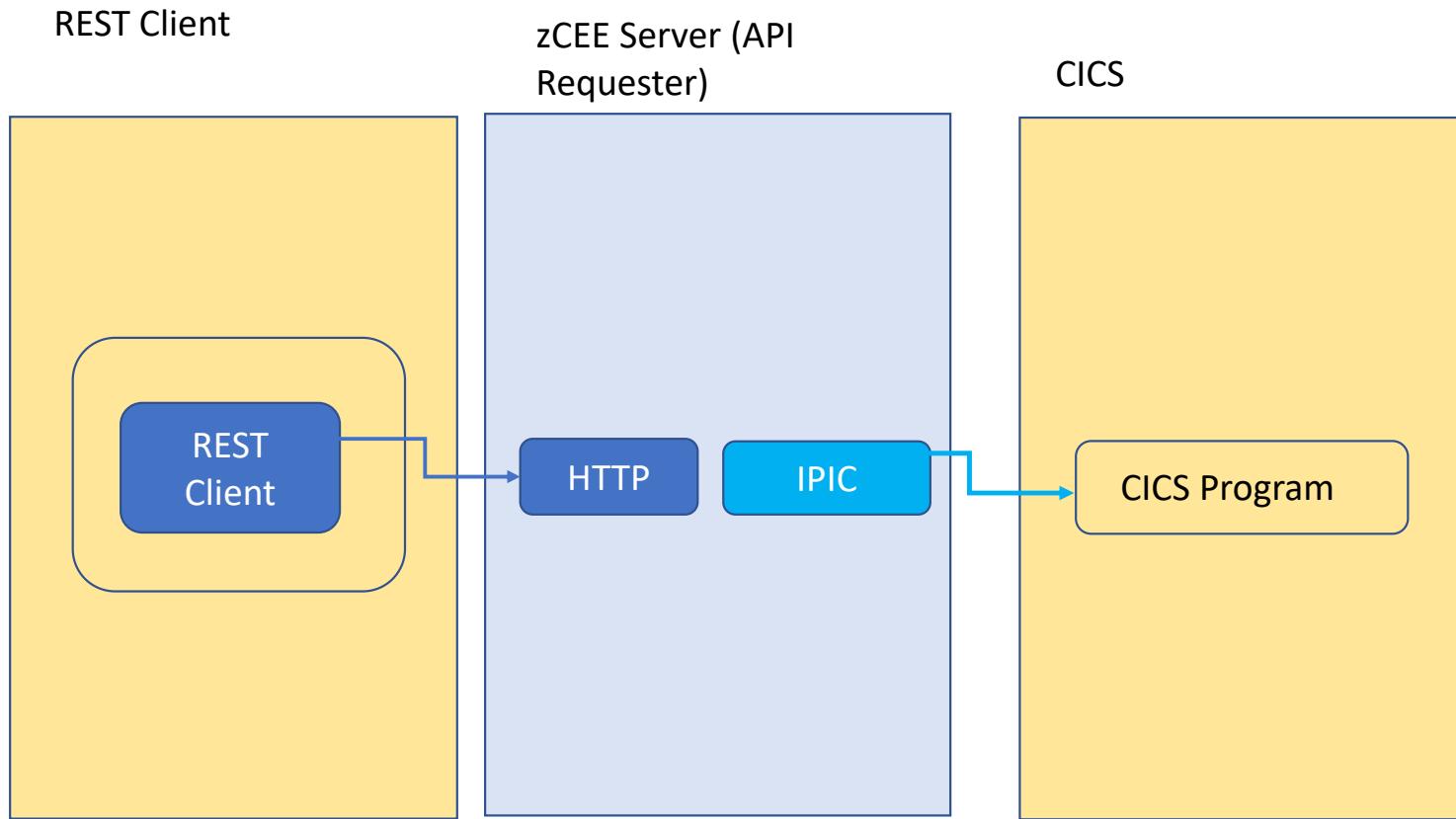
Tech-Tip: when Liberty is the client endpoint, and more than one personal certificate is connected to a key ring. Use the SSL repertoire *clientKeyAlias* attributes to select the personal certificate to be used in a handshake.

Liberty's inbound key ring

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.

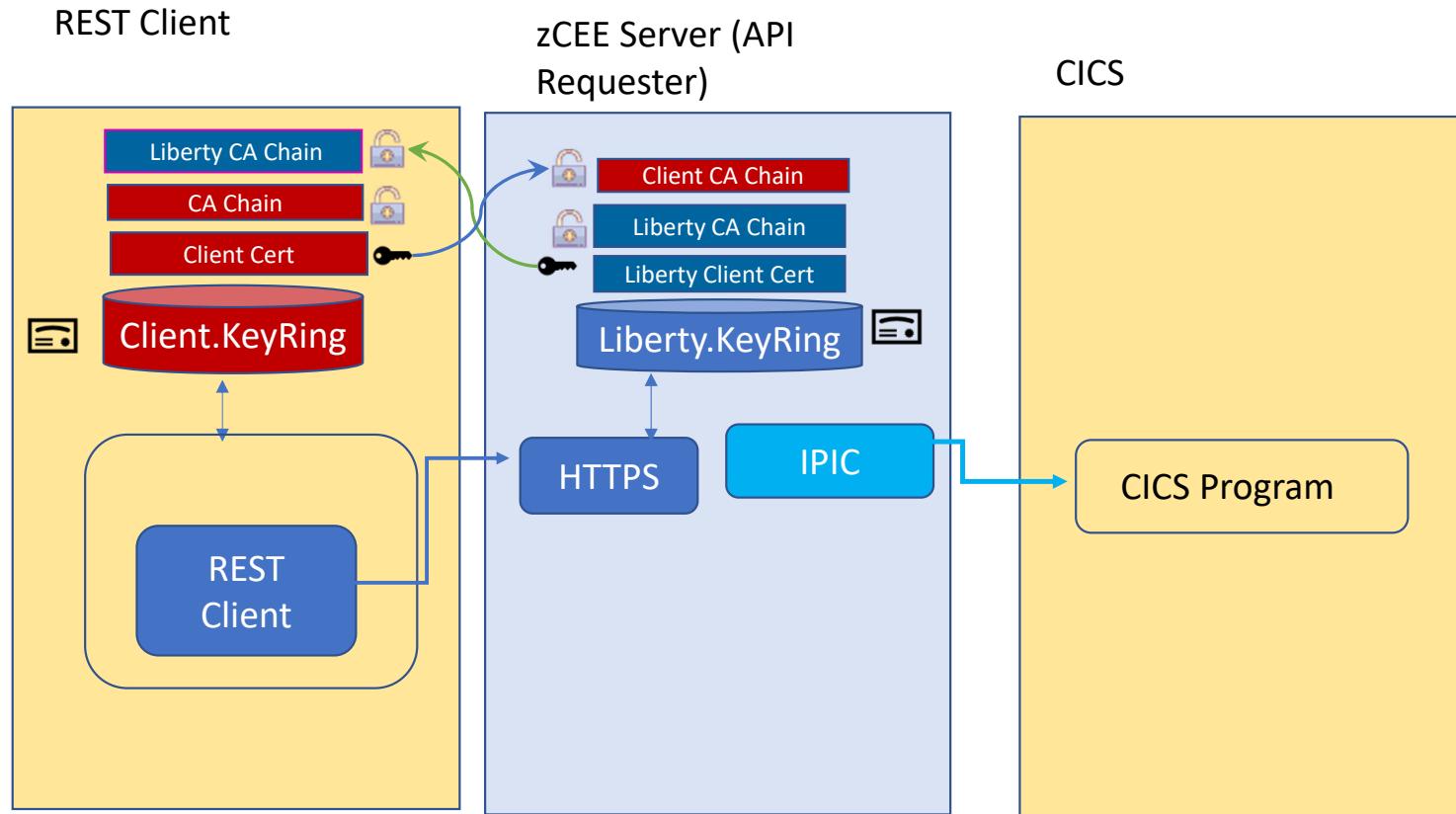


## No TLS between any endpoints



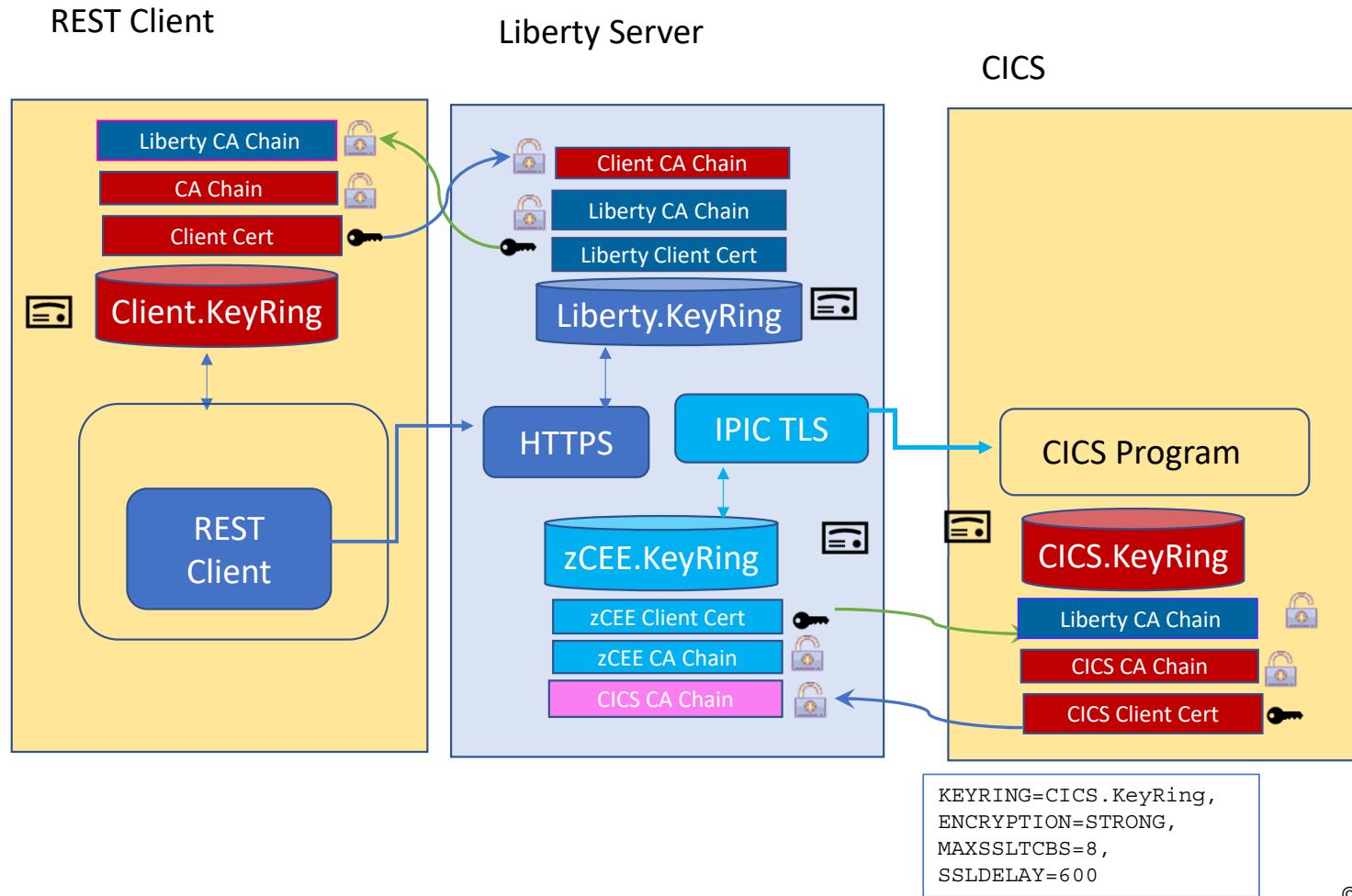


## TLS handshake between the client and z/OS Connect server



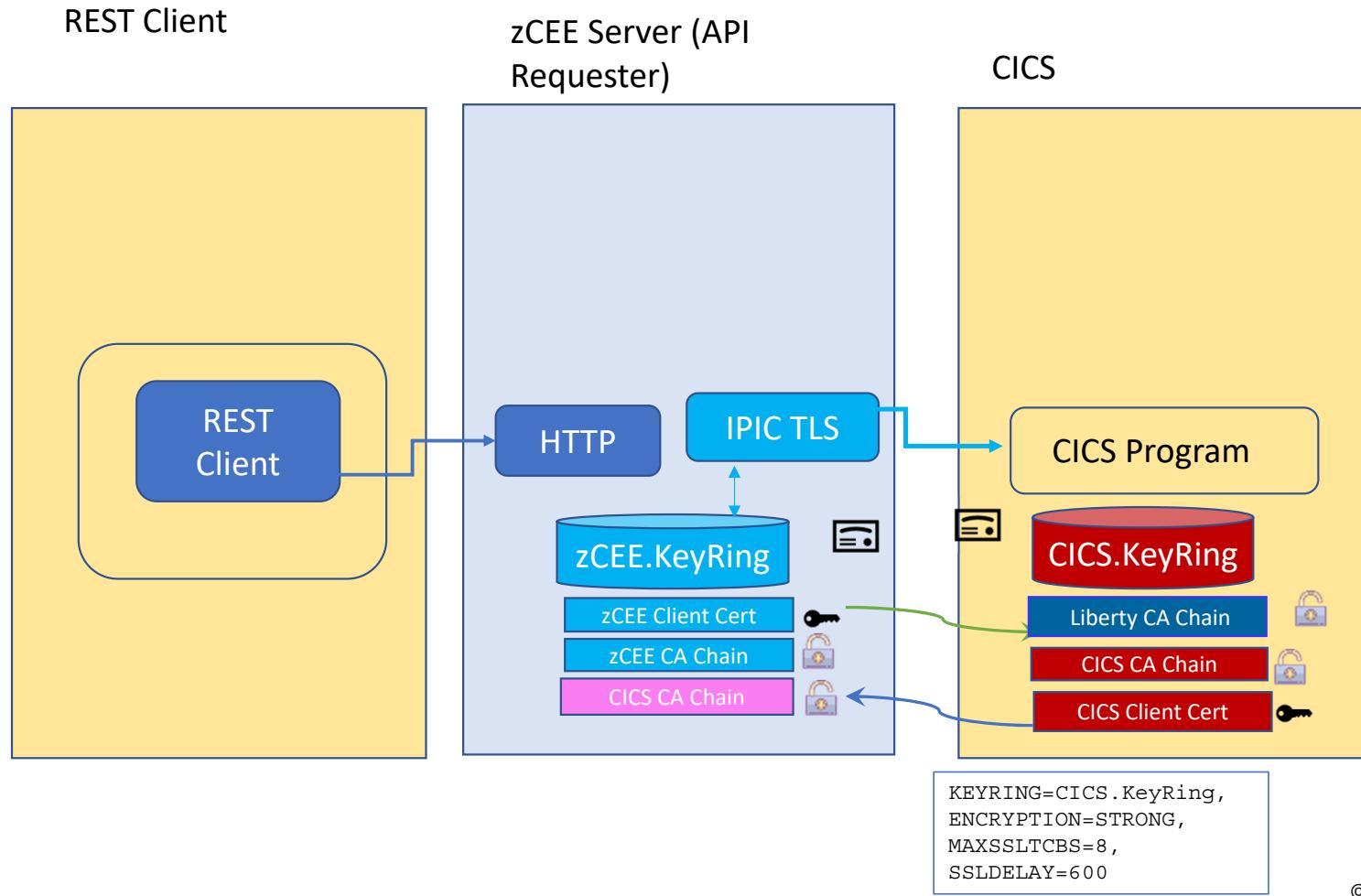


## TLS handshakes between all endpoints





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





# 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](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>



## Persistent connections

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

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

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

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



## TLS sessions

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

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

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

<ssloptions id="mySSLOptions" sslRef="DefaultSSLSettings"
sslSessionTimeout="10m"/>
```

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



# 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



# CICS IPIC using TLS

The server.xml file is the key configuration file:

**Required Configuration**

- Coded character set identifier (CCSID): 37
- Connection reference: catalog

**Optional Configuration**

- Transaction ID: (empty)
- Transaction ID usage: (dropdown menu)

**ipicSSLIDProp.xml**

```

<server description="CICS IPIC ID propagation connections">
  <!-- Enable features -->
  <featureManager>
    <feature>zosconnect:cicsService-1.0</feature>
  </featureManager>
  <zosconnect_cicsIpIcConnection id="catalog" host="wg31.washington.ibm.com" port="1493" zosConnectNetworkId="CSCVINC" zosConnectApplid="CSCVINC" transid="M10" transactionUsage="EIB_AND_MIRROR" sslCertsRef="cicsSSLSettings"/>
</server>

```

**Define IPIC/TLS connections to CICS**



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

2-character cipher number	4-character cipher number	Short name	Description <sup>1</sup>	FIPS 140-2	K Base security level FMID HCPT510	K Security level 3 FMID JCPT511	Dark mode
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 40-bit RC4 encryption	X	X		
03	0003	TLS_RSA_EXPORT_WITH_RC4_40_MD5	with MD5 message authentication and RSA (export) key exchange 128-bit RC4 encryption	X	X		
04	0004	TLS_RSA_WITH_RC4_128_MD5	with MD5 message authentication and RSA key exchange 128-bit RC4 encryption		X		
05	0005	TLS_RSA_WITH_RC4_128_SHA	with SHA-1 message authentication and RSA key exchange 40-bit RC2 encryption		X		
06	0006	TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	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>



# MQ JMS using TLS

\*twoWay Service

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Connection factory JNDI name: jms/qmgrCf

Request destination JNDI name: jms/requestQueue

Reply destination JNDI name: jms/replyQueue

Wait interval: 3000

MQMD format: MQSTR

Coded character set identifier (CCSID): 37

Is message persistent:

Reply selection: msgIDToCorrelID

Expiry: -1

Definition Configuration

LIBERTY.SSL.SVRCONN - Properties

General Extended MCA Exits SSL Statistics

SSL

CipherSpec

Set message security for this end of the channel

SSL Cipher Spec: TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256

Accept only certificates with Distinguished Names matching these values:

SSL Authentication: Required

Certificate label:

Apply OK Cancel

The server.xml file is the key configuration file:

Server Config

mqClientTLS.xml

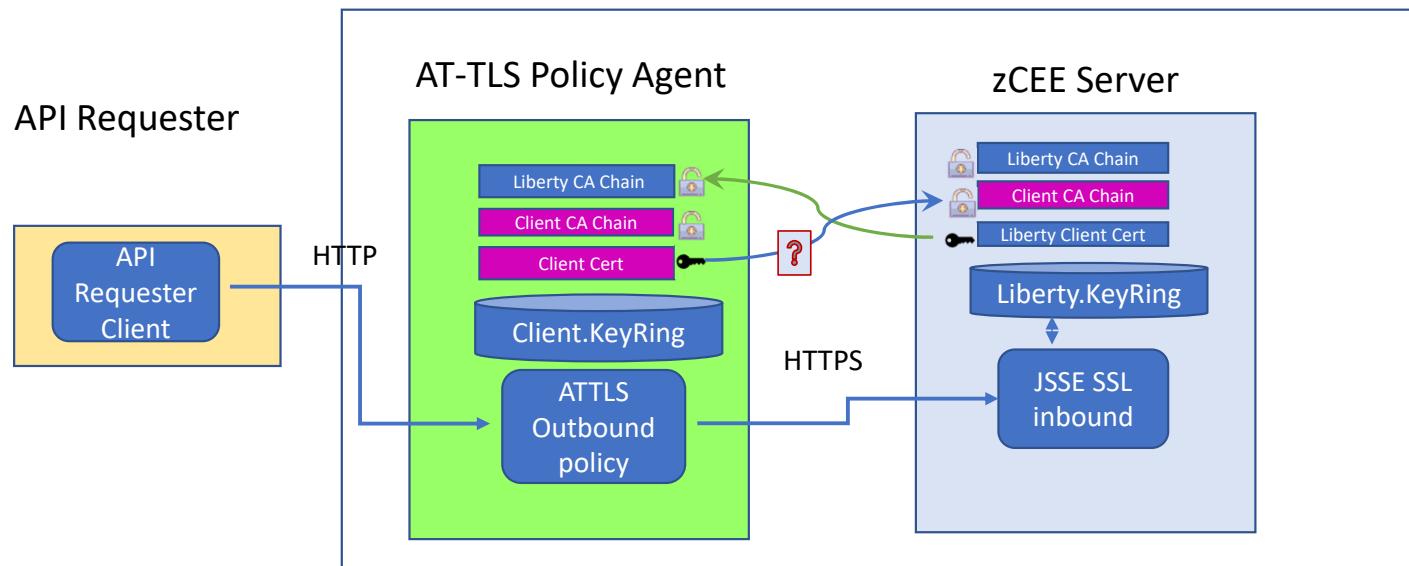
Design Source

```
<server description="MQ Service Provider">
  <featureManager>
    <feature>zosconnect:mqService-1.0</feature>
  </featureManager>
  <variable name="wmqJmsClient.rar.location"
    value="/u/johnson/jca/wmq.jmsra.rar"/>
  <wmqJmsClient nativeLibraryPath="/usr/lpp/mqm/V9R1M1/java/lib"/>
  <zosconnect_services>
    <service name="mqPutService">
      <property name="useCallerPrincipal" value="true"/>
    </service>
  </zosconnect_services>
  <connectionManager id="ConMgr1" maxPoolSize="5"/>
  <jmsConnectionFactory id="qmgrCf" jndiName="jms/qmgrCf">
    <connectionManagerRef>ConMgr1</connectionManagerRef>
    <properties.wmqJMS transportType="CLIENT"
      queueManager="ZMQ1"
      channel="LIBERTY.SSL.SVRCONN"
      hostName="wg31.washington.ibm.com"
      sslCipherSuite="SSL_RSA_WITH_AES_256_CBC_SHA256"
      port="1422"/>
  </jmsConnectionFactory>
  <jmsQueue id="q1" jndiName="jms/default">
    <properties.wmqJMS
      baseQueueName="ZCEE.DEFAULT.MQZCEE.QUEUE"
      CCSID="37"/>
  </jmsQueue>
</server>
```



## AT-TLS - outbound policy handshake scenarios

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



🔑 Certificate with a private key

🔒 Certificate Authority (CA) certificate

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

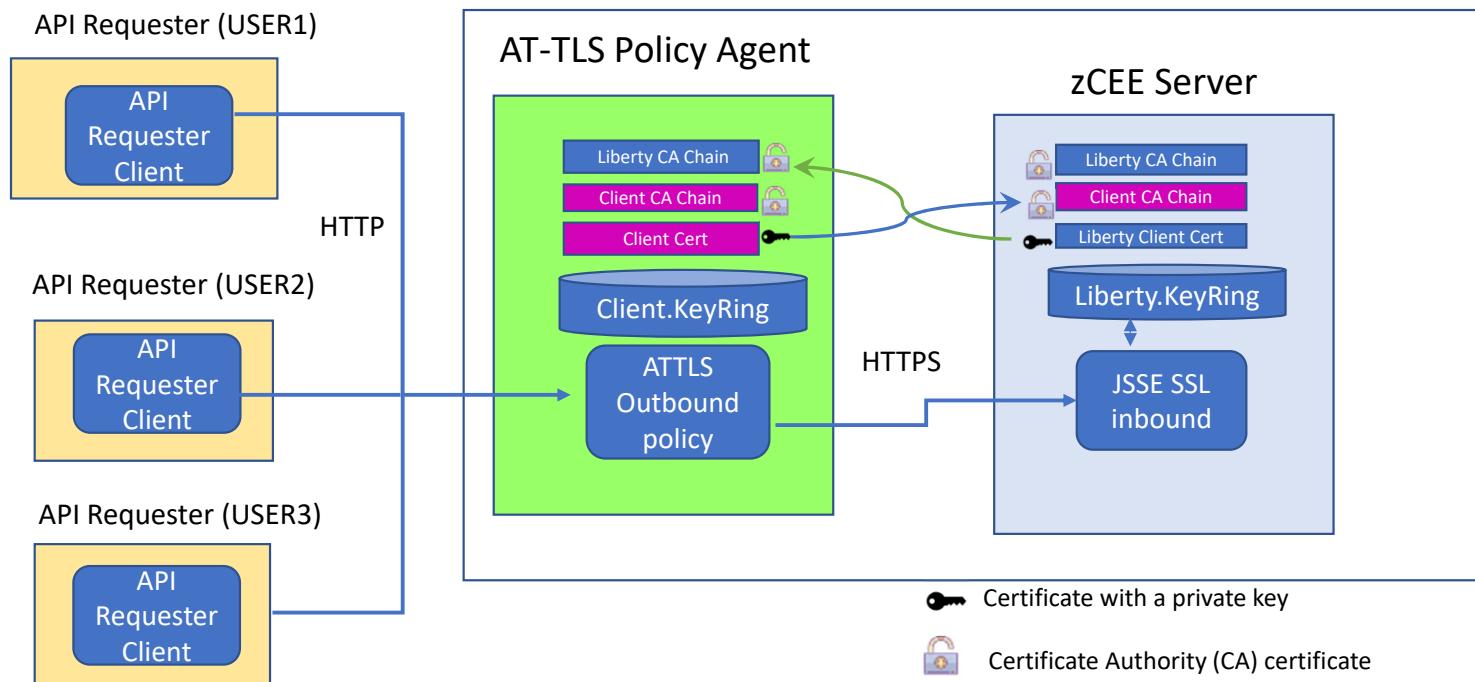


Question if this really needed, remember TLS encryption is independent of TLS authentication.



## AT-TLS - outbound policy handshake scenario

Use of a common key ring name for multiple client identities



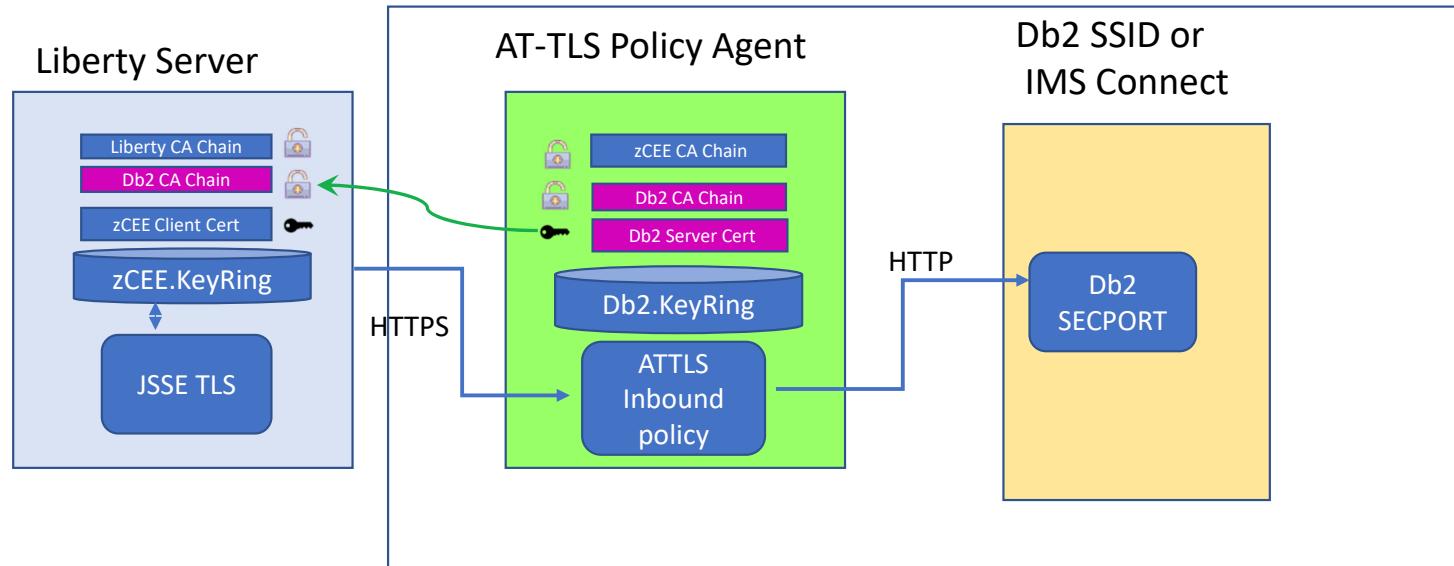
- Each user owns a keyring with the name Liberty.KeyRing.
- Each key ring has a different default client certificate for mutual authentication purposes.

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



## 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 unaware

Certificate with a private key

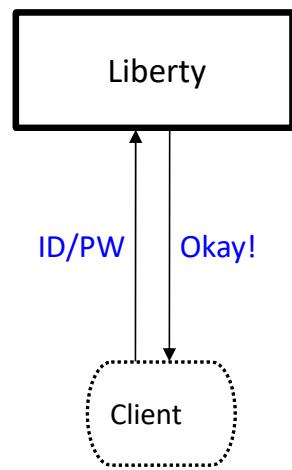
Certificate Authority (CA) certificate



# Authentication - Third Party Authentication

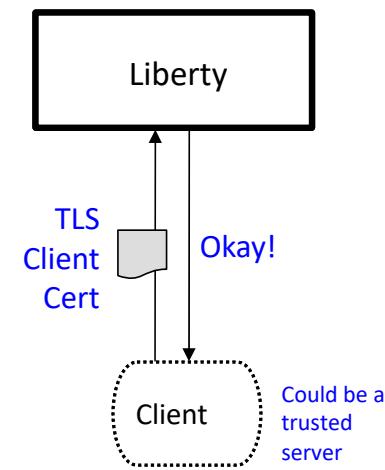
Several different ways this can be accomplished:

## Basic Authentication



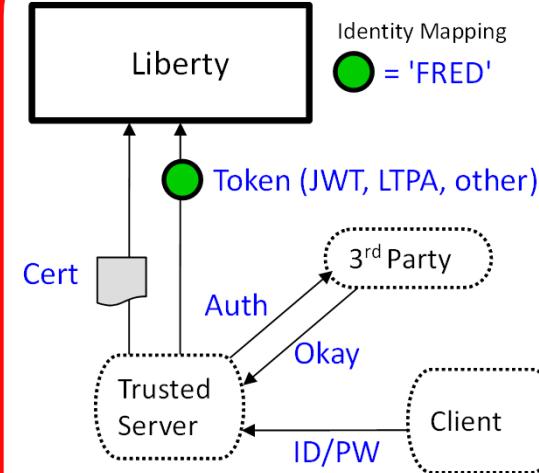
Server prompts for ID/PW  
Client supplies ID/PW or ID/PassTicket  
Server checks registry:  
• Basic (server.xml)  
• SAF

## 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 3<sup>rd</sup> party sever  
Client receives a trusted 3<sup>rd</sup> party token  
Token flows to Liberty z/OS and is mapped to an identity  
Registry options:  
We may know these detail.



# Third Party Authentication Examples

The image displays two side-by-side screenshots of web pages illustrating third-party authentication.

**Left Screenshot: UPS Sign Up**

This screenshot shows the UPS Sign Up page. At the top, there's a banner stating "UPS is open for business: Service impacts related to Coronavirus ...More". Below the banner, the UPS logo is displayed. A "Sign Up / Log in" link and a "Search or Track" input field are visible. The main section is titled "Sign Up" and includes a link for users who already have an ID. It provides several social media integration options: Google, Facebook, Amazon, Apple, and Twitter. Below these, there are fields for entering personal information: Name\*, Email\*, User ID\*, Password\*, and Phone. The password field includes a "Show" link. A "Feedback" button is located on the right side of the form.

**Right Screenshot: myNCDMV Log In**

This screenshot shows the myNCDMV Log In page. The background features a scenic view of autumn foliage. The page has "Log In" and "Sign Up" tabs at the top. The "Log In" tab is selected. It contains fields for "Email Address" (with placeholder "name@example.com") and "Password" (with placeholder "\*\*\*\*\*"). There is also a "Remember Me" checkbox. Below the password field are "Log In" and "Forgot Password" buttons. Further down, there are three social media login options: "Continue with Apple", "Continue with Facebook", and "Continue with Google". A "Continue as Guest" link is also present. A notice at the bottom states: "NOTICE FOR PUBLIC COMPUTER USERS - If you sign in with Google, Apple, or Facebook you are also signing into that account on this computer. Remember to sign out when you're done." The page is powered by "payit".



# 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](https://openid.net/specs/openid-connect-core-1_0.html)

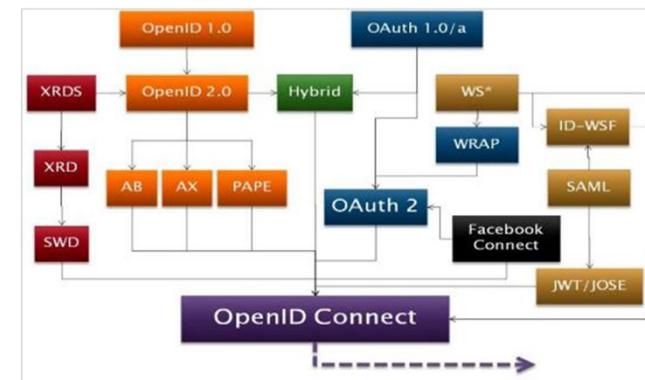
See the YouTube videos:

OAuth 2.0 and OpenID Connect (in plain English)

<https://www.youtube.com/watch?v=996OjexHze0>

OpenID Connect on Liberty

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



# 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

The screenshot shows the jwt.io debugger interface. On the left, under 'Encoded', is a long string of characters representing the JWT. On the right, under 'Decoded', are the three parts: 'HEADER', 'PAYLOAD', and 'Signature'. The 'Signature' part is highlighted with a red oval. A tooltip above the oval reads 'Mon Nov 02 2020 11:05:58 GMT-0500 (Eastern Standard Time)'. The 'PAYLOAD' section contains the following JSON:

```
{  
  "kid": "4qjX-bkX0Uw_F_uccjRMKB9ivMjXSQwj0RrkYRJq8DM",  
  "alg": "RS256"  
}  
  
{  
  "sub": "Fred",  
  "token_type": "Bearer",  
  "scope": [  
    "openid",  
    "profile",  
    "email"  
  ],  
  "azp": "rpSsl",  
  "iss": "https://wg31.washington.ibm.com:26213  
/oidc/endpoint/0P",  
  "aud": "myZcee",  
  "exp": 1604333158,  
  "iat": 16043330858,  
  "realmName": "zCEERealm",  
  "uniqueSecurityName": "Fred"  
}
```

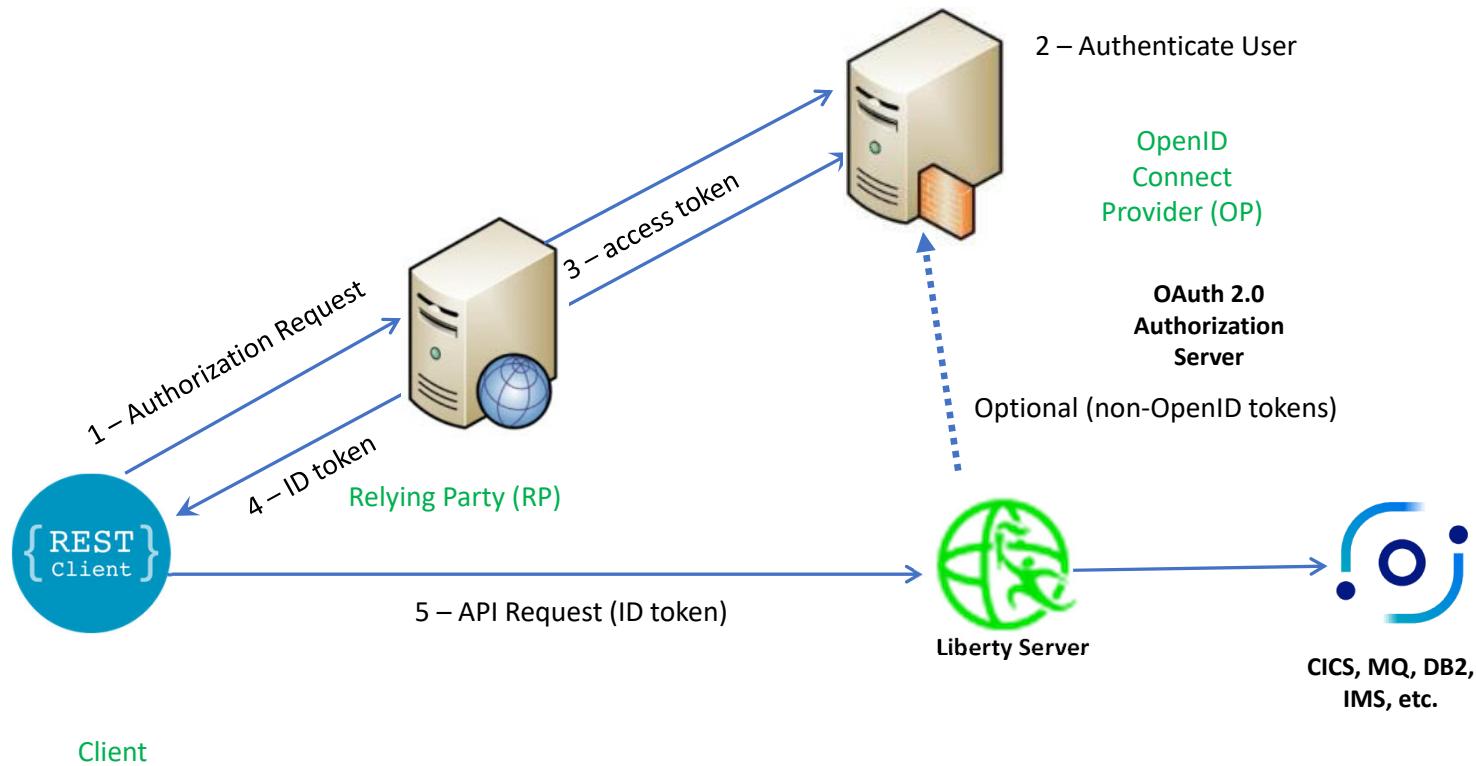
Values derived from the OAUTH configuration:

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

<https://jwt.io>



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



## 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](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=996OjexHze0>

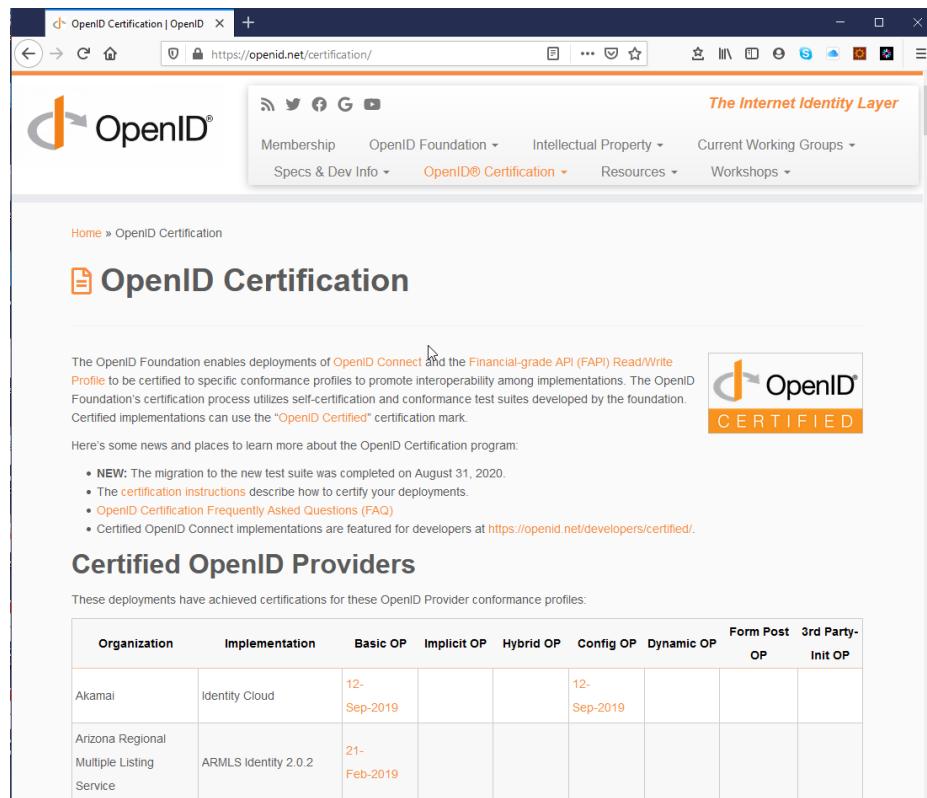
## Tech-Tip: Some basic OAuth/OpenID Connect terms

- **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 (JWT). 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.

## 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](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



The screenshot shows a web browser displaying the OpenID Certification page at <https://openid.net/certification/>. The page is titled "OpenID Certification" and features the OpenID logo. It includes a brief description of the certification process, a "CERTIFIED" badge, and a table of certified providers.

**Certified OpenID Providers**

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

<https://openid.net/certification/>



# Liberty OpenID Client identity mapping configuration attributes

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "kid": "kvjtqdlMjOTWiJrjOr73fu2MMt-FjiQrxU0YBzJLR4o",  
  "alg": "RS256"  
}
```

PAYOUT: 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="ATSRererUri"  
  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.

## 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:
>distribute_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:
>*<
```



## 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>
```



## 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



# 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



# 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

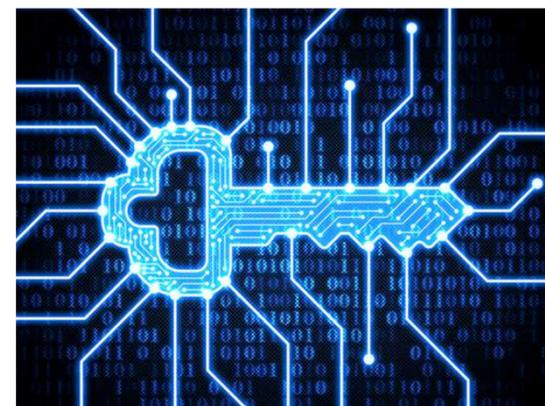
- A **remoteAddress** element is compared against the TCP/IP address of the client that sent the request.
- The **host** element is compared against the "Host" HTTP request header, which identifies the target host name of the request.
- The **requestUrl** element is compared against the URL that is used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

# General security terms or considerations

Security involves

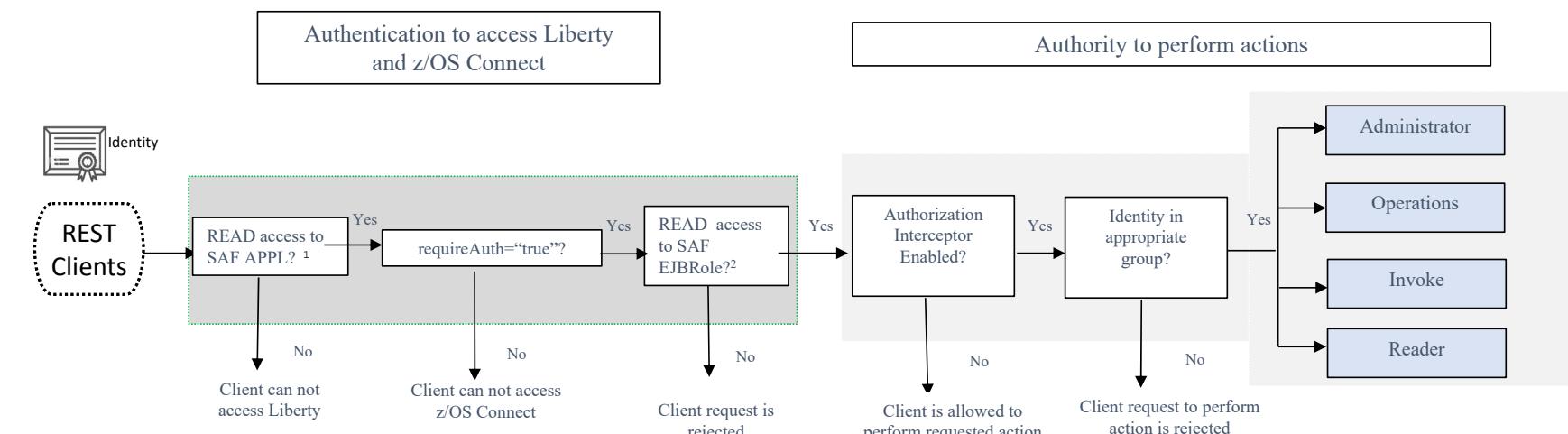
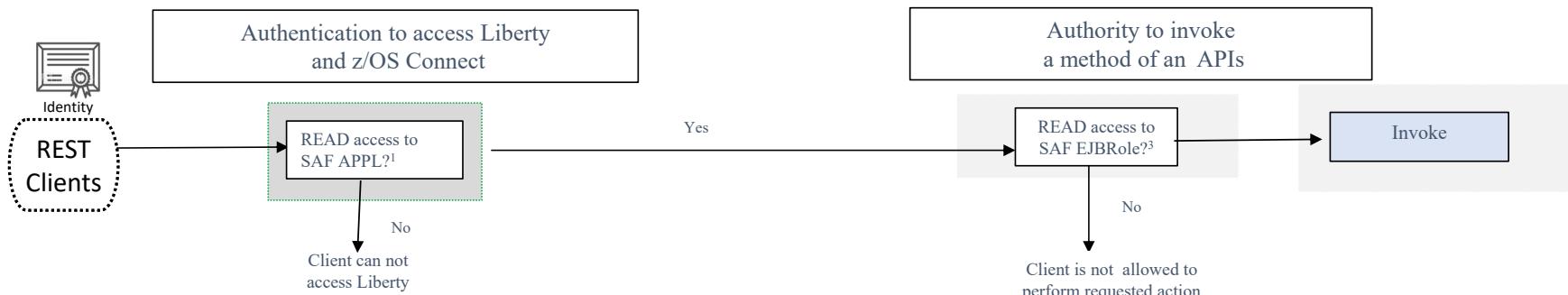
- Identifying who or what is requesting access (**Authentication**)
  - Basic Authentication
  - Mutual Authentication using Transport Layer Security (TLS), formerly known as SSL
  - Third Party Tokens
- Ensuring that the message has not been altered in transit (**Data Integrity**) and ensuring the confidentiality 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.





# Security flow – authentication/authorization



<sup>1</sup>RDEFINE APPL profilePrefix

<sup>2</sup>RDEFINE EJBROLE profilePrefix.zos.connect.access.roles.zosConnectAccess

<sup>3</sup>RDEFINE EJBROLE profilePrefix.resourceName.role



## 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-defined 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 from 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

The screenshot shows an IDE interface with the file 'web.xml' open. The code is a Java XML configuration for a web application. Two specific sections are highlighted with red boxes:

```
<!-- Allow unsecured GET access to resources required before login. -->
<security-constraint>
    <web-resource-collection>
        <url-pattern>/*</url-pattern>
        <http-method>GET</http-method>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
</security-constraint>

<!-- Enforce login to relevant resources. -->
<security-constraint>
    <web-resource-collection>
        <url-pattern>/</url-pattern>
        <url-pattern>/internal/*</url-pattern>
        <url-pattern>/index.html</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>MQWebAdmin</role-name>
        <role-name>MQWebAdminRO</role-name>
        <role-name>MQWebUser</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>

```

- MQWebAdmin
- MQWEBAdminRO
- MQWebUser

**WAR - Web application ARchive:** a compressed file that contains the libraries, JAR files, JSP, etc. that an application requires for deployment.

**EAR – Enterprise application ARchive:** a compressed file that contains the libraries, enterprise beans, and JAR files that an application requires for deployment.



## 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>



# 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.



# 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>
```

The screenshot shows a Windows Notepad window titled "cscvinc.yaml - Notepad". The file contains YAML configuration for a REST API endpoint. It includes sections for 'openapi', 'info', 'servers', 'x-ibm-zcon-roles-allowed', 'paths', and various 'post', 'get', and 'put' operations with their respective tags, operation IDs, and 'x-ibm-zcon-roles-allowed' lists.

```
*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



## 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>
```



## 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 EJB Roles 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.



## 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
```



## 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"/>
```

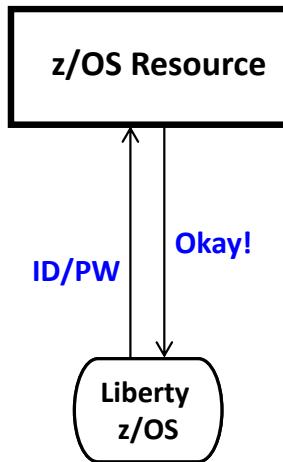
## **Security when accessing z/OS subsystems**



# Accessing z/OS resources

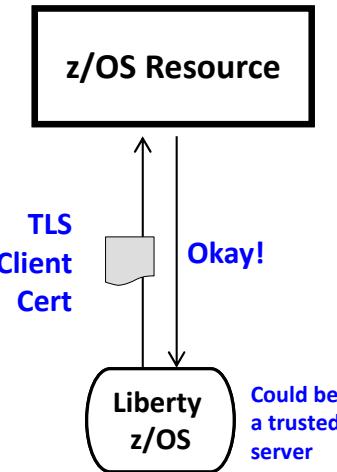
Several different ways this can be accomplished:

Basic Authentication



Liberty supplies ID/PW or  
ID/PassTicket

Client Certificate



Server prompts for certificate  
Liberty supplies certificate



# 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="DFSIIVPA"  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>
```



# 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.



# 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

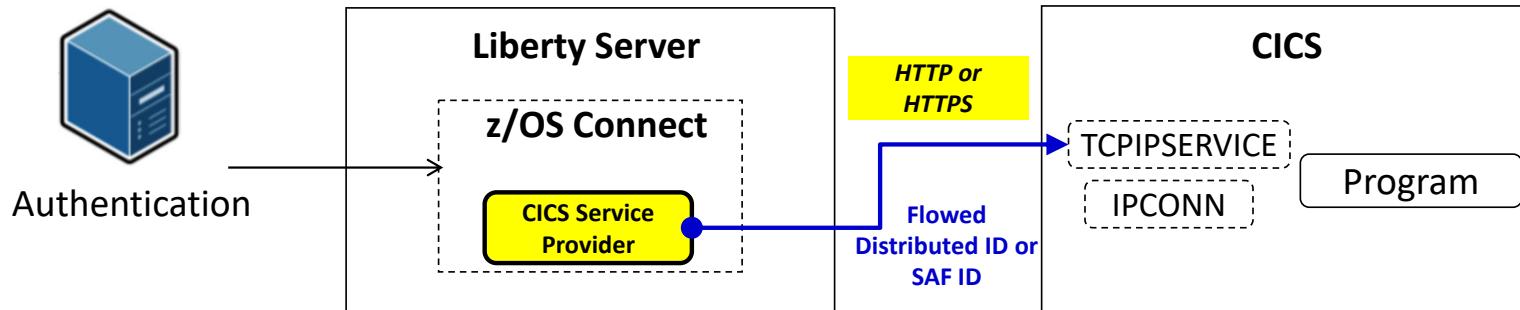
```
./securityUtility encode --encoding=hash XXXXXXXX
```

```
{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>

# 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.



# Flowing an identity to CICS

The zosconnect\_cicsIpConnection element is the key :

The screenshot shows the 'inquireSingle Service' configuration window. In the 'Required Configuration' section, 'Coded character set identifier (CCSID)' is set to 37 and 'Connection reference' is set to 'catalog'. In the 'Optional Configuration' section, 'Transaction ID' is set to 'catalog' and 'Transaction ID usage' is set to 'None'. The 'Configuration' tab is selected.

```
<zosconnect_cicsIpConnection id="catalog"
    host="wg31.washington.ibm.com"
    zosConnectNetworkid="CSCVINC"
    zosConnectApplid="CSCVINC"
    port="1493"/>
```

The terminal window displays the configuration of the TCP/IP service. The command issued is:

```
I TCPIPS
RESULT - OVERTYPE TO MODIFY
Tcpiipservice(CSCVINC)
Openstatus( Open )
Port(01493)
Protocol(pic)
Ssltype(Ssl)
Tlsversion(TLS)
Authenticate(Noauthentic)
Connections(000000)
Backlog( 01024 )
Maxdatalen( 000000 )
Urm( DFHISAPIP )
Privacy(Supported)
Ciphers(3538392F3233)
Host(ANY)
IpAddress(192.168.17.201)
Hosttype(Any)
Ipresolved(192.168.17.201)
+ Ipfamily(Ipv4family)
```

CICS TCPIPSERVICE

The terminal window displays the configuration of the CICS IPConn service. The command issued is:

```
I IPCONN
RESULT - OVERTYPE TO MODIFY
Ipconn(CSCVINC)
Applid(CSCVINC)
Networkid(CSCVINC)
Servstatus( Inservice )
Connstatus( Released )
Ssltype(Nossl)
Purgeinterval( 0 )
Hal(Notrequired)
Recvcount(001)
Sendcount(000)
Tcpiipservice(CSCVINC)
Port()
Host()
Hosttype()
Ipresolved(0.0.0.0)
Ipfamily(Unknown)
Pendstatus( Notpending )
+ Recovstatus( Norecovdata )
```

SYSID=CICS APPLID=CICS53Z TIME: 12.36.15 DATE: 02/22/21

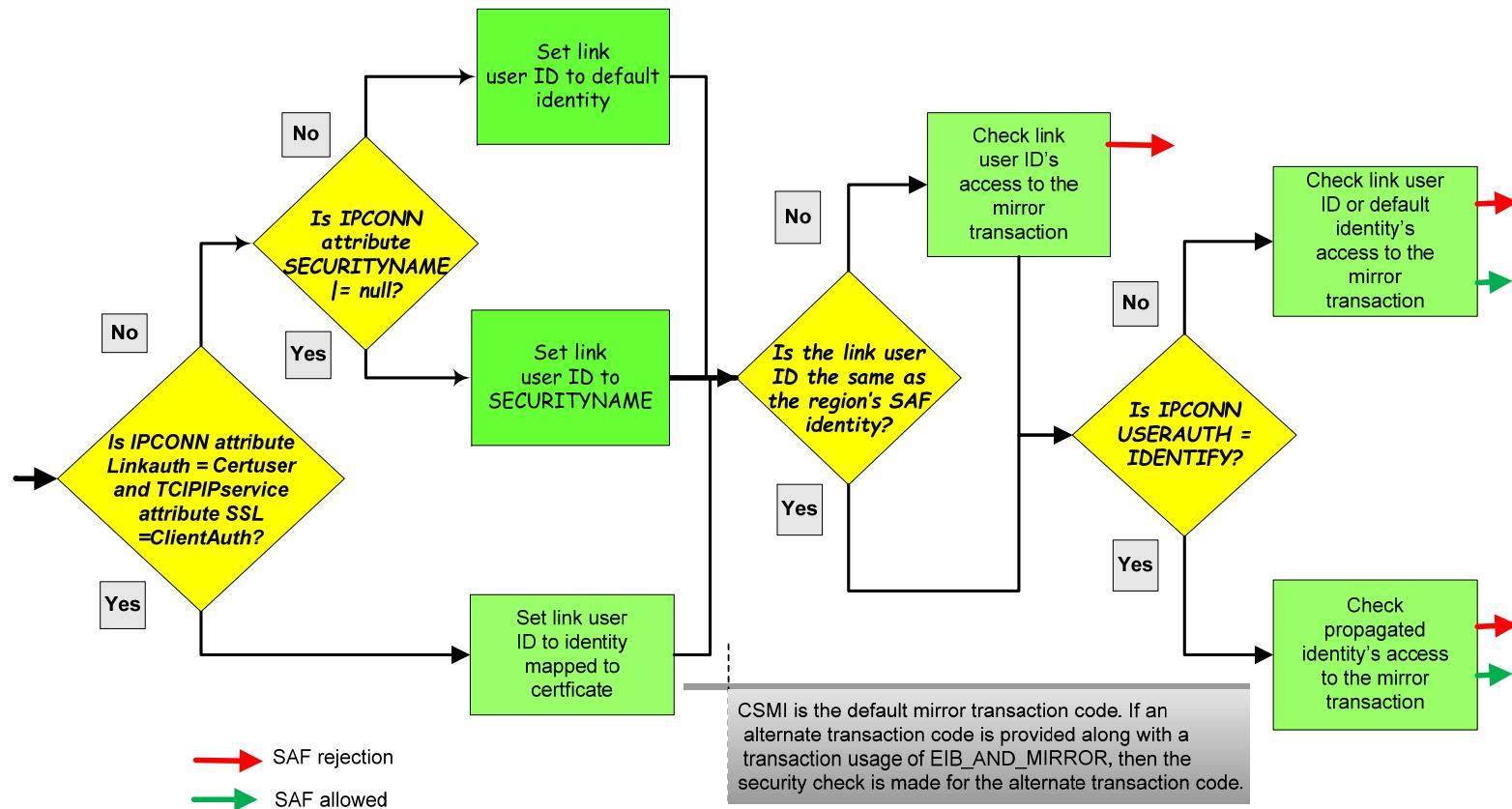
PF 1 HELP 2 HEX 3 END 5 VRR 7 SBH 8 SFH 10 SB 11 SF 17/044

Connected to remote server/host wg31 using lu/pool TCP00135 and port 23

CICS IPConn



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



# CICS IPCONN Resources



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

zosConnectApplid must match APPLID  
in an IPCONN resource

zosConnectNetworkid must match  
NETWORKID in an IPCONN resource

```
DEFINE IPCONN (ZOSCONN)  
    GROUP (SYSPGRP)  
    APPLID (ZCAPPL)  
    NETWORKID (ZCNETID)  
    TCPIPSERVICE (ZOSCONN)  
    LINKAUTH (SECUSER | CERTUSER)  
    USERAUTH (IDENTIFY)  
    IDPROP (REQUIRED | OPTIONAL)
```

**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.



# 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}"  
zosConnectApplid="${wlp.server.name}"  
sharedPort="true" port="${ipicPort}"/>  
<zosconnect_cicsIpicConnection id="catalog"  
host="${cicsHost}"  
zosConnectNetworkid="${serverPrefix}C"  
zosConnectApplid="${serverPrefix}C"  
sharedPort="true" port="${ipicPort}"/>  
<zosconnect_cicsIpicConnection id="miniloan"  
host="${cicsHost}"  
zosConnectNetworkid="${serverPrefix}M"  
zosConnectApplid="${serverPrefix}M"  
sharedPort="true" port="${ipicPort}"/>
```

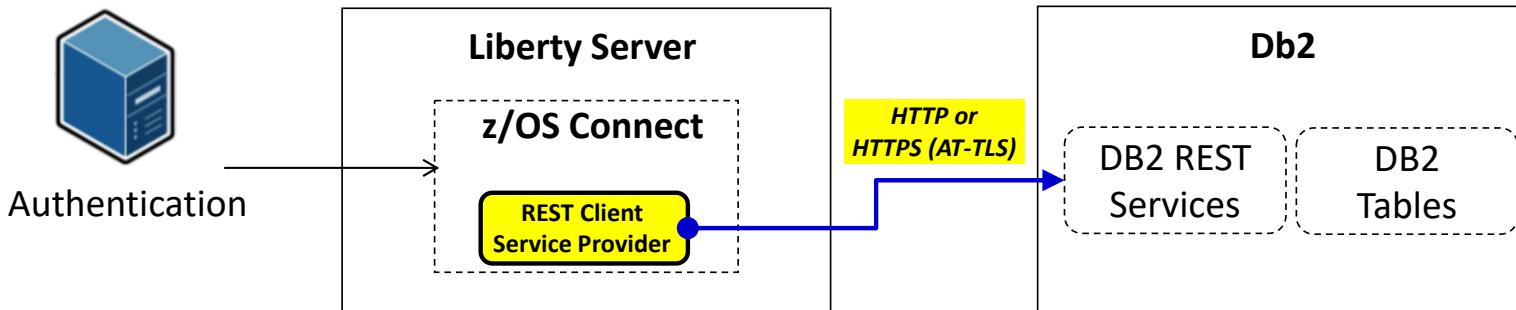
→ baqsvr1 or baqsvr2

→ baqsvr1C or baqsvr2C

→ baqsvr1M or baqsvr2M



## 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



# Server XML - Accessing a Db2 REST service

\*selectEmployee Service X

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Connection reference: db2conn

Definition Configuration

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" />
```



# PassTickets and Db2

- ☐ Basic authentication Db2 using a PassTicket depends on the Db2 configuration.

```
DSNL080I -DSN2 DSNLDDF DISPLAY DDF REPORT FOLLOWS:
```

```
DSNL081I STATUS=STARTD
DSNL082I LOCATION      LUNAME          GENERICCLU
DSNL083I DSN2LOC       USIBMWZ.DSN2APPL  USIBMWZ.DSNOAPPL
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 DSNLDDF DISPLAY DDF REPORT COMPLETE
```

```
DSNL080I -DSNC DSNLDDF DISPLAY DDF REPORT FOLLOWS:
```

```
DSNL081I STATUS=STARTD
DSNL082I LOCATION      LUNAME          GENERICCLU
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 DSNLDDF 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 determined for use in RACF resources is determined 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***

# IMS TM Connections and Interactions



ivtnoService Service

**Configuration**

**Required Configuration**

Enter the required configuration for this service.

Connection profile: **IMSCONN** (circled in red)

Interaction profile: **IMSINTER** (circled in red)

**Optional Configuration**

Enter the optional configuration for this service.

IMS destination override:

Program name:

Overview Configuration

Connection Profile

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

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

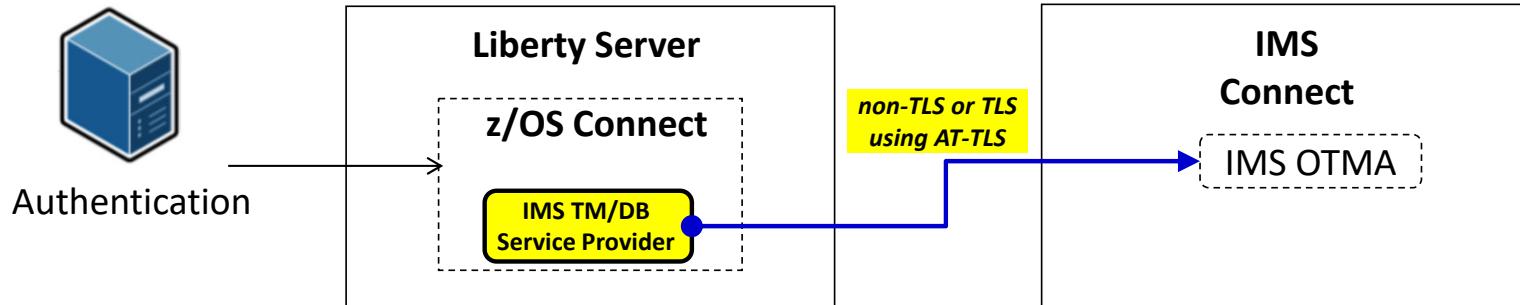
Interaction Profile

```
<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)
```

# 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



# 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=IMSTMPL)
ODACCESS= (ODBMAUTOCONN=Y, IMSPLEX= (MEMBER=IMS15HWS, TMEMBER=PLEX1) ,
DRDAPORT= (ID=5555, PORTTMOT=6000) , ODBMTMOT=6000, APPL=IMSDBAPL)
```

```
RDEFINE PTKTDATA IMSTMPL SSIGNON(0123456789ABCDEF) APPLDATA('NO REPLAY PROTECTION') UACC(NONE)
```

```
RDEFINE PTKTDATA IRRPTAUTH.IMSTMPL.* UACC(NONE)
```

```
PERMIT IRRPTAUTH.IMSTMPL.* 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

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Connection profile:

## ConnectionFactory

```
<connectionFactory id="DFSIVPACConn">
<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 , T MEMBER=OTMAMEM )
IMSPLEX= ( MEMBER=IMS14HWS , T MEMBER=PLEX1 )
ODACCESS= ( ODBMAUTOCCONN=Y ,
DRDAPORT= ( ID=5555 , PORTTMOT=6000 ) , ODBMTMOT=6000 )
```



# API toolkit – Creating Services for IMS DB

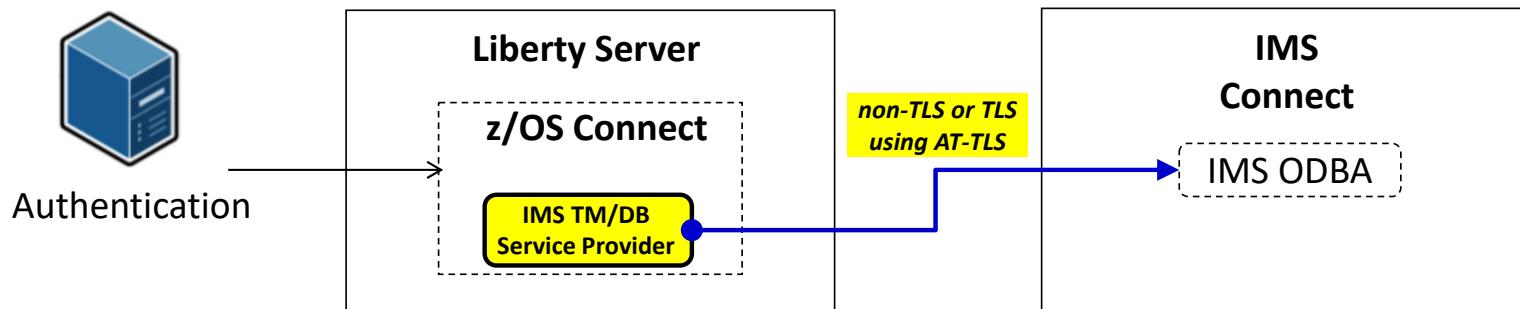
## Creating a service project from the IMS Catalog

The screenshot shows the 'Service Project Editor: Definition' window for an 'IMS Database Service'. The 'General Information' section shows 'Type: IMS Database Service', 'Version: 1.0.0', and a 'Description' field. The 'Actions' section lists steps: 1. Input service version, 2. Specify the SQL command for the service, 3. Specify Database Connection Properties and Generate Service Interface, 4. Complete the configuration for the service, 5. Deploy the service, and 6. Export the service. The 'Enter or import SQL Command' section contains the SQL query: 'SELECT FIRSTNAME, ZIPCODE, PHONENBR, A1111111 FROM ATSVPA.A1111111 WHERE A1111111=?'. This query is circled in red. Below it, the 'Specify Database Connection Properties and Generate Service Interface' section shows 'Database Connection: wg31:5555' and 'Database Name: DFSIVPA'. There is also a 'Generate Service Interface...' button.

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=DFSVVD1,DEVICE=3380,SIZE=2048  
SEGM NAME=A1111111,PARENT=0,BYTES=40,RULES=(LLV,LAST),  
PTR=(TB,CTR)  
FIELD NAME=(A1111111,SEQ,U),BYTES=010,START=00001,TYPE=C  
FIELD NAME=FIRSTNAME,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
```

# 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="DFSIVPACConn"> <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="DFSIVPACConn"> <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



# Using JMS to access MQ (One-Way)

mqGetService Service

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Connection factory JNDI name: jms/qmgrCf

Destination JNDI name: jms/default

Coded character set identifier (CCSID): 37

Optional Configuration

Enter the optional configuration for this service.

Wait interval:

Message selector:

Definition Configuration

mqClient.xml

Read only Close

Design Source

```
<server description="MQ Service Provider">
<featureManager>
    <feature>zosconnect:mqService-1.0</feature>
</featureManager>
<variable name="wmqJmsClient.rar.location"
    value="/usr/lpp/mqm/V9R1M1/java/lib/jca/wmq.jmsra.rar"/>
<wmqJmsClient nativeLibraryPath="/usr/lpp/mqm/V9R1M1/java/lib"/>
<zosconnect_services>
    <service name="mqPutService">
        <property name="useCallerPrincipal" value="false"/>
    </service>
</zosconnect_services>
<connectionManager id="ConMgr1" maxPoolSize="5"/>
<jmsConnectionFactory id="qmgrCF" jndiName="jms/qmgrCF">
    <connectionManagerRef>ConMgr1</connectionManagerRef>
    <properties.wmqJMS transportType="CLIENT"
        queueManager="ZMQ1"
        channel="LIBERTY.DEF.SVRCONN"
        hostname="wg31.washington.ibm.com"
        port="1422" />
</jmsConnectionFactory>
<jmsQueue id="q1" jndiName="jms/default">
    <properties.wmqJMS
        baseQueueName="ZCEE.DEFAULT.MQZCEE.QUEUE"
        CCSID="37"/>
</jmsQueue>
</server>
```



# Using JMS to access MQ (Two-Way)

\*twoWay Service X

**Service Project Editor: Configuration**

**Required Configuration**

Enter the required configuration for this service.

Connection factory JNDI name:

Request destination JNDI name:

Reply destination JNDI name:

Wait interval:

MQMD format:

Coded character set identifier (CCSID):

Is message persistent:

Reply selection:

Expiry:

**Definition Configuration**

mq.xml

Design Source

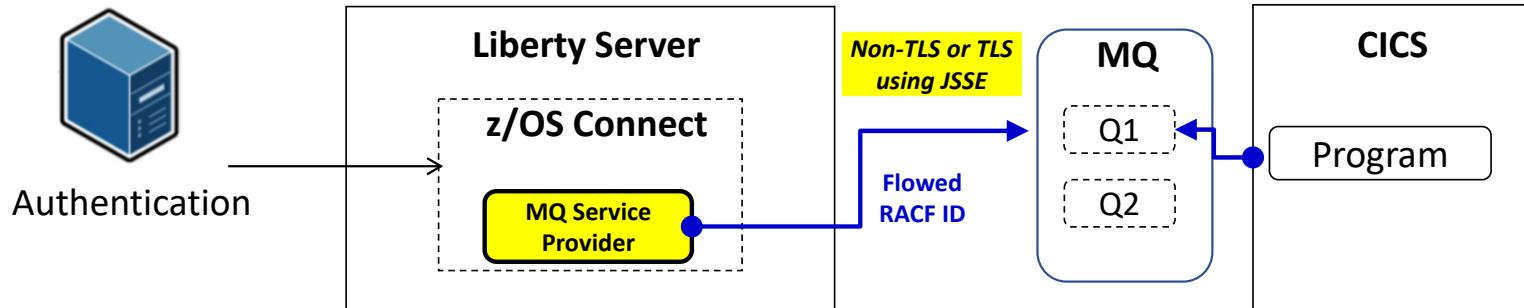
```

2 <featureManager>
3   <feature>zosconnect:mqService-1.0</feature>
4 </featureManager>
5
6 <variable name="wmqJmsClient.rar.location"
7   value="/usr/lpp/mqm/V9R1M1/java/lib/jca/wmq.jmsra.rar"/>
8 <wmqJmsClient nativeLibraryPath="/usr/lpp/mqm/V9R1M1/java/lib"/>
9
10 <connectionManager id="ConMgr1" maxPoolSize="5"/>
11
12 <jmsConnectionFactory id="qmgrCF" jndiName="jms/qmgrCf"
13   connectionManagerRef="ConMgr1">
14   <properties.wmqJms transportType="BROADCASTER"
15     queueManager="QMZ1" />
16 </jmsConnectionFactory>
17
18 <jmsConnectionFactory id="qmgrCF2" jndiName="jms/qmgrCF2"
19   connectionManagerRef="ConMgr1">
20   <properties.wmqJms transportType="CLIENT"
21     queueManager="ZMQ1"
22     channel="LIBERTY.DEF.SVRCONN"
23     hostName="wg31.washington.ibm.com"
24     port="1422" />
25 </jmsConnectionFactory>
26
27 <jmsQueue id="q1" jndiName="jms/default">
28   <properties.wmqJms
29     baseQueueName="ZCONN2.DEFAULT.MQZEE.QUEUE"
30     CCSID="37"/>
31 </jmsQueue>
32
33 <jmsQueue id="requestQueue" jndiName="jms/request">
34   <properties.wmqJms
35     baseQueueName="ZCONN2.TRIGGER.REQUEST"
36     targetClient="MQ"
37     CCSID="37"/>
38 </jmsQueue>
39
40 <jmsQueue id="replyQueue" jndiName="jms/replyQueue">
41   <properties.wmqJms
42     baseQueueName="ZCONN2.TRIGGER.RESPONSE"
43     targetClient="MQ"
44     CCSID="37"/>
45 </jmsQueue>
46
47

```



# 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>
```

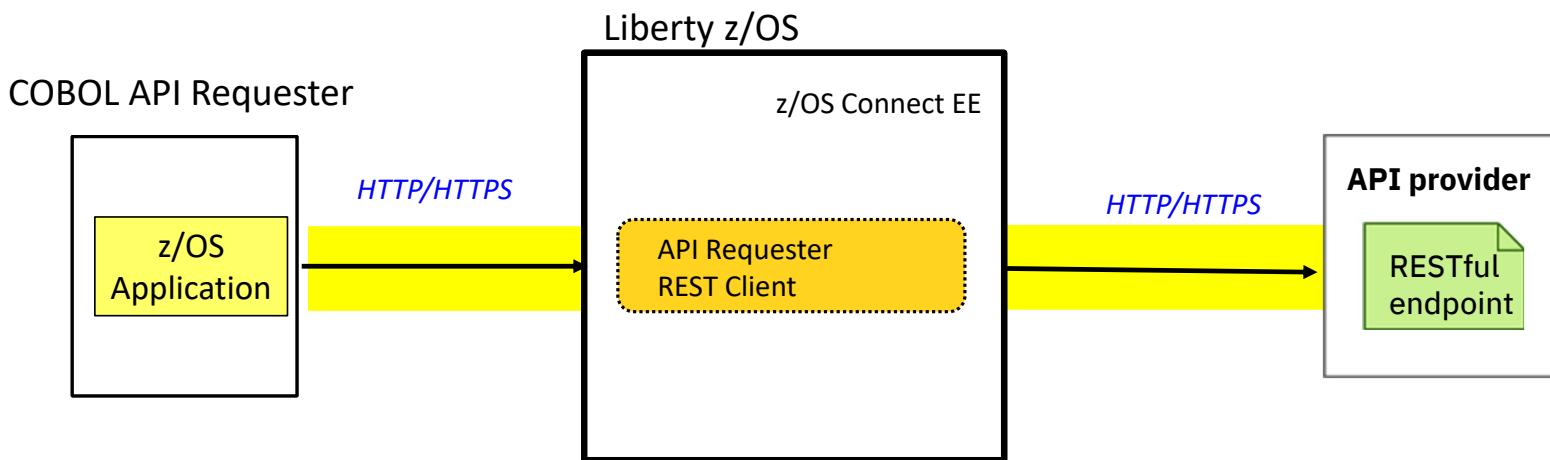
Define identity propagation to MQ

**Security when accessing non-z/OS systems**

**z/OS Connect API Requester Security**



# 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:

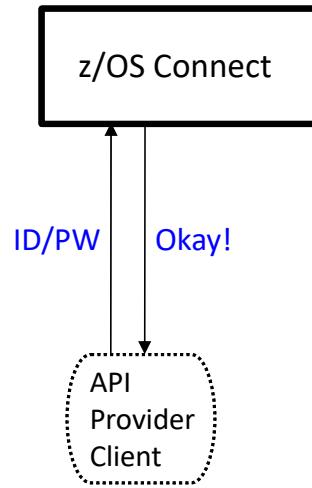
- CICS URIMAP resource (default BAQURIMP)
  - HOST
  - PORT
  - SCHEME (HTTP/HTTPS)



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

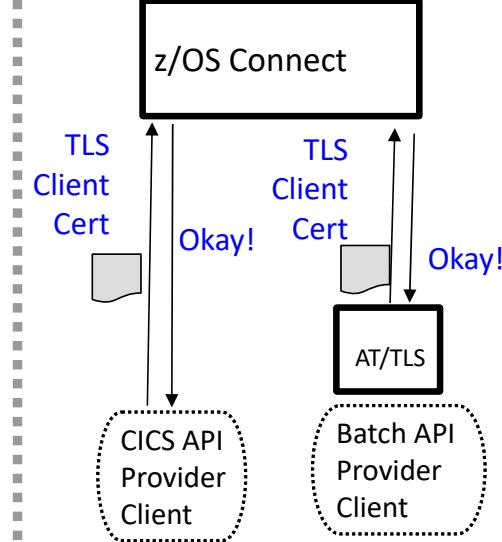
Two options for providing credentials for authentication

## 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**



# Configuring connections to the z/OS API requester server

## Default CICS URI MAP\*

```
WG31 - 3270
File Edit Settings View Communication Actions Window Help
I URIMAP
RESULT - OVERTYPE TO MODIFY
Urimap(BAQURIMP)
Usage(Client)
Enablestatus(Enabled)
Availstatus(Notapplic)
Scheme(Http)
Redirecttype(None)
Tcpinservice()
Port(09120)
Host(wg31.washington.ibm.com:9120)
Path(/)
Analyzerstat(Noanalyzer)
Hosttype(Hostname)
Ipresolved(0.0.0.0)
Ipfamily(Unknown)
Socketclose(000030)
Sockpoolsize(000000)
Transaction()
+ Converter()

PF 1 HELP 2 HEX 3 END      5 VAR      7 SBH 8 SFH      10 SB 11 SF
M A D
Connected to remote server/host wg31a using lu/pool TCP00120 and port 23
01/012
```

## 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



# 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:

```
//CEELOPTS 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**



# 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.



# 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.
***** PROCEDURES
***** 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
```

# Tech/Tip: API Requester - HTTP v HTTPS



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

```
CEE0PTS DD *
  POSIX(ON),
  ENVAR ("BAQURI=wg31.washington.ibm.com",
  "BAQPORT=9080")
```

```
CEE0PTS DD *
  POSIX(ON),
  ENVAR ("BAQURI=wg31.washington.ibm.com",
  "BAQPORT=9443")
```

CICS URIMAPS

The image shows two side-by-side CICS URIMAPS configuration screens. Both screens have a title bar 'WG31' and a menu bar 'File Edit Settings View Communication Actions Window Help'. The left screen is titled 'CICS RELEASE' and the right screen is titled 'CICS RELEASE = 0710'. Both screens show an 'OVERTYPE TO MODIFY' command followed by a 'CEDA ALTER UriMap(BAQURIMP)' command. The configuration details are as follows:

Setting	Value (Left Screen - HTTP)	Value (Right Screen - HTTPS)	
Urimap	: BAQURIMP	: BAQURIMP	
Group	: SYSGRP	: SYSGRP	
Description	==> URIMAP for z/OS Connect EE server	==> URIMAP for z/OS Connect EE server	
Status	==> Enabled	==> Enabled	
Usage	==> Client	==> Client	
	Server   Client   Pipeline   Atom	Server   Client   Pipeline   Atom	
	Jvmserver	Jvmserver	
Universal Resource Identifier			
SCHEME	==> HTTP	==> HTTPS	
Port	==> 09120	==> 09443	
HOST	==> wg31.washington.ibm.com	==> wg31.washington.ibm.com	
Path	==> /	==> /	
(Mixed Case)	==>	==>	
	==>	==>	
	==>	==>	
	==>	==>	
+ OUTBOUND CONNECTION POOLING			
	SYSID=CICS APPL	SYSID=CICS APPLID=CICSS3Z	
PF 1 HELP 2 COM 3 END	6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11	PF 1 HELP 2 COM 3 END	6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
MB C		MB C	
Connected to remote server/host wg31 using lu/pool TCP00133 and port 23		Connected to remote server/host wg31 using lu/pool TCP00133 and port 23	

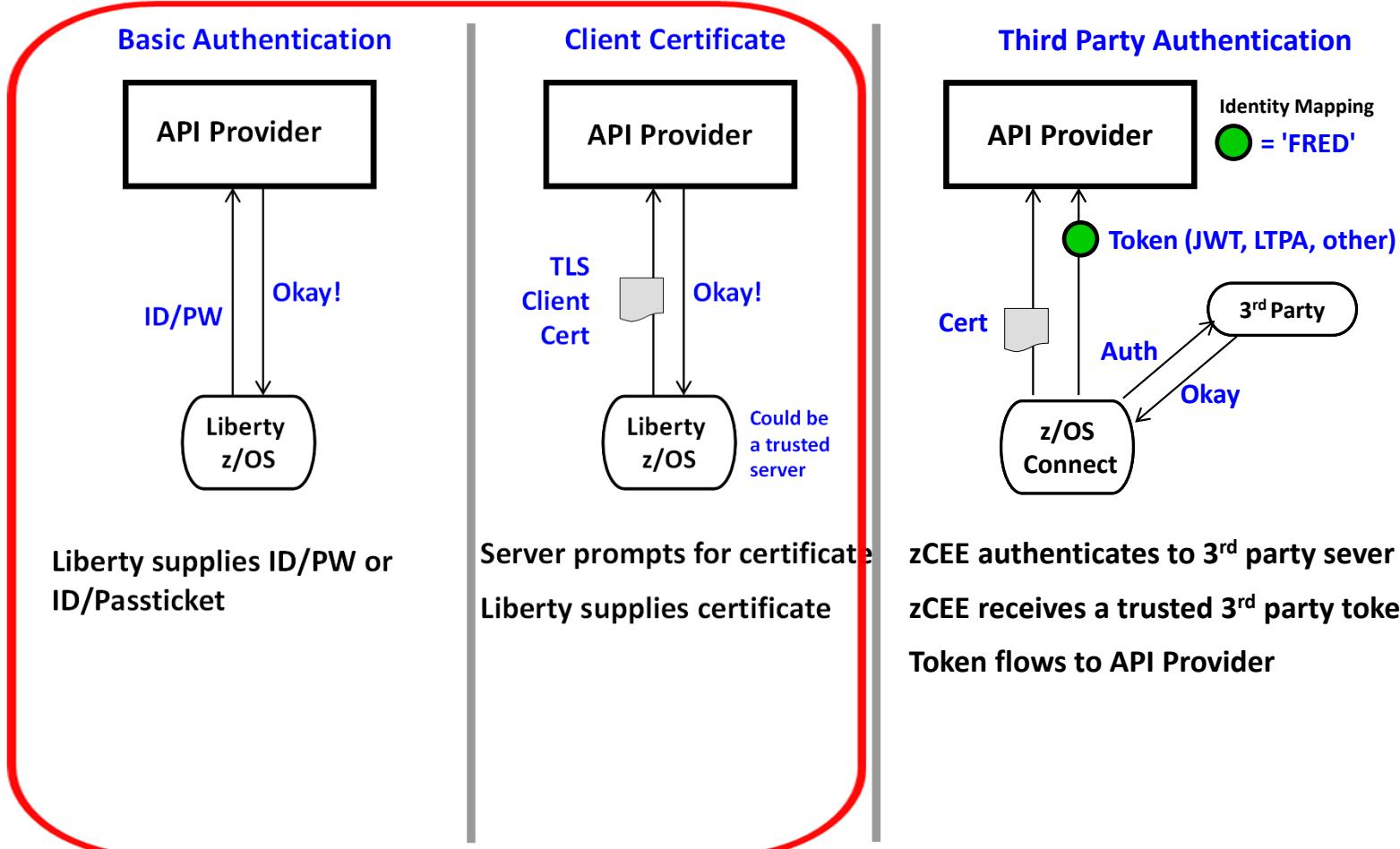
Field BAQ-ZCON-SERVER-URI was added to BAQRINFO in V3.0.37.

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



# Accessing non-z/OS resources

Several different ways this can be accomplished:





# Configuring Basic and/or TSL support

Basic authentication with HTTP protocol

```
<zosconnect_endpointConnection id="cscvincAPI"  
    host="http://wq31.washington.ibm.com" port="9080"  
    authenticationConfigRef="myAuthData" />  
  
<zosconnect_authData id="myAuthData"  
    user="zCEEClient" password="secret"/>
```

TLS with HTTPS protocol

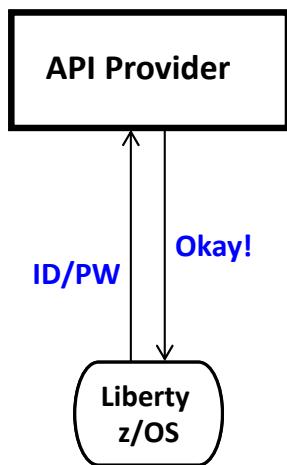
```
<zosconnect_endpointConnection id="cscvincAPI"  
    host="https://wq31.washington.ibm.com" port="9443"  
    authenticationConfigRef="myAuthData" 1  
    sslCertsRef="OutboundSSLSettings" />  
  
<zosconnect_authData id="myAuthData" 1  
    user="zCEEClient" password="secret"/>
```

<sup>1</sup> Optional, if mutual authentication is enabled by the server endpoint

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

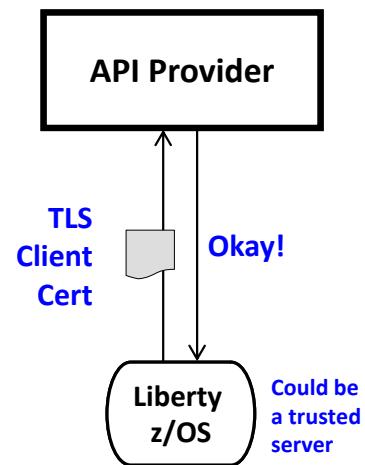
Several different ways this can be accomplished:

## Basic Authentication



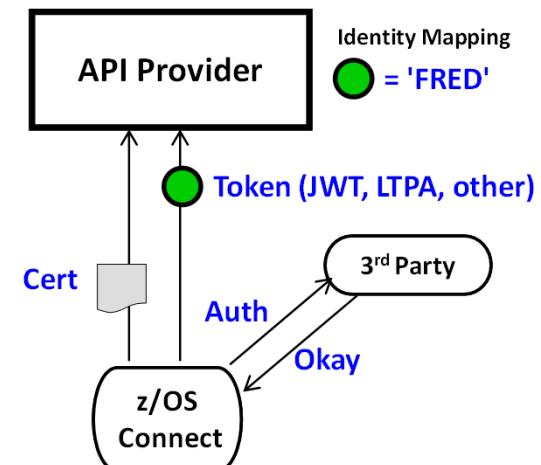
zCEE supplies ID/PW or  
ID/Passticket

## Client Certificate



Server prompts for certificate  
zCEE supplies certificate

## Third Party Authentication



zCEE authenticates to 3<sup>rd</sup> party sever  
zCEE receives a trusted 3<sup>rd</sup> party token  
Token flows to API Provider

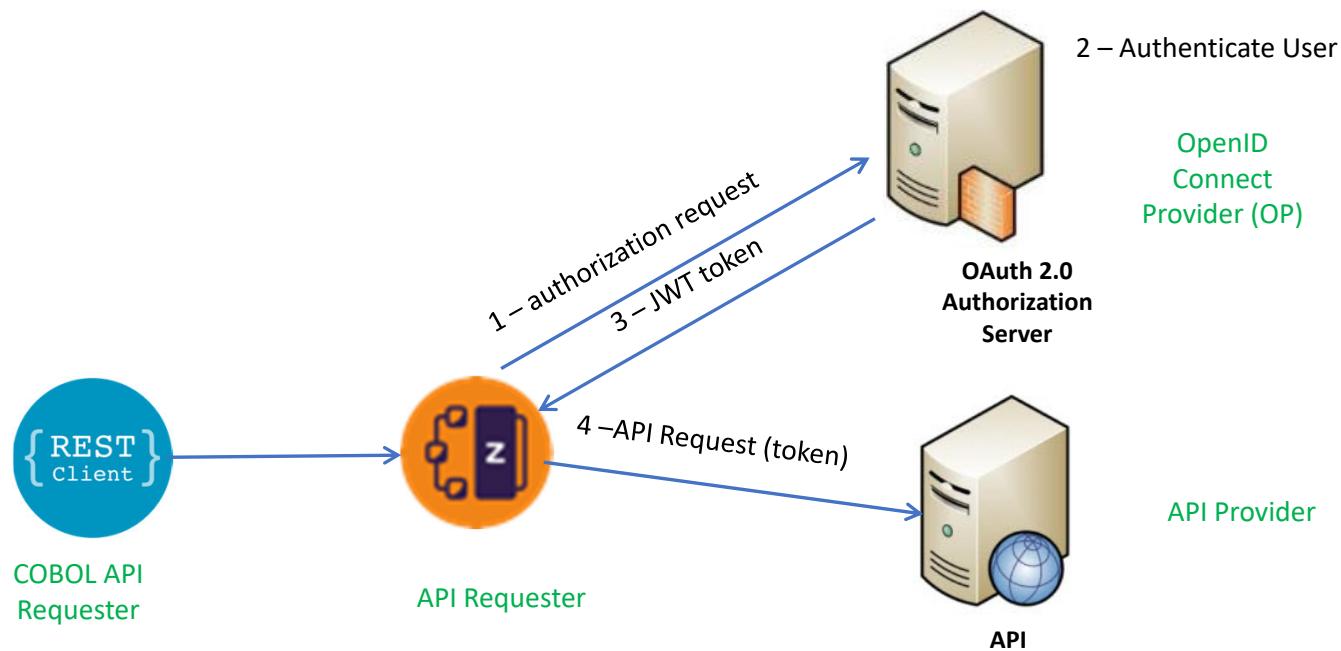
# **z/OS Connect API Requester - Token Support**



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.
  - o When you need to specify the HTTP verb that is used in the request to the authentication server
  - o When you need to specify how the JWT is returned from the authentication server (for example, in an HTTP header or in a custom field in a JSON response message).
  - o When you need to use a custom header name for sending the JWT to the request endpoint.
3. Use the locally generated JWT support when you need to send a JWT that is generated by the z/OS Connect EE server.

# z/OS Connect OAuth Flow for API requester



## Grant Types:

- client\_credentials
- password



# OAuth Grant Types Supported by z/OS Connect

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

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

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

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



# Configuring OAuth support – BAQRINFO copy book

```

wg31 master
File Edit Settings View Communication Actions Window Help
Menu Utilities Compilers Help
BROWSE ZCEE30.SBAQCDOB(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.
  A3 BAQ-REQUEST-TINFO-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

```

**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



# Obtaining a JWT using request parameters

The image displays two terminal windows from the IBM i environment, labeled 'wg31 master' and 'wg31 base'. The 'wg31 master' window shows host API definitions for OAuth parameters, while the 'wg31 base' window shows the assembly language code for generating a JWT.

**wg31 master**

```
File Edit Settings View Communication Actions Window Help
Menu Utilities Compilers Help
BROWSE ZCEE30.SBAQCOB(BAQHCONC) Line 0000000020 Col 001 080
Command ==>
* 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)
```

**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
000003 01 JWT-USER PIC X(10) VALUE 'myUsername'.
000004 01 JWT-PSWD PIC X(10) VALUE 'myPassword'.
000005 01 JWT-SCOPE PIC X(10) VALUE 'myScope'.
* Send JWT credentials to z/OS Connect
MOVE BAQR-OAUTH-USERNAME TO
BAQ-REQ-PARM-NAME OF BAQ-REQ-PARMS(1)
SET BAQ-REQ-PARM-ADDRESS OF
BAQ-REQ-PARMS(1) TO ADDRESS OF JWT-USER
MOVE LENGTH OF JWT-USER TO
BAQ-REQ-PARM-LENGTH OF BAQ-REQ-PARMS(1)
MOVE BAQR-OAUTH-PASSWORD TO
BAQ-REQ-PARM-NAME OF BAQ-REQ-PARMS(2)
SET BAQ-REQ-PARM-ADDRESS OF
BAQ-REQ-PARMS(2) TO ADDRESS OF JWT-PSWD
MOVE LENGTH OF JWT-PSWD TO
BAQ-REQ-PARM-LENGTH OF BAQ-REQ-PARMS(2)
MOVE BAQR-OAUTH-SCOPE TO
BAQ-REQ-PARM-NAME OF BAQ-REQ-PARMS(3)
SET BAQ-REQ-PARM-ADDRESS OF
BAQ-REQ-PARMS(3) TO ADDRESS OF JWT-SCOPE
MOVE LENGTH OF JWT-SCOPE TO
BAQ-REQ-PARM-LENGTH OF BAQ-REQ-PARMS(3)
* Call the API endpoint using BAQEXEC
***** Bottom of Data *****
----- 400 Line(s) not Displayed
***** **** Bottom of Data *****
----- 400 Line(s) not Displayed
```



# 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/token1
    basicAuthRef="tokenCredential" 2
    sslCertsRef="OutboundSSLSettings" />

<zosconnect_authData id="tokenCredential" 2
    user="zCEEclient" password="secret"/>
```

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

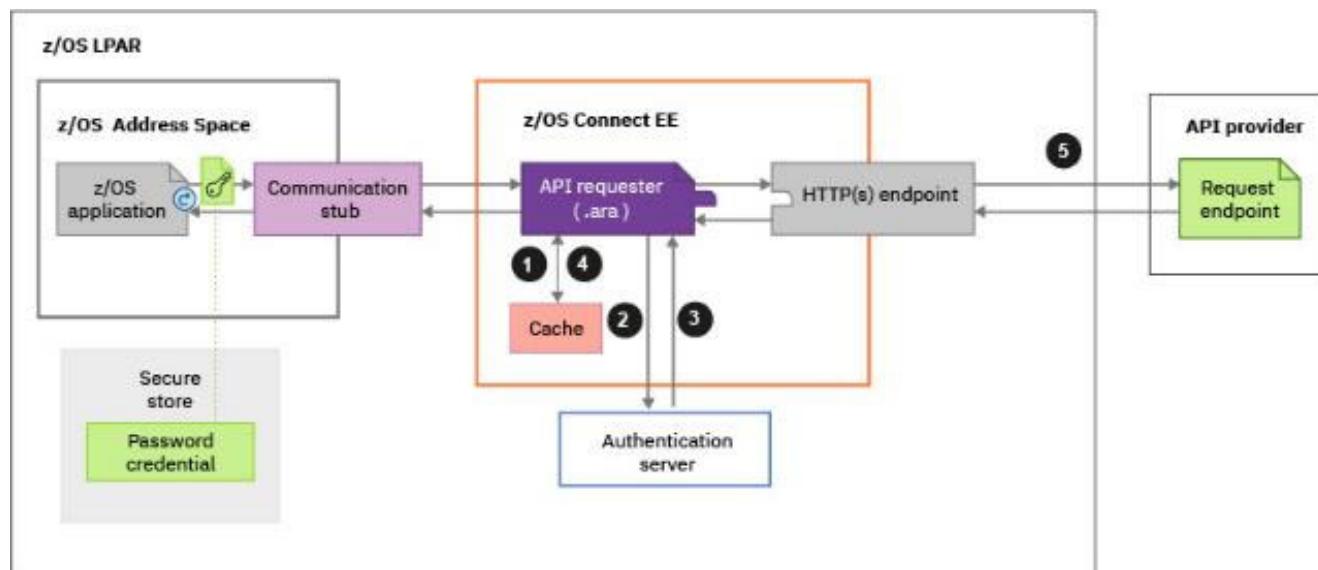
<sup>1</sup>See URL [https://www.ibm.com/support/knowledgecenter/SS7K4U\\_liberty/com.ibm.websphere.wlp.zseries.doc/ac/twlp\\_oidc\\_token\\_endpoint.html](https://www.ibm.com/support/knowledgecenter/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ac/twlp_oidc_token_endpoint.html)

<sup>2</sup> These credentials can be specified by the application

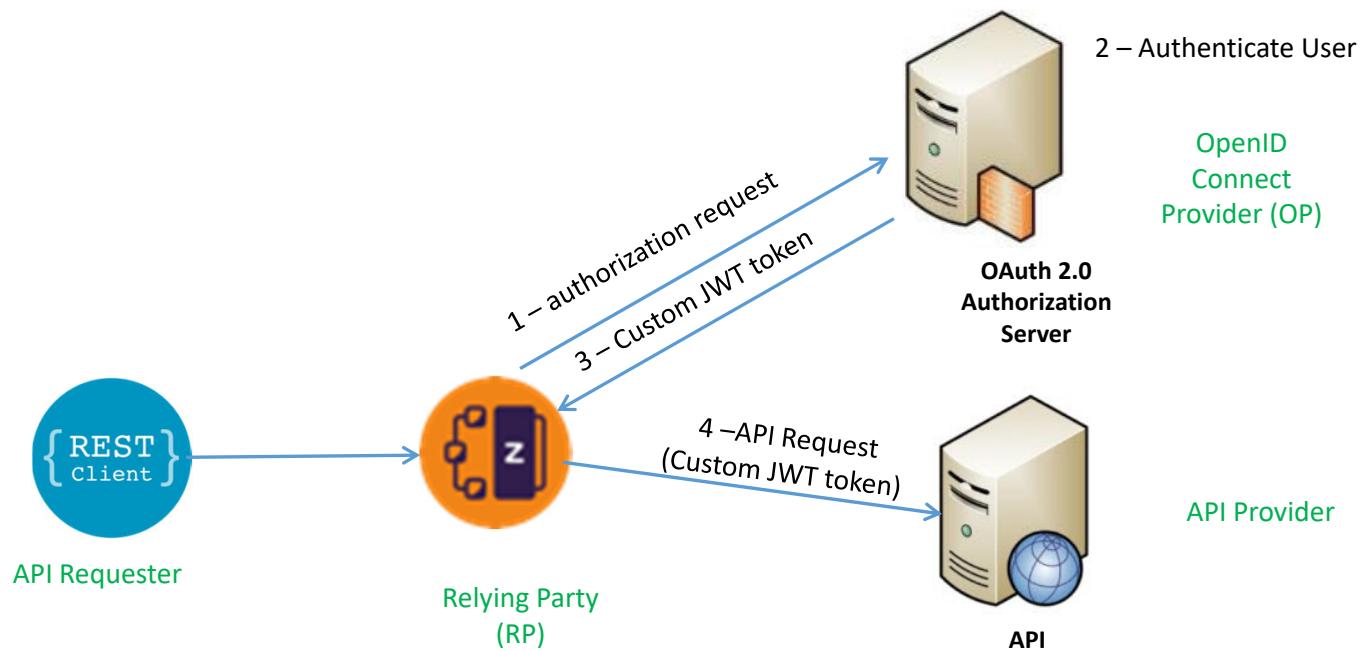


## Calling an API with using a JWT custom flow

- ❑ 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.



# z/OS Connect OAuth Custom Flow





# API Requester – JWT Custom flow

```
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.



# API Requester – JWT Custom flow

WG31 - 3270

Menu Utilities Compilers Help

BROWSE ZCEE30.SBAQCDB(BAQHCONC)

Line 0000000020 Col 001 080

Command ==>

```

* 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-TOKEN-USERNAME            PIC X(22)
  VALUE 'BAQHAPI-Token-Username'.
77 BAQR-TOKEN-PASSWORD             PIC X(22)
  VALUE 'BAQHAPI-Token-Password'.
77 BAQR-TOKEN-CUSTOM-PARMS        PIC X(25)
  VALUE 'BAQHAPI-Token-CustomParms'.
77 BAQR-TOKEN-CUSTOM-HEADERS      PIC X(27)
  VALUE 'BAQHAPI-Token-CustomHeaders'.

* 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)
  VALUE 'BAQHAPI-Server-Host'.
77 BAQZ-SERVER-PORT                PIC X(19)
  VALUE 'BAQHAPI-Server-Port'.
77 BAQZ-SERVER-TIMEOUT              PIC X(22)
  VALUE 'BAQHAPI-Server-Timeout'.
77 BAQZ-SERVER-USERNAME             PIC X(23)
  VALUE 'BAQHAPI-Server-Username'.

```

MA A

Connected to remote server/host wg31 using lu/pool TCP00112 and port 23 | Adobe PDF or

WG31 - 3270

File Edit Edit\_Settings Menu Utilities Compilers Test Help

EDIT JOHNSON.ZCEE.SOURCE(BAQZUSER) - 01.01 Columns 00001 00072

Command ==>

```

***** **** Top of Data ****
==MSG> -CAUTION- Data contains invalid (non-display) characters. Use command
==MSG>     ==> FIND P. to position cursor to these
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID. HBRMINM.
000003 ENVIRONMENT DIVISION.
000004 DATA DIVISION.
000005 WORKING-STORAGE SECTION.
000006 01 MY-USER PIC {10} VALUE '/myUsername'.
000007 01 MY-PSWD PIC {10} VALUE '/myPassword'.
000008 ...
000009 PROCEDURE DIVISION.
000010 ...
000011 ...
000012 ...
000013 ...
000014 MOVE BAQR-TOKEN-USERNAME TO
000015   BAQ-ZCON-PARM-NAME OF BAQ-ZCON-PARMS(1).
000016 SET BAQ-ZCON-PARM-ADDRESS OF BAQ-ZCON-PARMS(1) TO
000017   address of MY-USER.
000018 MOVE LENGTH OF MY-USER TO
000019   BAQ-ZCON-PARM-LENGTH(1) OF BAQ-ZCON-PARMS(1).
000020 ...
000021 MOVE BAQR-TOKEN-PASSWORD TO
000022   BAQ-ZCON-PARM-NAME OF BAQ-ZCON-PARMS(2).
000023 SET BAQ-ZCON-PARM-ADDRESS OF BAQ-ZCON-PARMS(2) TO
000024   ADDRESS OF MY-USER.
000025 MOVE LENGTH OF MY-USER TO
000026   BAQ-ZCON-PARM-LENGTH(1) OF BAQ-ZCON-PARMS(2).
***** **** Bottom of Data ****

```

MA A

Connected to remote server/host wg31 using lu/pool TCP00112 and port 23 | Adobe PDF on Documents\\*.pdf 05/009



# Configuring JWT Custom flow

```
<zosconnect_endpointConnection id="cscvincAPI"
    host="http://wq31.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://wq31.washington.ibm.com:59443/oidc/endpoint/OP/token1
    basicAuthRef="tokenCredential" 2
    sslCertsRef="OutboundSSLSettings" />

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

<sup>1</sup>See URL [https://www.ibm.com/support/knowledgecenter/SS7K4U\\_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp\\_oidc\\_token\\_endpoint.html](https://www.ibm.com/support/knowledgecenter/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_oidc_token_endpoint.html)

<sup>2</sup> These credentials can be specified by the application

# Configuring Custom JWT flow



Request Token Example 1

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

Response Token

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

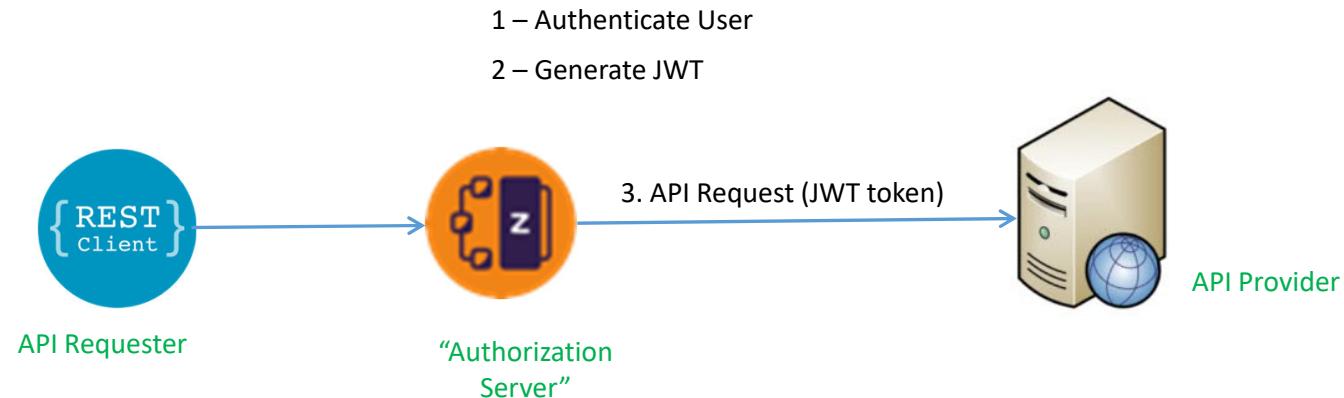
Response Token Example 2

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

Response Token

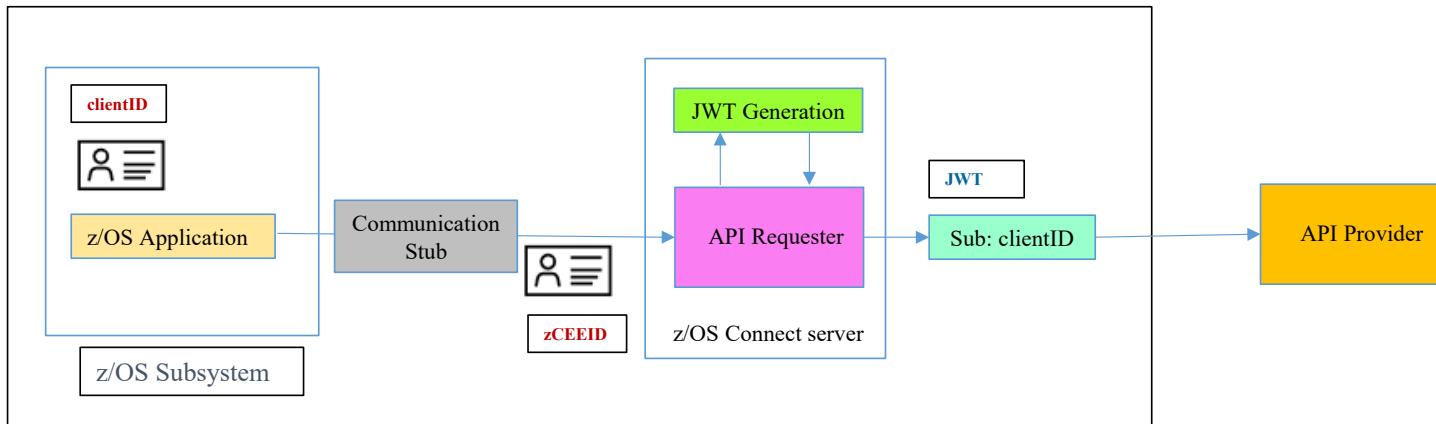
```
<tokenResponse  
    tokenLocation="body" responseFormat="JSON" tokenPath=".tokenname" />
```

# z/OS Connect JWT Generation – V3.0.43





# 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 <b><i>zCEEID</i></b> and checks whether <b><i>zCEEID</i></b> is a surrogate of <b><i>clientID</i></b> . If <b><i>zCEEID</i></b> is a surrogate of <b><i>clientID</i></b> , the server further checks whether <b><i>clientID</i></b> has the authority to invoke an API requester; otherwise, a BAQR7114E message occurs.
	ASSERT_ONLY	Identity assertion is enabled. The zCEE server authenticates <b><i>zCEEID</i></b> and directly checks whether <b><i>clientID</i></b> has the authority to invoke an API requester
false	ASSERT_SURROGATE	Identity assertion is enabled. The zCEE server checks whether <b><i>clientID</i></b> 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 <b><i>clientID</i></b> has the authority to invoke an API requester



# 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*

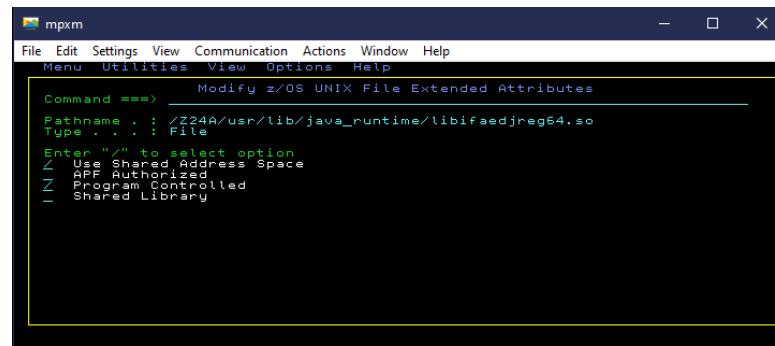
```
RDEFINE SURROGAT clientID.BAQASSRT UACC(NONE) OWNER(SYS1)  
PERMIT clientID.BAQASSRT CLASS(SURROGAT) ACCESS(READ) ID(zCEEID)
```

*OR*

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

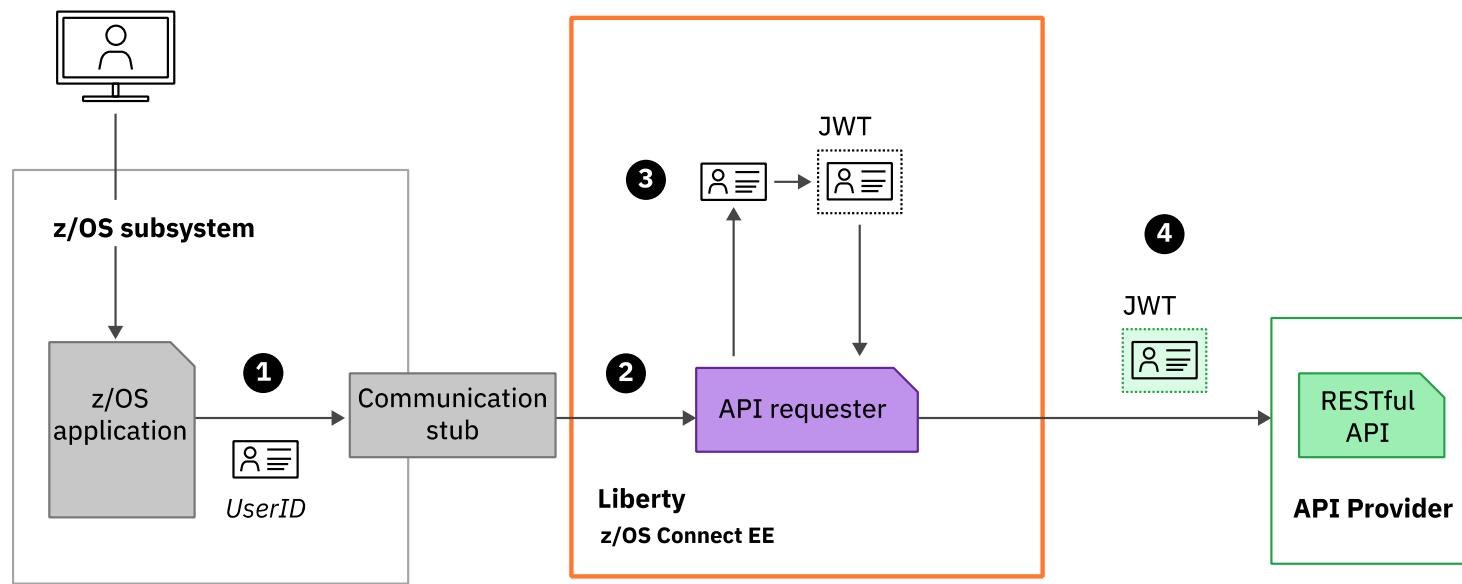
- *Enable the program control bit for Java shared object ifaedjreg64*

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





# 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



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

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

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

# 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"/>
```



# 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.



# Liberty Wildfire Github Site

<https://ibm.biz/BdPRGD>

The screenshot shows a GitHub repository page. At the top, there's a navigation bar with a file icon, a search bar containing "ibm-wsc/zCONNEE-Wildfire-Workshop", and a plus sign icon. Below the bar is a header with back, forward, refresh, and search icons, followed by the URL "https://github.com/ibm-wsc/zCONNEE-Wildfire-Workshop". To the right are star, message, fork, and settings icons.

The main content area displays a list of files and folders:

File/Folder	Description	Last Updated
archive	Add files via upload	last year
containers	Delete containers/readme	9 months ago
misc	Add files via upload	4 months ago
Accessing REST APIs from zOS usin...	Add files via upload	4 days ago
Accessing zOS resources with REST ...	Add files via upload	9 months ago
README.md	Update README.md	3 years ago
WebSphere Liberty on zOS Managi...	Add files via upload	25 minutes ago
WebSphere Liberty on zOS Configu...	Add files via upload	25 minutes ago
WebSphere Liberty on zOS Introdu...	Add files via upload	25 minutes ago
zOS Connect EE V3 Advanced Topic...	Add files via upload	3 years ago
ZOS Connect EE V3 Getting Started....	Add files via upload	3 years ago

A red box highlights the first five items in the list: README.md, WebSphere Liberty on zOS Management, WebSphere Liberty on zOS Configuration, WebSphere Liberty on zOS Introduction, and zOS Connect EE V3 Advanced Topics.

Below the file list is a "README" section with a "Edit" button. It contains the following text:

This repository contains material from the z/OS Connect EE Wildfire workshops run by the IBM Washington Systems Center. It should be referenced frequently for updates to the presentations, exercises, samples and other material.

On the right side of the page, there are sections for "Packages", "Contributors" (with entries for emitchj and Jbrefach), "Deployments" (with a entry for github-pages), and "Languages" (COBOL 100.0%).

At the bottom, there's a footer with the GitHub logo and copyright information: "© 2024 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information".

Thank you for listening and your questions.

[mitchj@us.ibm.com](mailto:mitchj@us.ibm.com)

© 2017, 2025 IBM Corporation  
Slide 131



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.

## **Miscellaneous Odds and Ends**



## Tech-Tip: RACF resources for using PassTickets

- ❑ A PTKTDATA resource is defined using the *appName* assigned to the target subsystem:

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

Where:

*appName* 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.appName.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)
```



# 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 token with the identity and password encoded or formatted using Base64.
  - Examples using the API Explorer feature , cURL, and a Java client.

The screenshot shows the IBM API Explorer interface for the 'cscvinc' service. On the left, there's a list of operations: POST /cscvinc/employee, DELETE /cscvinc/employee/{employee}, and GET /cscvinc/employee/{employee}. The 'GET' operation is selected. On the right, there's a 'Response Class (Status 200)' section with a JSON example. Below it, there's a 'Parameters' section with an 'Authorization' field containing 'Basic dXNlcnJvZG9tZWQ=' and an 'employee' field containing '000050'. A red circle highlights the 'Authorization' field. To the right of this, a sign-in dialog box is displayed with the URL 'mpz3.washington.ibm.com:9443'. The dialog asks for 'Username' (user1) and 'Password' (\*\*\*\*\*). A red box surrounds the entire sign-in dialog. At the bottom, there's a 'Try it out!' button.

The screenshot shows a Windows Command Prompt window and a Java code editor. The Command Prompt window shows a curl command being run: `c:\>curl -X GET --user FRED --insecure https://mpz3.washington.ibm.com:9443/cscvinc/employee/111111 {"cscvincSelectServiceOperationResponse": {"cscvincContainer": {"response": {"CE1BRESP": 0, "CE1BRESP2": 0, "USERID": "CICUSER", "filea": {"employeeNumber": "111111", "name": "C. BAKER", "address": "OTTAWA, ONTARIO", "phoneNumber": "51212003", "date": "26 11 81", "amount": "$0011.00"} }}}`. A red circle highlights the 'curl' command. The Java code editor shows a snippet of code for a 'ZeeGetjava' class. It includes a 'BasicAuthenticator' class and a 'try' block that constructs a URL, opens a connection, sets the Content-Type to 'application/json', encodes the credentials 'Basic dXNlcnJvZG9tZWQ=', and adds an 'Authorization' header with the encoded value. A red circle highlights the 'Authorization' header line.

## 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 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 known as "shared secret" or "session key" (**symmetric encryption**)
15. Client and server will use this "shared secret" or "session key" to encrypt messages sent between the endpoints.

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

## 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.
```

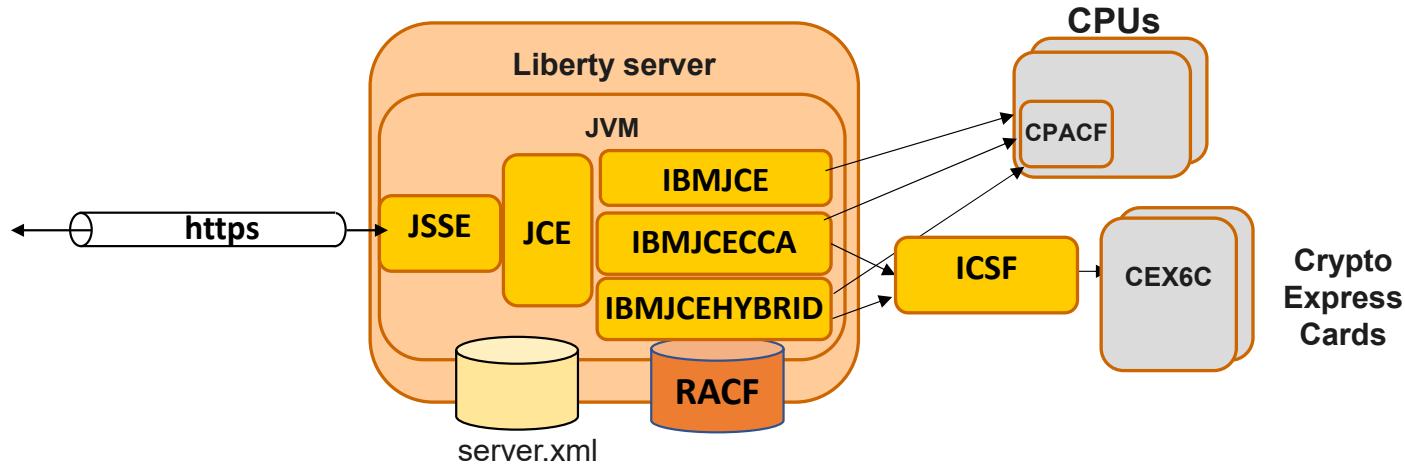
```
enum {  
    hello_request(0),  
    client_hello(1),  
    server_hello(2),  
    certificate(11),  
    server_key_exchange (12),  
    certificate_request(13),  
    server_hello_done(14),  
    certificate_verify(15),  
    client_key_exchange(16),  
    finished(20),  
    (255) }  
HandshakeType;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

\* 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

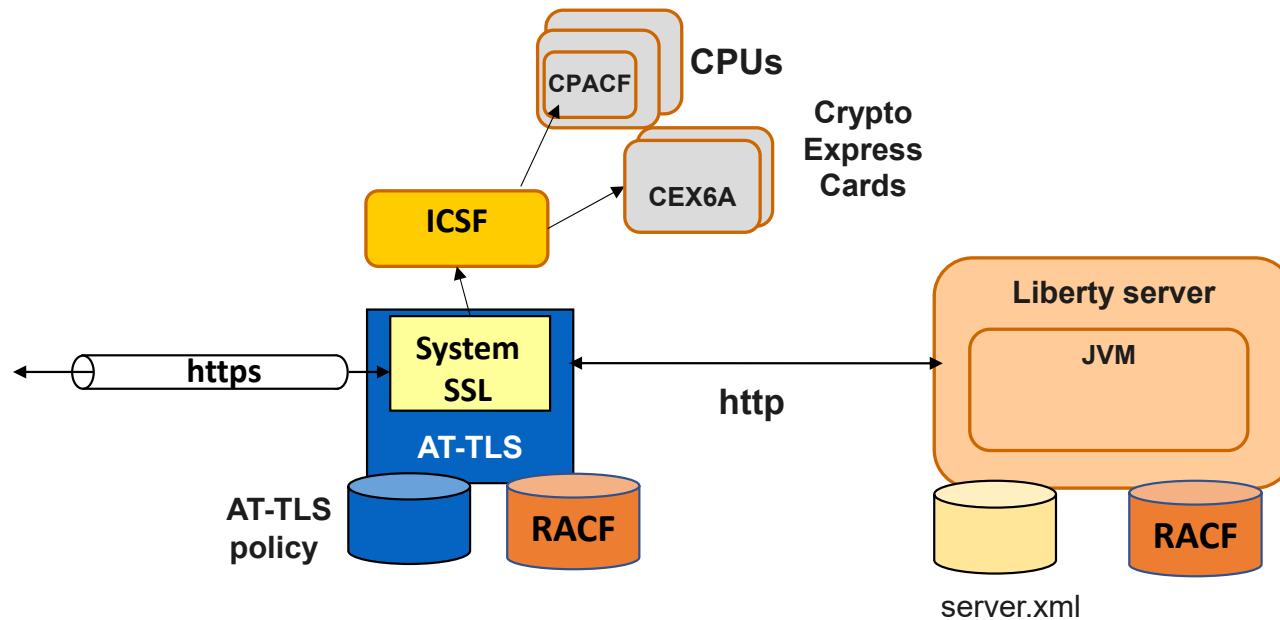


- z/OS Connect EE support for TLS is based on **Liberty** server support
- **Java Secure Socket Extension (JSSE)** API provides framework and Java implementation of TLS protocols used by Liberty HTTPS support
- **Java Cryptography Extension (JCE)** is standard extension to the Java Platform that provides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



# Liberty and using AT-TLS

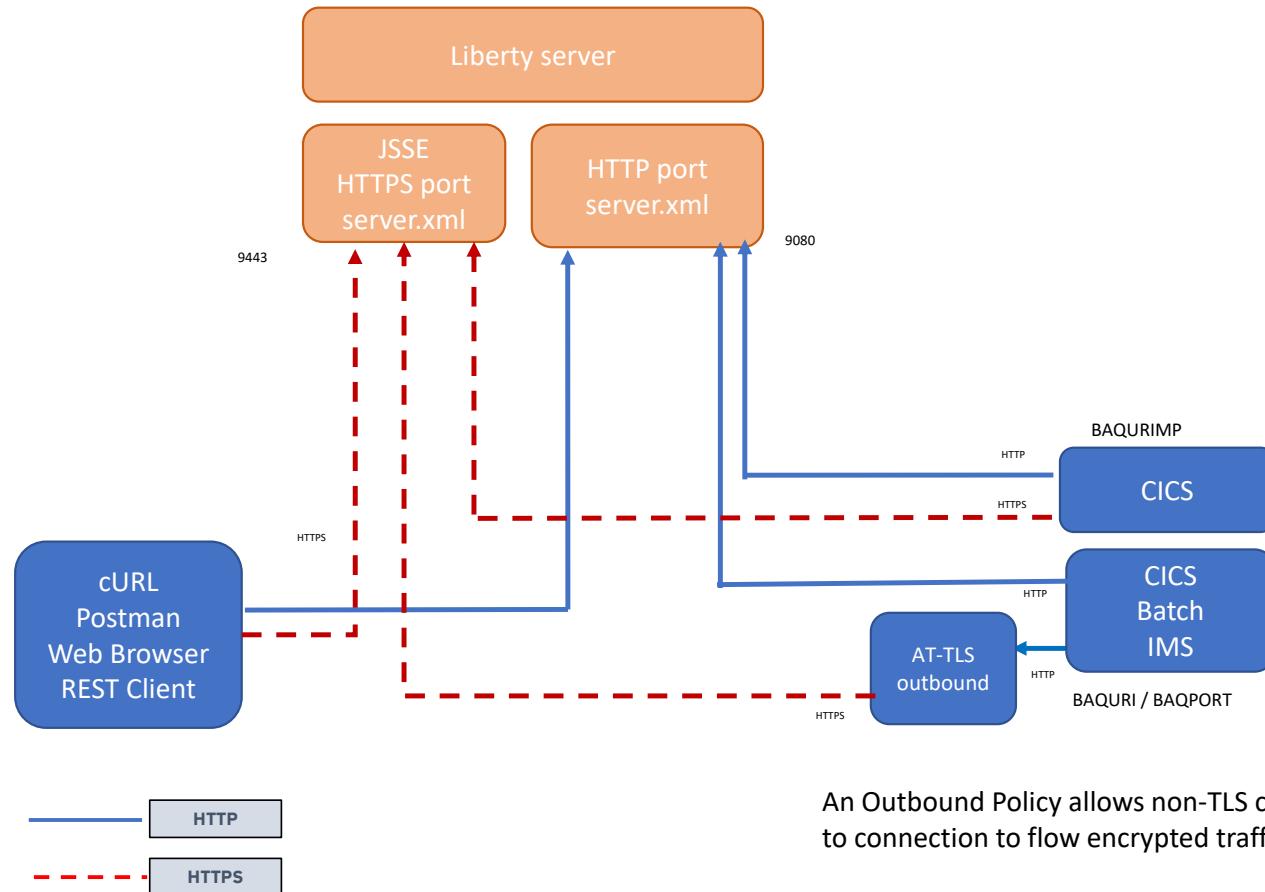
The server XML configuration uses no HTTPS protocol, key rings or other JSSE attributes



- **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

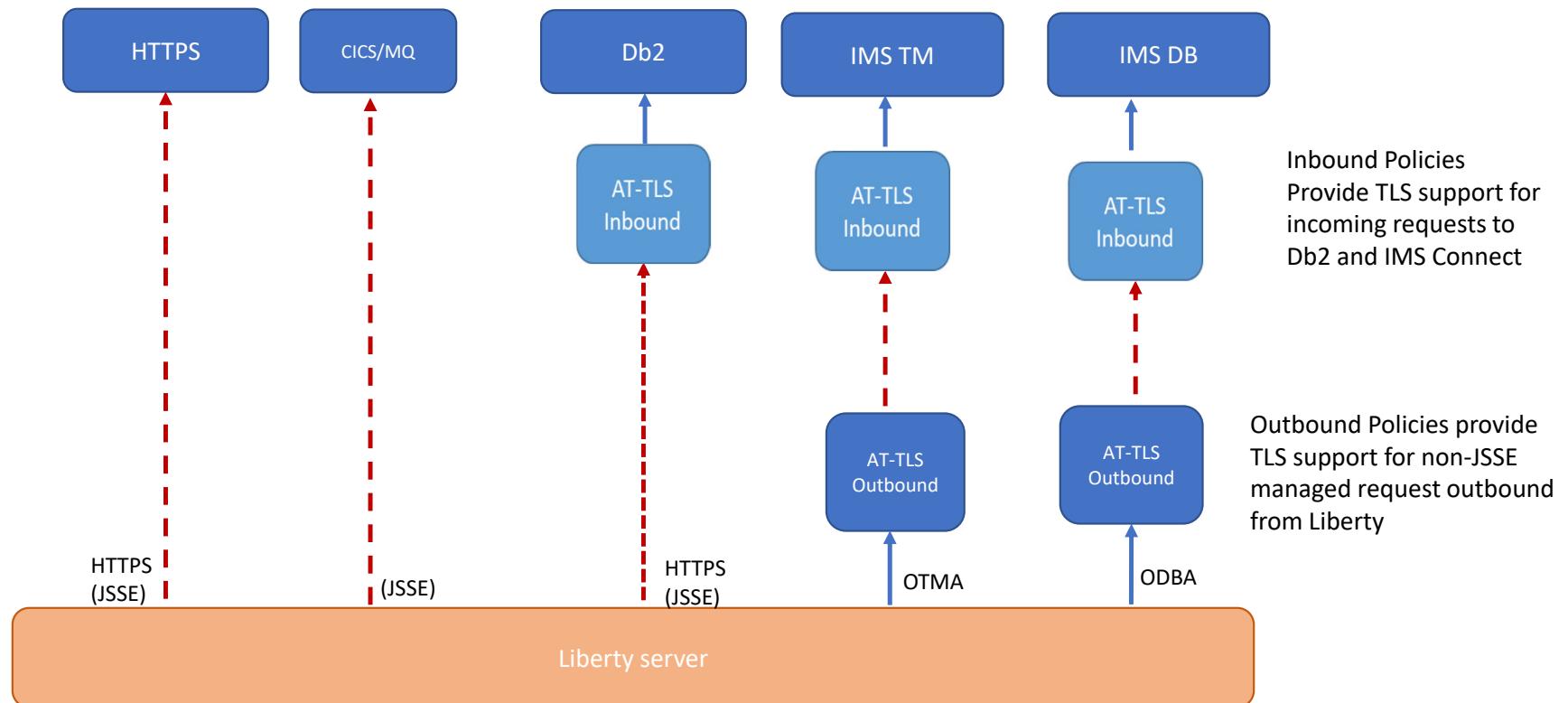


## Tech-Tip: TLS client encryption to a Liberty server scenarios





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





# 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

# 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 <a href="#">IBM SDK Policy files</a> topic in the <a href="#">Security Reference for IBM SDK, Java Technology Edition</a> 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	

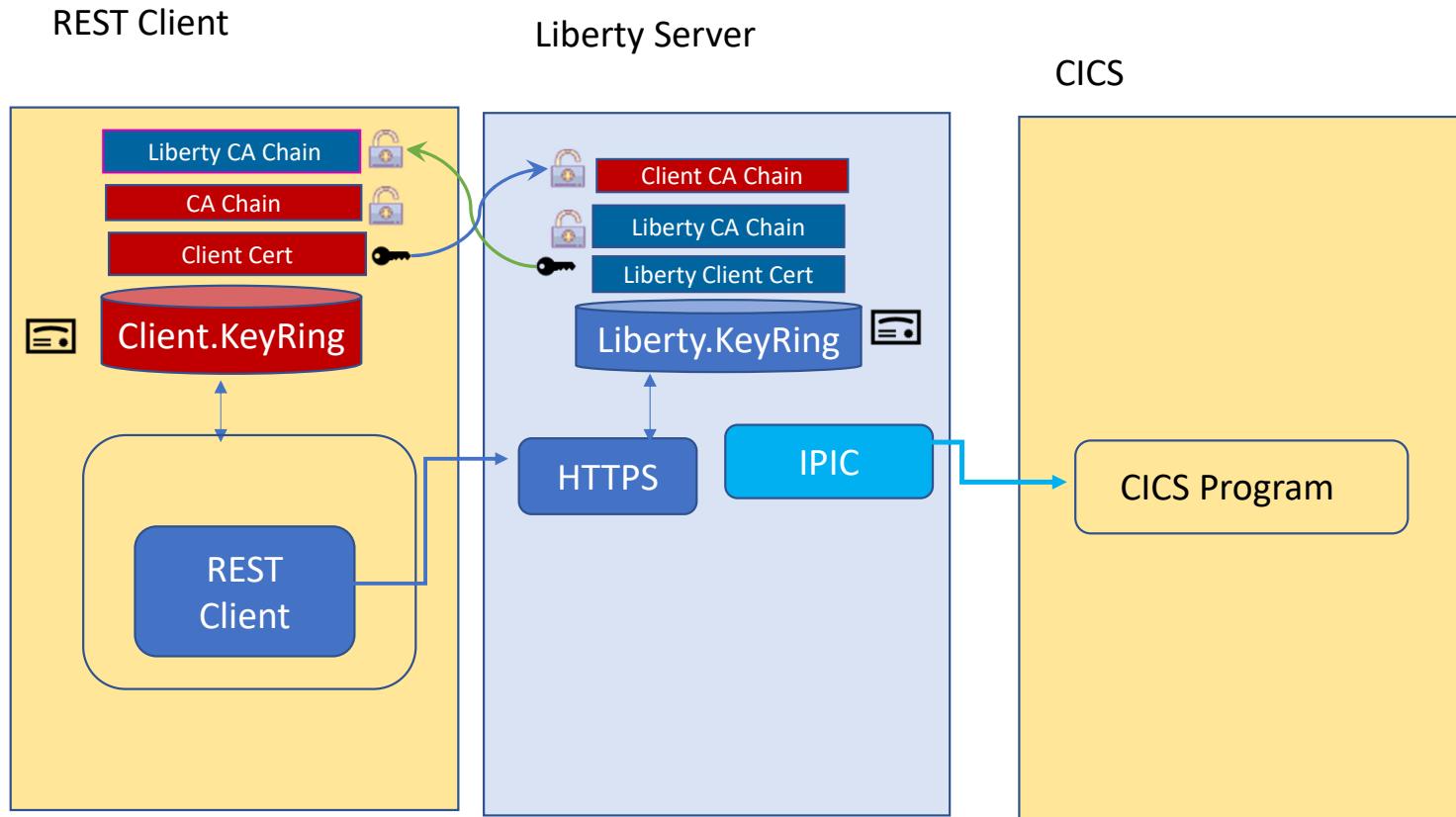
<https://www.ibm.com/docs/en/ibm-mq/9.3?topic=java-tls-cipherspecs-ciphersuites-in-mq-classes>

mitchj@us.ibm.com

© 2017, 2025 IBM Corporation  
Slide 144

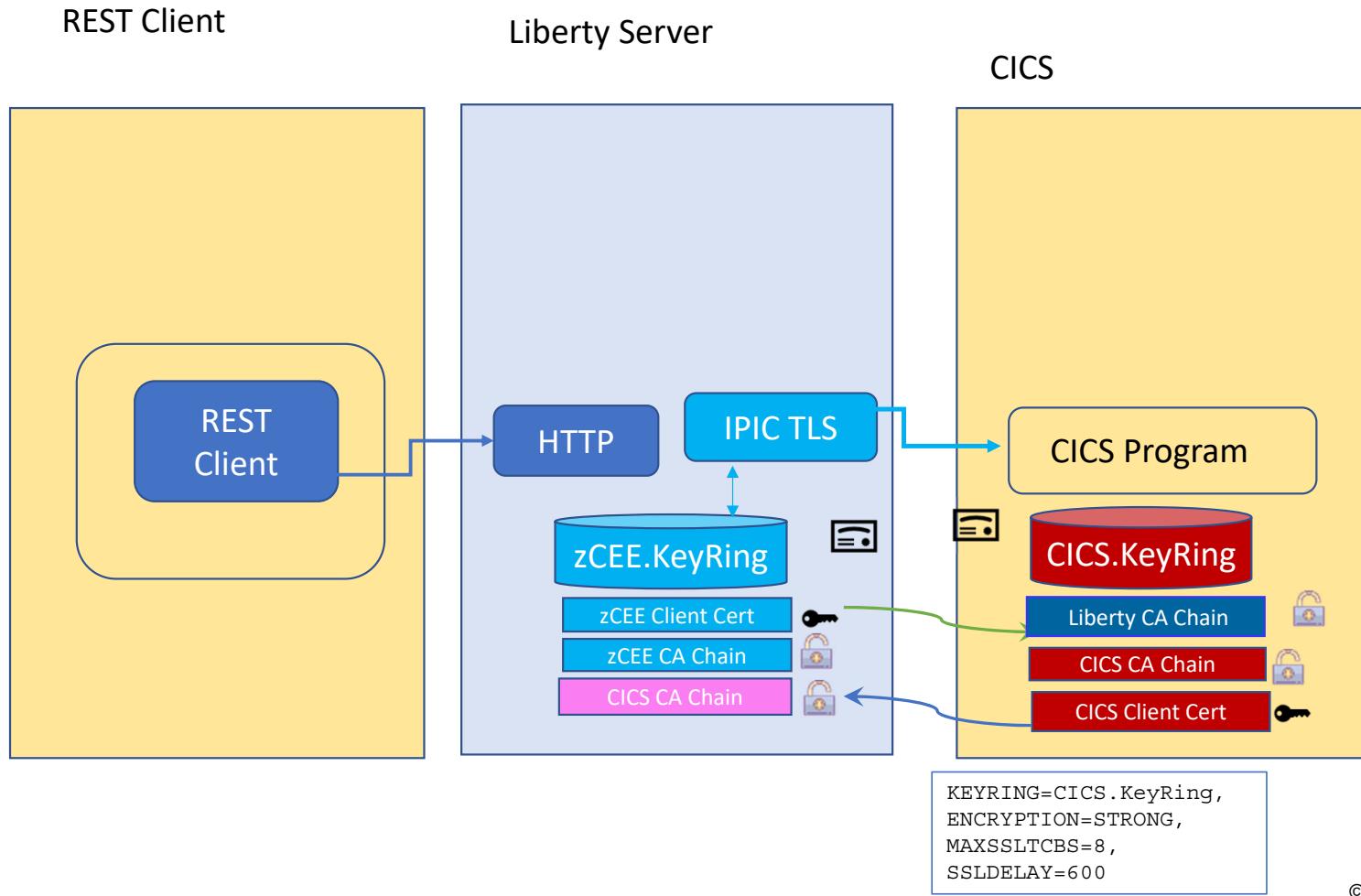


## TLS handshake between the client and the Liberty server



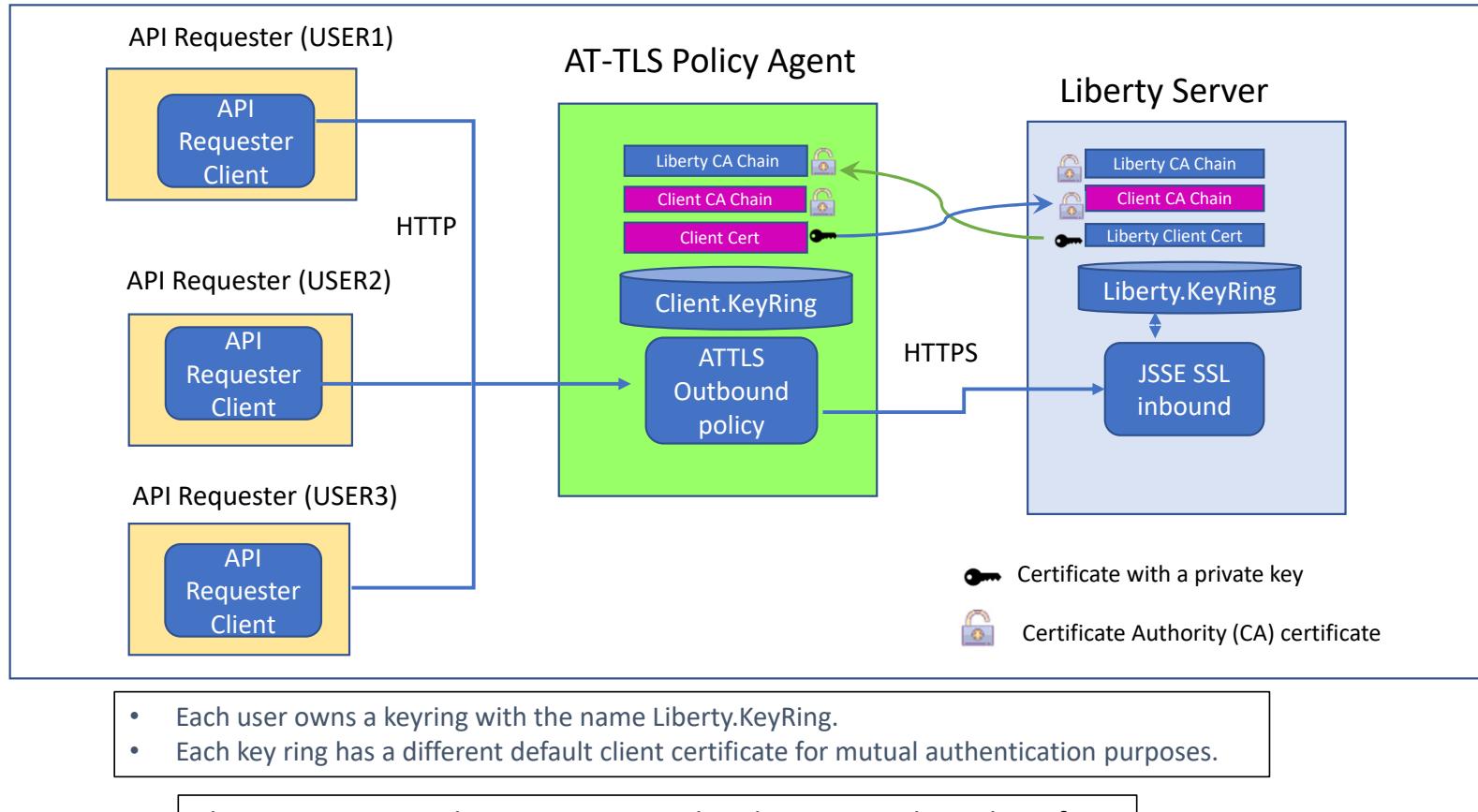


## TLS handshake between the Liberty server and the target endpoint



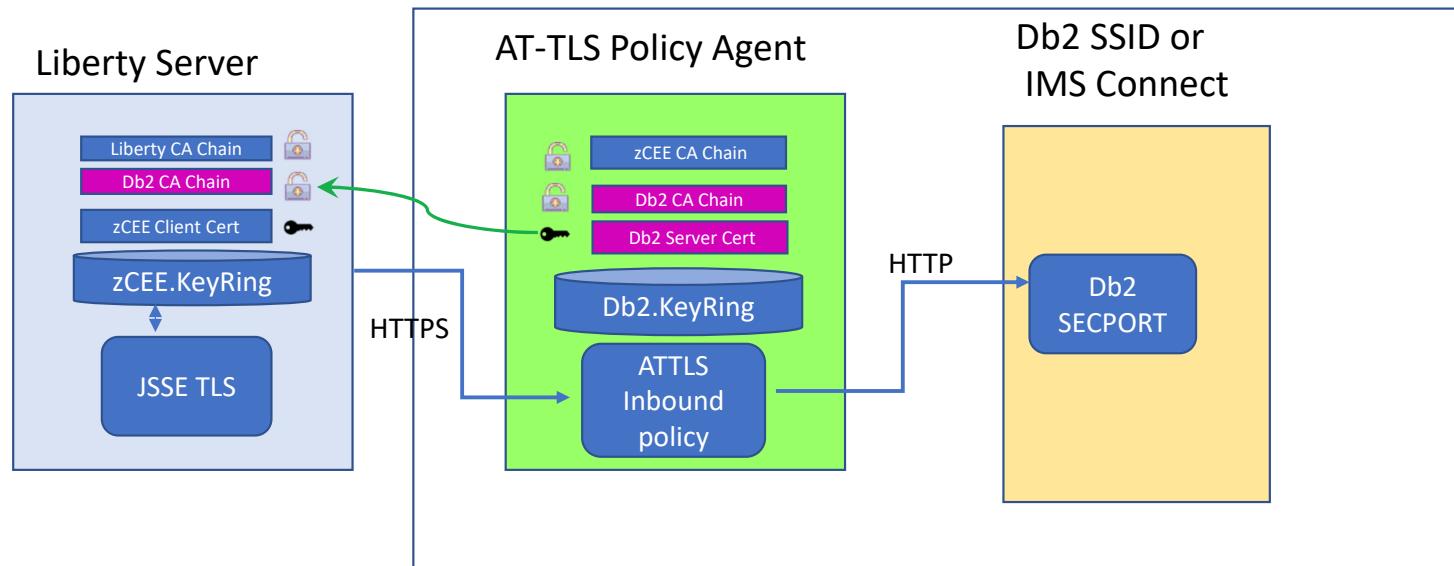
## AT-TLS - outbound policy handshake scenario

Use of a common key ring name for multiple client identities



## 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 unaware

🔑 Certificate with a private key

🔒 Certificate Authority (CA) certificate



## Tech/Tip: Let's explore a flow using a Liberty OpenID 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.



## 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.



## 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) to 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.



# 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).



## 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).

# 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')
```



# Tech-Tip: CICS IPCONN and TCPIPSERVICE resources for HA

CICS Specific TCPIPSERVICE - IPIC

```
TCpipservice : IPIC1
GROup       : SYSPGRP
Urm         ==> DFHISAIP
POrtnumber   ==> 01492
STatus      ==> Open
PROtocol    ==> IPic
TRansaction ==> CISS
Host        ==> ANY
Ipaddress   ==> ANY
SPeciftcpS  ==>
```

CICS Generic TCPIPSERVICE - IPICG

```
TCpipservice : IPICG1
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

<sup>1</sup>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.



## 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 *preferredSpecificHost* 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.

# 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:

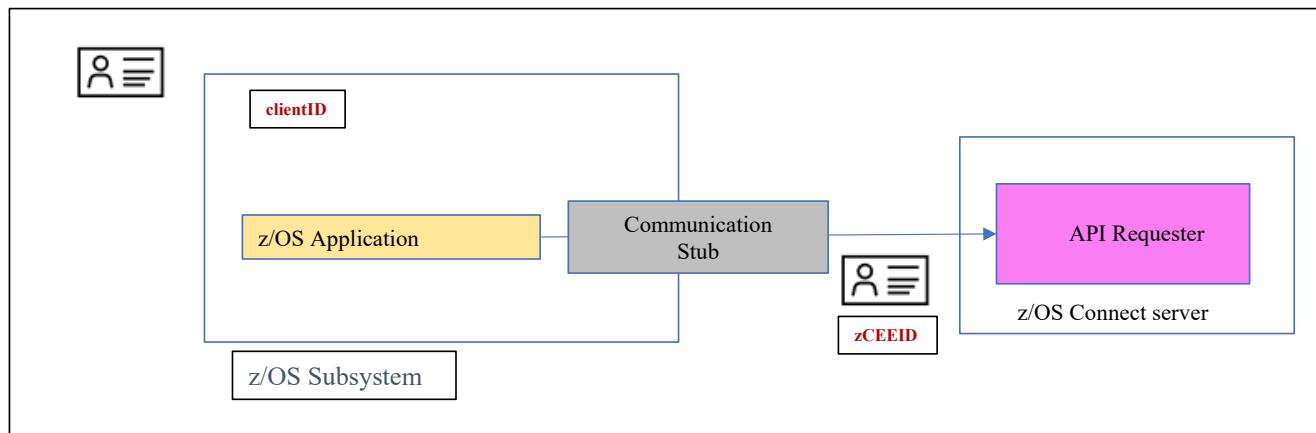
```
{
  "StatusCode": 500,
  "StatusDescription": "Service zCEEService.selectEmployee discovery failed due to
SQLCODE=-551 SQLSTATE=42501, USER2 DOES NOT HAVE THE PRIVILEGE TO PERFORM OPERATION EXECUTE
PACKAGE ON OBJECT zCEEService.selectEmployee. Error Location:DSNLJACC:35"
}
```

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
```



# 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 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 <b><i>zCEEID</i></b> and checks whether <b><i>zCEEID</i></b> has the authority to invoke an API requester.
	ASSERT_SURROGATE	Identity assertion is enabled. The zCEE server authenticates <b><i>zCEEID</i></b> and checks whether <b><i>zCEEID</i></b> is a surrogate of <b><i>clientID</i></b> . If <b><i>zCEEID</i></b> is a surrogate of <b><i>clientID</i></b> , the server further checks whether <b><i>clientID</i></b> has the authority to invoke an API requester; otherwise, a BAQR7114E message occurs.
	ASSERT_ONLY	Identity assertion is enabled. The zCEE server authenticates <b><i>zCEEID</i></b> and directly checks whether <b><i>clientID</i></b> 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 <b><i>clientID</i></b> 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 <b><i>clientID</i></b> 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>

```



# Configuring OAuth support – BAQRINFO copy book

```

wg31 master
File Edit Settings View Communication Actions Window Help
Menu Utilities Compilers Help
BROWSE ZCEE30.SBAQC0B(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-TINFO-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.
04/015
Connected to remote server/host wg31z using lu/pool TCP00145

```

**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



# Obtaining a JWT using request parameters

The image displays two terminal windows side-by-side, both titled "wg31 master".

**Terminal Window 1 (Left):**

```
BROWSE ZCEE30.SBAQCOB(BAQHCONC)
Command ==>
* 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)
```

**Terminal Window 2 (Right):**

```
EDIT USER1.ZCEE30.SOURCE(GETAPI) - 01.02
Command ==>
***** **** 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'.

* Send JWT credentials to z/OS Connect
MOVE BAQR-TOKEN-USERNAME TO
  BAQ-REQ-PARM-NAME OF BAQ-REQ-PARMS(1)
SET BAQ-REQ-PARM-ADDRESS OF
  BAQ-REQ-PARMS(1) TO ADDRESS OF JWT-USER
MOVE LENGTH OF JWT-USER TO
  BAQ-REQ-PARM-LENGTH OF BAQ-REQ-PARMS(1)
MOVE BAQR-TOKEN-PASSWORD TO
  BAQ-REQ-PARM-NAME OF BAQ-REQ-PARMS(2)
SET BAQ-REQ-PARM-ADDRESS OF
  BAQ-REQ-PARMS(2) TO ADDRESS OF JWT-PSWD
MOVE LENGTH OF JWT-PSWD TO
  BAQ-REQ-PARM-LENGTH OF BAQ-REQ-PARMS(2)

* Call the API endpoint using BAQEXEC
000023
000024
000025
000026
000027
```



# API Requester – JWT Custom flow

```
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.



# API Requester – JWT Custom flow

WG31 - 3270

Menu Utilities Compilers Help

BROWSE ZCEE30.SBAQCDB(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-TOKEN-USERNAME PIC X(22)
  VALUE 'BAQHAPI-Token-Username'.
77 BAQR-TOKEN-PASSWORD PIC X(22)
  VALUE 'BAQHAPI-Token-Password'.
77 BAQR-TOKEN-CUSTOM-PARMS PIC X(25)
  VALUE 'BAQHAPI-Token-CustomParms'.
77 BAQR-TOKEN-CUSTOM-HEADERS PIC X(27)
  VALUE 'BAQHAPI-Token-CustomHeaders'.

* 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)
  VALUE 'BAQHAPI-Server-Host'.
77 BAQZ-SERVER-PORT PIC X(19)
  VALUE 'BAQHAPI-Server-Port'.
77 BAQZ-SERVER-TIMEOUT PIC X(22)
  VALUE 'BAQHAPI-Server-Timeout'.
77 BAQZ-SERVER-USERNAME PIC X(23)
  VALUE 'BAQHAPI-Server-Username'.

```

MA A

Connected to remote server/host wg31 using lu/pool TCP00112 and port 23 | Adobe PDF or

WG31 - 3270

File Edit Edit\_Settings Menu Utilities Compilers Test Help

EDIT JOHNSON.ZCEE.SOURCE(BAQZUSER) - 01.01 Columns 00001 00072  
Command ==> Scroll ==> PAGE

```

***** **** Top of Data ****
==MSG> -CAUTION- Data contains invalid (non-display) characters. Use command
==MSG>      ==> FIND P. to position cursor to these
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID. HBRMINM.
000003 ENVIRONMENT DIVISION.
000004 DATA DIVISION.
000005 WORKING-STORAGE SECTION.
000006 01 MY-USER PIC {10} VALUE '/myUsername'.
000007 01 MY-PSWD PIC {10} VALUE '/myPassword'.
000008 ...
000009 PROCEDURE DIVISION.
000010 ...
000011 ...
000012 ...
000013 ...
000014 MOVE BAQR-TOKEN-USERNAME TO
000015   BAQ-ZCON-PARM-NAME OF BAQ-ZCON-PARMS(1).
000016 SET BAQ-ZCON-PARM-ADDRESS OF BAQ-ZCON-PARMS(1) TO
000017   address of MY-USER.
000018 MOVE LENGTH OF MY-USER TO
000019   BAQ-ZCON-PARM-LENGTH(1) OF BAQ-ZCON-PARMS(1).
000020 ...
000021 MOVE BAQR-TOKEN-PASSWORD TO
000022   BAQ-ZCON-PARM-NAME OF BAQ-ZCON-PARMS(2).
000023 SET BAQ-ZCON-PARM-ADDRESS OF BAQ-ZCON-PARMS(2) TO
000024   ADDRESS OF MY-USER.
000025 MOVE LENGTH OF MY-USER TO
000026   BAQ-ZCON-PARM-LENGTH(1) OF BAQ-ZCON-PARMS(2).
***** **** Bottom of Data ****

```

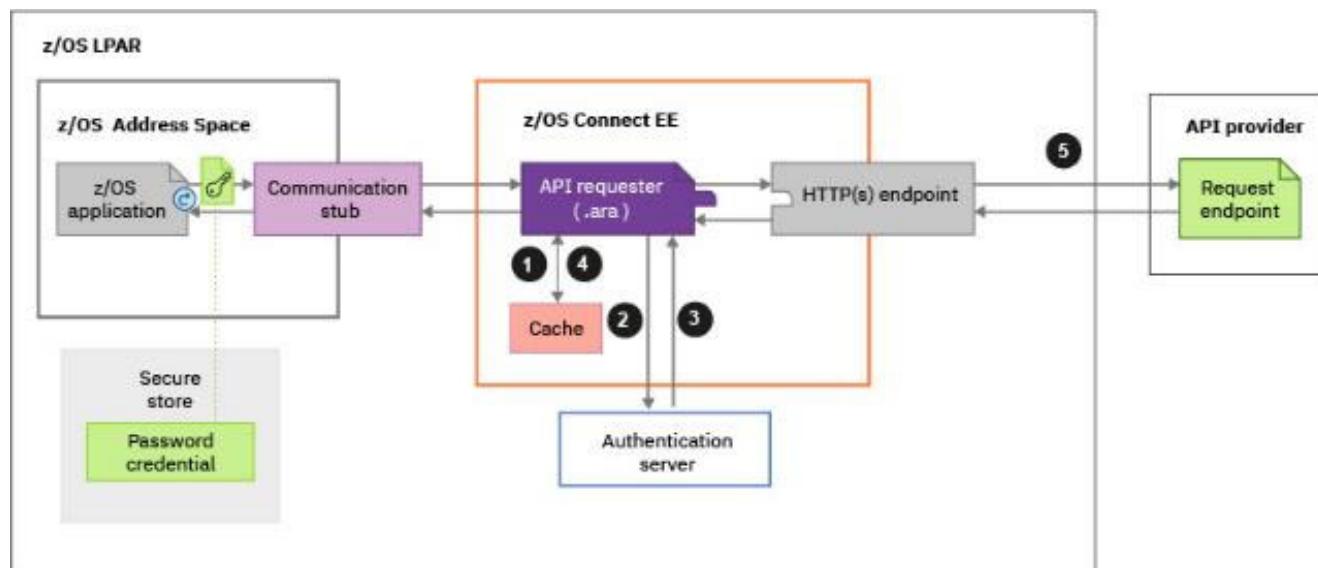
MA A

Connected to remote server/host wg31 using lu/pool TCP00112 and port 23 | Adobe PDF on Documents\\*.pdf 05/009

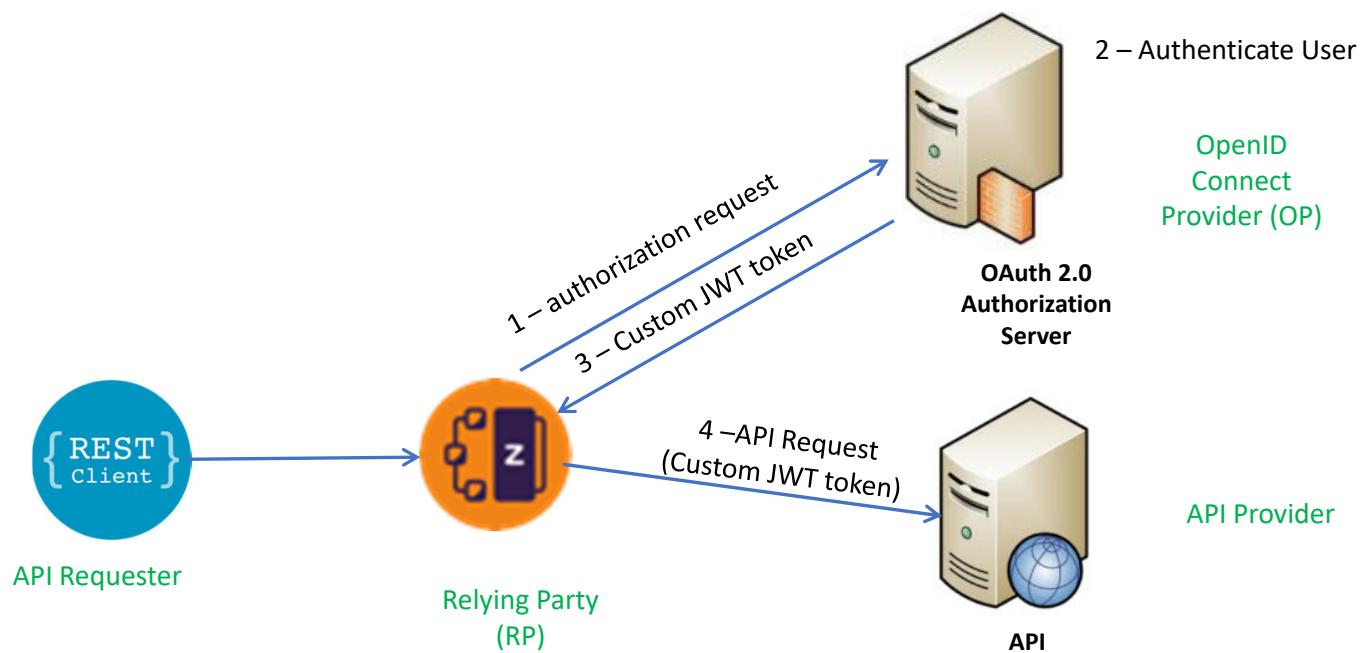


## Calling an API with using a JWT custom flow

- ❑ 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.



# z/OS Connect OAuth Custom Flow





# Configuring JWT Custom flow

```
<zosconnect_endpointConnection id="cscvincAPI"
    host="http://wq31.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://wq31.washington.ibm.com:59443/oidc/endpoint/OP/token1
    basicAuthRef="tokenCredential" 2
    sslCertsRef="OutboundSSLSettings" />

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

<sup>1</sup>See URL [https://www.ibm.com/support/knowledgecenter/SS7K4U\\_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp\\_oidc\\_token\\_endpoint.html](https://www.ibm.com/support/knowledgecenter/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_oidc_token_endpoint.html)

<sup>2</sup> These credentials can be specified by the application



# Configuring Custom JWT flow

## Request Token Example 1

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

## Response Token

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

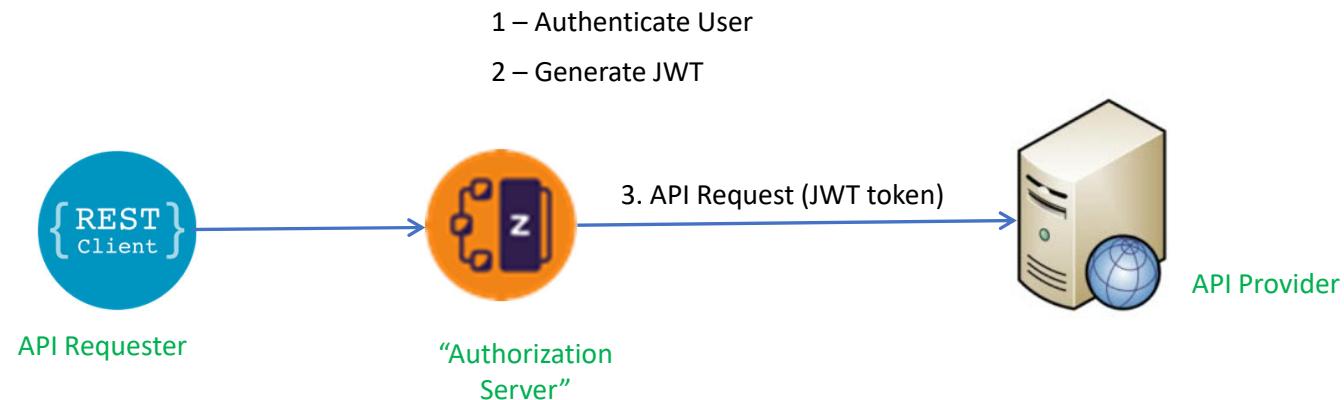
## Response Token Example 2

```
<tokenRequest credentialLocation="body"  
    requestMethod="POST"  
    // Use XML escaped characters in requestBody  
    requestBody="
```

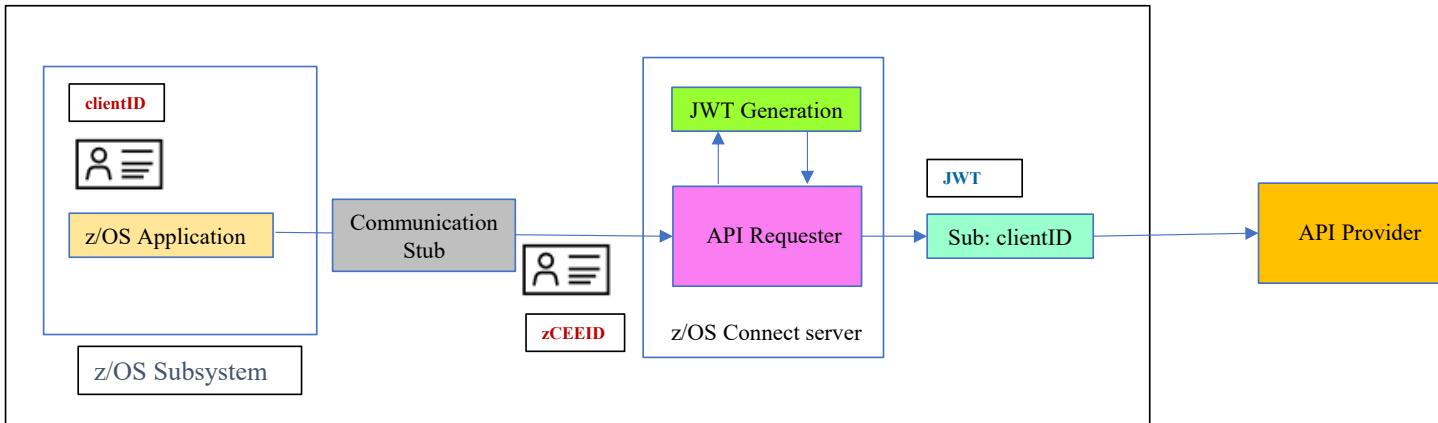
## Response Token

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

# z/OS Connect JWT Generation – V3.0.43



# 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.

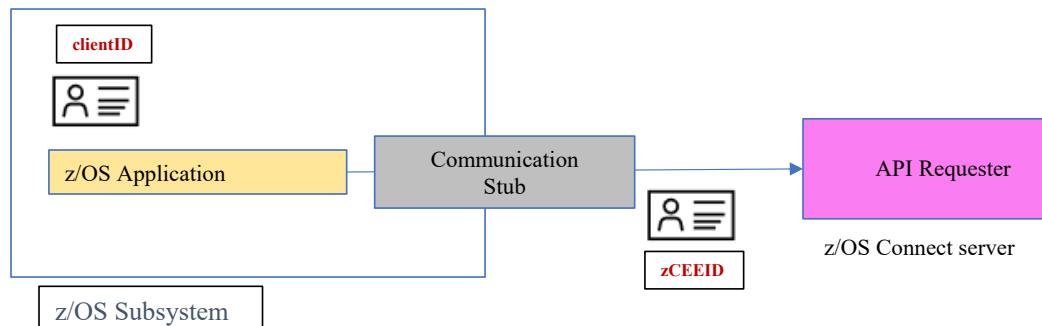
***clientID*** – the identity under which the z/OS application is executing.

- 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 <b><i>zCEEID</i></b> and checks whether <b><i>zCEEID</i></b> is a surrogate of <b><i>clientID</i></b> . If <b><i>zCEEID</i></b> is a surrogate of <b><i>clientID</i></b> , the server further checks whether <b><i>clientID</i></b> has the authority to invoke an API requester; otherwise, a BAQR7114E message occurs.
	ASSERT_ONLY	Identity assertion is enabled. The zCEE server authenticates <b><i>zCEEID</i></b> and directly checks whether <b><i>clientID</i></b> has the authority to invoke an API requester
false	ASSERT_SURROGATE	Identity assertion is enabled. The zCEE server checks whether <b><i>clientID</i></b> 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 <b><i>clientID</i></b> has the authority to invoke an API requester



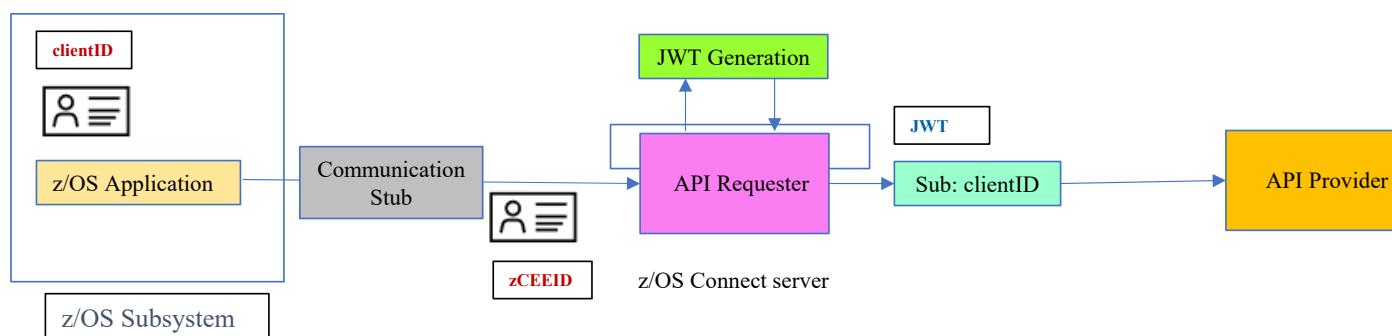
# API Requester - authentication with identity assertion and JWT generation



***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.

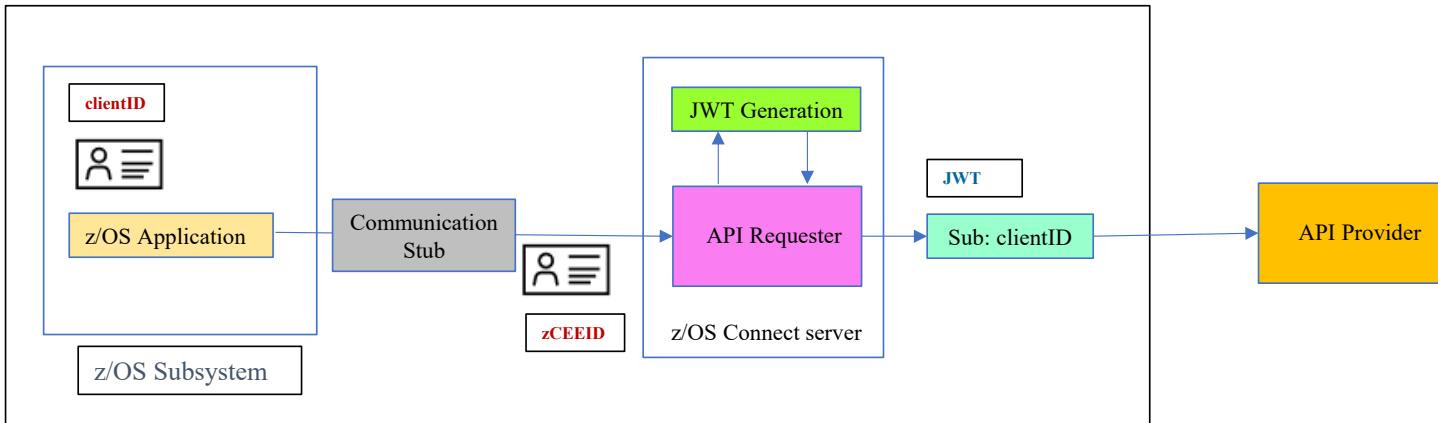
***clientID*** – the identity under which the z/OS application is executing.

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



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

# 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 <b><i>zCEEID</i></b> and checks whether <b><i>zCEEID</i></b> is a surrogate of <b><i>clientID</i></b> . If <b><i>zCEEID</i></b> is a surrogate of <b><i>clientID</i></b> , the server further checks whether <b><i>clientID</i></b> has the authority to invoke an API requester; otherwise, a BAQR7114E message occurs.
	ASSERT_ONLY	Identity assertion is enabled. The zCEE server authenticates <b><i>zCEEID</i></b> and directly checks whether <b><i>clientID</i></b> has the authority to invoke an API requester
false	ASSERT_SURROGATE	Identity assertion is enabled. The zCEE server checks whether <b><i>clientID</i></b> 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 <b><i>clientID</i></b> has the authority to invoke an API requester



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

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

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



## 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.