ACSP5000-1

# IBM Advanced Crypto Service Provider REST API

## Version 1.0

## Installation and Configuration Guide

## z/OS

## IBM Crypto Competence Center, Denmark

# Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectable rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

- **Trademarks**

  Java™ is a trademark of Oracle®

  IBM®, System z®, z/OS®, RACF® are registered trademarks of International Business Machines Corporation.

This edition applies to the IBM Distributed Key Management System (DKMS) product:

- IBM Advanced Crypto Service Provider REST API, (ACSP-REST-1-0)

and to all subsequent releases and modifications until otherwise indicated in new editions.

Comments may be addressed to your IBM representative or the IBM branch office serving your locality. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

# Contents

# About this Publication

This publication contains information on how to install, configure and operate the **IBM Advanced Crypto Service Provider REST API.**

The IBM Advanced Crypto Service Provider REST API (hereafter called ACSP-REST) is a REST interface to IBM z/OS Cryptographic Services.

The programming interfaces in ACSP-REST include the following

- Representational State Transfer (REST) services for:
  - Cipher: Encrypt/Decrypt
  - Random Number Generate
  - Hash
- JSON and XML schemas for each REST service

## Who should use this document

This information is intended for:

- System programmers installing and configuring ACSP-REST in the z/OS environment. See Chapter 2 for details.
- Security administrators configuring ACSP-REST for z/OS users. See Chapter Error: Reference source not found for details.
- Application programmers writing programs that invoke the ACSP-REST APIs. See Chapter 5 for usage instructions and samples.

## Where to find more information

### IBM Advanced Crypto Service Provider (ACSP)

- ACSP2100 ACSP PKCS#11 API CCA/EP11 Installation and Programmers Guide
- ACSP2000 ACSP Programmers Guide
- ACSP4000 ACSP Installation and Configuration Guide
- ACSP4010 ACSP C Client Installation and Configuration Guide
- ACSP4020 ACSP Messages

### IBM Java API to CCA (jCCA)

See: http://www.ibm.com/security/cccc/tools/jcca.shtml

### IBM z/OS Integrated Cryptographic Service Facility (ICSF)

- SC14-7505-00 Cryptographic Services ICSF Overview

- SC14-7506-00 Cryptographic Services ICSF Administrator's Guide
- SC14-7507-00 Cryptographic Services ICSF System Programmer's Guide
- SC14-7508-00 Cryptographic Services ICSF Application Programmer's Guide

### IBM Cryptographic Co-processor

For a list of IBM Cryptographic Co-processor Publications see:

http://www.ibm.com/security/cryptocards/

## IBM ACSP-REST V1.0 License Terms

The IBM Advanced Crypto Service Provider REST API (ACSP-REST) is licensed to you under the terms of the International License Agreement for Non-Warranted Programs (ILAN) and its accompanying License Information and Notices files.

The ILAN and its accompanying License Information file are available at http://www-03.ibm.com/software/sla/sladb.nsf/displaylis/01488832ADB0C2AB85257B6E004BB017?OpenDocument

Copies of the license files can also be found in the Program's License Folder.

The ILAN file contains General Terms and Country-unique Terms.

The ACSP_REST_V1.0_License Information file contains Program name, version and release, Export and Import restrictions clause and when applicable a Separately Licensed Code clause.

# Summary of Changes

## Release 1.0

This is the first release of the IBM Advanced Crypto Service Provider REST API.

In this release, you can use the ACSP-REST API to perform cryptographic operations on a z/OS system. The REST-enabled cryptographic services in this release include:

- Cipher: Encrypt/Decrypt
- Random Number Generate
- Hash

To help you get started with using the ACSP-REST APIs, samples are included in this documentation and schema files are provided with the installation.

# 1. Overview

**z/OS Cloud Services** provides a set of APIs that applications can use to exploit z/OS capabilities. The APIs are easy-to-use HTTP services that are language- and platform-independent, stateless, scalable, and easily parsed.

The IBM Advanced Crypto Service Provider REST API is a cryptography service available for on- and off-platform use for z/OS Cloud Services (zCS). It provides access to z/OS cryptographic function using an HTTP Web service. This exposes ACSP services for use to distributed clients.

This service centralizes the definition, use, and maintenance of cryptographic keys, simplifying key management. It allows the use of secure and protected keys to avoid the exposure of sensitive key information outside of the trusted, hardened and tamper evident cryptographic co-processor. ACSP-REST also permits access control and logging on key service usage in addition to service charge-back to applications based on actual usage.

Any authorized RESTful application with the appropriate credentials may utilize the service. Such an application would submit an HTTP request over SSL/TLS to the URL of an ACSP-REST service along with a constructed “service request payload” used to configure the cryptographic operation. Once the request has been processed, the application returns a standard HTTP response carrying a “service response payload” containing the results of the requested operation. The service payloads are defined and formatted using standard JSON and XML media types and are based on the provided schemas.

Illustration 1 below is a diagram of a request/response interaction between a client and the encrypt service:

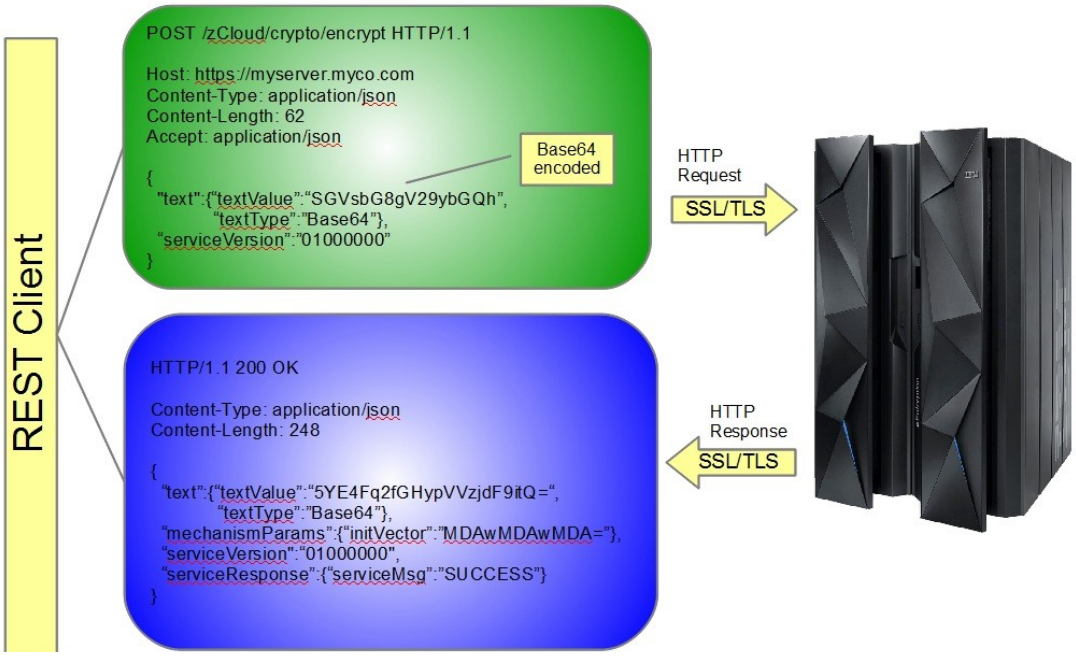


Illustration 1: A sample request/response interaction

# 2. Installation

## Program Requirements

The prerequisites for the z/OS ACSP-REST service are:

- IBM jCCA Package version 1.6.3 or later
- IBM z/OS V2R1 or later
- IBM z/OS Java Runtime Environment 64-bit (JRE) 7.0 or later.
- IBM z/OS Integrated Cryptographic Service Facility (ICSF) level HCR77A0 or later.
- IBM Websphere Application Server for z/OS Version 8.5.5.2 Liberty Profile with the Angel process
- IBM z/OSMF version 2.1 or later

## Installation Package

The IBM ACSP REST API package is delivered as an ACSP-RESTvrmm.pax file, where v is version, r is release, and mm is modification level, for example 1350.

Create the following directory structure in the Unix System Services file system.

```
mkdir -p /usr/lpp/acsp/rest
```

FTP in binary mode then unpax the ACSP-RESTvrmm.pax file into that directory.

```
cd /usr/lpp/acsp/rest
pax -rvf ACSP-RESTvrmm.pax .
```

The following files are included in the pax file:

| File name | Description |
|---|---|
| acsp-rest.changes.txt | Change log |
| acsp-rest.read.me | Installation information |
| acsp-rest.war | ACSP-REST modules and files |
| acsp-rest.properties | ACSP-REST default application parameters |
| schema/*.xsd<br>schema/*.json | JSON and XML Schemas |
| workflow/ZCS* | Sample z/OSMF workflow for Websphere Liberty Server provisioning |
| jcca/libjcca_zos_* | JCCA z/OS DLL's for specific ICSF releases |

## JCCA Installation

Inside the jcca folder are a set of files needed configure the JCCA component. These files are 31/64 bit z/OS Dynamic Load Libraries used to interface with the native hardware

drivers for the IBM Crypto Express cards on System z.

Create the following directory and copy the jcca files into it:

```
mkdir /usr/lpp/acsp/bin
cp /usr/lpp/acsp/rest/jcca/* /usr/lpp/acsp/bin
cd /usr/lpp/acsp/bin
```

Now select the proper .so files depending on the current release of ICSF that is running and ensure that the execute bits are turned on:

```
chmod -x libjcca_zos*
```

A link must also be created to the specific .so file - named libjcca16.so for 31 bit and libjcca16_64.so for 64 bit.

For example, to link to the HCR77A0 version, execute the following commands:

For 31 bit

```
ln -s libjcca_zos_HCR77A0_31.so.1.6.3.6 libjcca16.so
```

For 64 bit

```
ln -s libjcca_zos_HCR77A0_64.so.1.6.3.6 libjcca16_64.so
```

# 3. Security

z/OS Cloud Services uses SSL/TLS communication and requires client certificate authentication.

For each client (individual or organization) using the REST API, the security administrator must create a z/OS user ID and add or generate a client certificate that is associated with that z/OS user ID. The password-protected client certificate must be exported and installed in the client's browser or keystore for connection establishment.

The Liberty Server user ID also requires authorization to perform SAF checking and perform operations under the client user ID.

The following commands must be issued from a user with the RACF SPECIAL attribute or other authorities as listed below:

CLAUTH authority to the USER class

GROUP-SPECIAL authority in the zCS administrator group (e.g. ZCSADMIN)

UPDATE authority to the following resources in the FACILITY class:

- IRR.DIGTRING.ADDRING

CONTROL authority to the following resources in the FACILITY class:

- IRR.DIGTCERT.ADD
- IRR.DIGTCERT.GENCERT
- IRR.DIGTCERT.EXPORTKEY
- IRR.DIGTRING.CONNECT

The Liberty server user ID must have READ authority to the following resources in the CSFSERV class:

| Resource | Service | Service Description |
|---|---|---|
| CSFRNGL | CSNBRNGL | CCA Random Number Generation |

**Note:** The following security setup procedures are available as a workflow (more information in the section entitled “Automated Deployment (Using z/OSMF Workflow)”), but this section should still be reviewed by a security administrator.

**Note:** Everything in blue in the following sections is for demonstration purposes only and can be changed to fit your installation's individual needs.

## Server Certificates

Use the RACDCERT GENCERT command to create server certificates. The example below assumes the Liberty Server user ID is LIBSVR.

```
SETROPTS CLASSACT(DIGTCERT)

RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('zOS Cloud Services CA')
OU('Liberty') O('IBM')) WITHLABEL('zCloudLibertyCA') NOTAFTER(DATE(2035-12-31)
TIME(23:59:59)) RSA SIZE(2048)

RACDCERT ID(LIBSVR) GENCERT SUBJECTSDN(CN('myserver.myco.com')
OU('Liberty') O('IBM')) WITHLABEL('zCloudLibertyServer') SIGNWITH(CERTAUTH
LABEL('zCloudLibertyCA')) NOTAFTER(DATE(2035-11-30) TIME(23:59:59)) RSA
SIZE(2048)

SETROPTS RACLIST(DIGTCERT) REFRESH
```

Add the server certificates to a SAF key ring.

```
SETROPTS CLASSACT(DIGTRING)

RACDCERT ID(LIBSVR) ADDRING(zCloudKeyRing)

RACDCERT ID(LIBSVR) CONNECT(LABEL('zCloudLibertyServer') RING
(zCloudKeyRing) DEFAULT USAGE(PERSONAL))

RACDCERT ID(LIBSVR) CONNECT(CERTAUTH RING(zCloudKeyRing) LABEL
('zCloudLibertyCA') USAGE(CERTAUTH))

SETROPTS RACLIST(DIGTRING) REFRESH
```

Create a dataset for the server certificate and export it for installation into a client browser or application truststore.

```
RACDCERT CERTAUTH EXPORT (LABEL('zCloudLibertyServer')) DSN
('HLQ.LIBSRV.CERT') FORMAT(CERTDER)
```

## Client User IDs

Use the ADDUSER command to create z/OS user IDs. In effect, this will create a common identity shared by multiple application instances. In the context of this publication, we are using a functional user ID in which application identities are mapped back to this common user ID.

```
ADDUSER BLUETEAM OMVS(UID(0025) HOME(/)) NOPASSWORD OWNER(ZCSADMIN)
```

For easier administration, organize the z/OS user IDs into groups.

```
ADDGROUP ZCSACSP OMVS(GID(1125)) OWNER(ZCSADMIN) SUPGROUP(ZCSADMIN)

CONNECT BLUETEAM GROUP(ZCSACSP) AUTHORITY(USE)
```

Any user ID requiring access to the ACSP libraries (such as the WAS Liberty Server user ID, the ACSP started task ID, and any other ACSP users), should also be connected to the group.

```
CONNECT LIBSVR GROUP(ZCSACSP) AUTHORITY(USE)
```

## Client Certificates

Use the RACDCERT GENCERT command to create new certificates or the RACDCERT ADD command to import an existing certificate.

```
RACDCERT ID(BLUETEAM) GENCERT SUBJECTSDN(CN('Blue Team') OU('Location')
O('Company')) WITHLABEL('BLUETEAM.CERT') SIGNWITH(CERTAUTH
LABEL('zCloudLibertyCA'))  NOTAFTER(DATE(2035-11-30) TIME(23:59:59)) RSA SIZE(2048)

SETROPTS RACLIST(DIGTCERT) REFRESH
```

Create a dataset for the certificate and export it for installation into the client browser or keystore.

```
RACDCERT ID(BLUETEAM) EXPORT (LABEL('BLUETEAM.CERT')) DSN
('HLQ.BLUETEAM.CERT') FORMAT(PKCS12DER) PASSWORD('changeme')
```

## User Authorization

In order to use the REST API, all z/OS Cloud Services users must be authorized to the z/OS Cloud Services prefix as defined by the SAF Credential profile prefix in the server.xml file (see Chapter 4). **Note:** Ensure that the Liberty Server unauthenticated default user ID (i.e. WSGUEST) exists.

```
SETROPTS CLASSACT(EJBROLE) GENERIC(EJBROLE)

RDEFINE EJBROLE ZCLOUD.acsp-rest.user UACC(NONE) OWNER(ZCSADMIN)

PERMIT ZCLOUD.acsp-rest.user CLASS(EJBROLE) ACCESS(READ) ID(ZCSACSP)
PERMIT ZCLOUD.acsp-rest.user CLASS(EJBROLE) ACCESS(READ) ID(WSGUEST)

SETROPTS RACLIST(EJBROLE) REFRESH
```

**Note:** The high-level qualifier must be upper-case.

The z/OS Cloud Services users (and WSGUEST user ID) must also have READ access to the z/OS Cloud Services prefix in the APPL class.

```
RDEFINE APPL ZCLOUD UACC(NONE) OWNER(ZCSADMIN)

PERMIT ZCLOUD CLASS(APPL) ACCESS(READ) ID(ZCSACSP)
PERMIT ZCLOUD CLASS(APPL) ACCESS(READ) ID(WSGUEST)

SETROPTS RACLIST(APPL) REFRESH
```

## Liberty Authorization

The WAS Liberty server must be authorized to

- access the Liberty angel process services, which is required for the z/OS authorized services
- access the z/OS authorized services
- access the SAF authentication services

```
RDEFINE SERVER BBG.ANGEL UACC(NONE) OWNER(ZCSADMIN)

RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE) OWNER(ZCSADMIN)

RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC(NONE)
OWNER(ZCSADMIN)

PERMIT BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(LIBSVR)

PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(LIBSVR)

PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS(SERVER) ACCESS(READ)
ID(LIBSVR)

SETROPTS RACLIST(SERVER) REFRESH
```

The Liberty Server user ID must have READ access to the z/OS Cloud Services prefix.

```
RDEFINE SERVER BBG.SECPFX.ZCLOUD UACC(NONE) OWNER(ZCSADMIN)

PERMIT BBG.SECPFX.ZCLOUD CLASS(SERVER) ACCESS(READ) ID(LIBSVR)

SETROPTS RACLIST(SERVER) REFRESH
```

The Liberty Server user ID must be authorized to synchronize requests to the z/OS thread.

```
RDEFINE FACILITY BBG.SYNC.ZCLOUD UACC(NONE) OWNER(ZCSADMIN)

PERMIT BBG.SYNC.ZCLOUD CLASS(FACILITY) ACCESS(CONTROL) ID(LIBSVR)

SETROPTS RACLIST(FACILITY) REFRESH
```

The Liberty Server user ID must have READ access to list SAF keyrings.

```
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ACCESS(READ) ID(LIBSVR)
SETROPTS RACLIST(FACILITY) REFRESH
```

## ACSP Library Authorization

**Note:** In order to perform the following steps, the user must have UID(0) or one of the BPX or UNIXPRIV privileges.

The ACSP code must run in a clean program-controlled environment. Program control is managed by the extattr command. To set the ACSP code to be program-controlled, perform the following steps:

Go to the ACSP bin directory.

```
cd /usr/lpp/acsp/bin
```

List the attributes for the libjcca*.so and libjcca*.so symbolic links and note the actual files being referenced. Make sure you find the actual files referenced by the symbolic links. **You must set the attributes on the actual files. Setting the extended attributes on the symbolic links will be ineffective.**

```
ls -E libjcca16.so
ls -E libjcca16_64.so
```

Update the extended attributes of the **actual** files to allow program control.

```
extattr +p libjcca_zos_HCR77A0_31.so.1.6.3.6
extattr +p libjcca_zos_HCR77A0_64.so.1.6.3.6
```

Verify the settings by checking for the '-ps-'.

```
ls -E libjcca_zos_HCR77A0_31.so.1.6.3.6
ls -E libjcca_zos_HCR77A0_64.so.1.6.3.6
```

Give group authorization to the acsp libraries

```
chgrp ZCSACSP /usr/lpp/acsp/*
chmod -R 774 /usr/lpp/acsp/*
```

## ICSF Authorization

For authorization to the underlying ICSF services, see ”Required Authorizations” in Chapter 5.

## Server STARTED Profile

A STARTED class profile must be defined for the Liberty server to be started as a z/OS started task.

The example below assumes the name of the procedure to be used to start the server is ZCSZSRV (see Chapter 4), and the Liberty Server user ID is LIBSVR.

```
RDEFINE STARTED  ZCSZSRV.* UACC(NONE) STDATA(USER(LIBSRV)
PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))

SETROPTS RACLIST(STARTED) GENERIC(STARTED) REFRESH
```

**Note:** The server user ID must also have write access to files in the $WLP_USER_DIR directory (see Chapter 4).

# 4. Deployment

## Manual Deployment

A Liberty server instance can be deployed manually. Use the following chart for variable references in the steps below:

| Component | Variable Name | Description |
|---|---|---|
| **WAS Liberty Installation** | **WLP_INSTALL_DIR** | The WAS Liberty installation directory<br>(e.g. /usr/lpp/zWebSphere/V8R5/wlp) |
| | **WLP_USER_DIR** | The WAS Liberty user server directory<br>(e.g. /etc/liberty) |
| **ACSP-REST** | **ACSP_REST_INSTALL_DIR** | The ACSP-REST installation directory<br>(e.g. /usr/lpp/acsp/rest) |

**Note:** Everything in blue in the following sections is for demonstration purposes only and can be changed to fit your installation's individual needs.

1. Create a Liberty server instance

    $WLP_INSTALL_DIR/bin/server create cryptosrv

2. Configure the Liberty server

    cd $WLP_USER_DIR/servers/cryptosrv

3. Edit the server.xml in the $WLP_USER_DIR/servers/cryptosrv directory. Update the host (as shown below) and/or port numbers and verify that any field in blue below matches your security setup.

```
<server description="cryptosrv">

    <!-- HTTPS / Port -->
    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpsPort="9450" />

    <!-- Enable features -->
    <featureManager>
        <feature>ssl-1.0</feature>
        <feature>appSecurity-2.0</feature>
        <feature>zosSecurity-1.0</feature>
        <feature>jaxrs-1.1</feature>
    </featureManager>

    <!-- App Info -->
    <application type="war" id="acsp-rest" name="acsp-rest"
                 location="${server.config.dir}/apps/acsp-rest.war" />
    <applicationMonitor dropinsEnabled="false" />

    <!-- z/OS Security -->
    <safRegistry id="saf" />
    <safAuthorization id="saf" />
    <safCredentials profilePrefix="ZCLOUD"
                    unauthenticatedUser="WSGUEST" />
    <safRoleMapper profilePattern="ZCLOUD.%resource%.%role%"
                   toUpperCase="false" />
    <syncToOSThread appEnabled="true" />

   <!-- SSL Configuration -->
   <sslDefault sslRef="zCloudSSLConfig" />
     <ssl id="zCloudSSLConfig"
          keyStoreRef="zCloudKeyStore"
          trustStoreRef="zCloudTrustStore"
          clientAuthentication="true"
          sslProtocol="TLS"
          securityLevel="HIGH" />
     <keyStore id="zCloudTrustStore"
               location="safkeyring:///zCloudKeyRing"
               type="JCERACFKS"
               password="password"
               fileBased="false"
               readOnly="true" />
     <keyStore id="zCloudKeyStore"
               location="safkeyring:///zCloudKeyRing"
               type="JCERACFKS"
               password="password"
                fileBased="false"
                readOnly="true" />
</server>
```

4. Create a jvm.options file. Add the liberty server path along with the zCS and ACSP-REST environment variables (as shown below). Give relevant permissions to the created files.

oedit $WLP_USER_DIR/servers/cryptosrv/jvm.options

```
-Dacsp-rest=/usr/lpp/acsp/rest/acsp-rest.properties
```

```
chmod 750 $WLP_USER_DIR/servers/cryptosrv/jvm.options
```

5. Create a server.env file. Set JAVA_HOME to the z/OS Java Installation and LIBPATH to the installed ACSP binaries.

```
oedit $WLP_USER_DIR/servers/cryptosrv/server.env
```

```
JAVA_HOME=/usr/lpp/java/J7.1_64
LIBPATH=/usr/lpp/acsp/bin:$LIBPATH
```

```
chmod 750 $WLP_USER_DIR/servers/cryptosrv/server.env
```

6. Edit the acsp-rest.properties file from $ACSP_REST_INSTALL_DIR and update the default configurations. **Note:** The key label must exist in an ICSF Key Data Set and you must have authorization to the key label and services. See Chapter 5, Error: Reference source not found and Error: Reference source not found, for authorization requirements.

```
#ACSP-REST Properties File

#-* Cipher: Encrypt/Decrypt *-#

#Service Bias - PERFORMANCE = CPACF / SECURITY = CEX
cipherServiceBias = PERFORMANCE

#Cipher Key Label - Key Identifier
# cipherKeyLabel:  Secret/Symmetric or Public/Asymmetric key
label
# cipherKeyLabel2: Private/Asymmetric key label
cipherKeyLabel = AES.DATA.CLEAR

#Cipher Key Type - Cryptographic key algorithm
cipherKeyType = AES

#Cipher Mechanism - Processing Rule
cipherMechanism = CBC_PAD

#-* Hash *-#

#Hash Mechanism
hashMechanism = SHA256
```

7. Copy the acsp-rest.war from $ACSP_REST_INSTALL_DIR to the apps folder.

```
cp $ACSP_REST_INSTALL_DIR/acsp-rest.war
$WLP_USER_DIR/servers/cryptosrv/apps/.
```

8. Copy the $ACSP_REST_INSTALL_DIR/workflow/ZCSZSRV.stc sample Liberty start procedure to your PROCLIB dataset.

9. Ensure the user ID associated with the server task has write access to the files in the

$WLP_USER_DIR/servers directory.

10. Issue the following z/OS operator command to start the Liberty server instance:

```
START ZCSZSRV,PARMS='cryptosrv',INSTDIR='/usr/lpp/zWebSphere/V8R5/wlp',
USERDIR='/etc/liberty'
```

**Note:** If you need to stop the Liberty server instance, use the following command:

```
STOP ZCSZSRV
```

## Automated Deployment (Using z/OSMF Workflow)

The ACSP-RESTvrmm.pax file also includes a sample workflow to help simplify the deployment process. Once the .pax file is extracted, the workflow scripts can be found in the workflow folder.

The sample workflow comprises of the following steps:

1 Create Server and Started Task. This step comprises two substeps:

1.1 Create Server. This substep creates an instance of the IBM Websphere Application Server (WAS) Liberty Profile server by invoking the WAS Liberty CREATE SERVER shell command.

It accepts the following user variables:

- Server Name: name of the server instance to be created
- JAVA_HOME: the path to the Java Runtime Environment install
- WLP_USER_DIR: the path to the WAS Liberty Profile user area
- WLP_INSTALL_DIR: the path to the WAS Liberty Profile install

1.2 Install Server Started Task. This substep installs the JCL procedure for the Liberty server started task from the ACSP-REST installation directory to the PROCLIB dataset.

It accepts the following user variables:

- PROCLIB: name of the PROCLIB dataset in which the JCL procedure for the WAS Liberty server started task is to be installed
- Started Task Name: name of the started task to be assigned to the Liberty server
- ACSP_REST_INSTALL_DIR: the path to the ACSP-REST install
- Overwrite Option: a checkbox indicating whether the JCL procedure for the WAS Liberty server started task should be overwritten if one exists

1.3 Define Started Task Profile. This substep issues RACF commands to assign a user ID to the Liberty server. The script also creates or updates the RACF STARTED profile for the started task to be used to start the Liberty server.

It accepts the following user variables:

- PROCLIB: name of the PROCLIB dataset that contains the JCL procedure for the Liberty server started task

- Started Task Name: name of the started task assigned to Liberty server
- Server Name: name of the Liberty server instance
- Server ID: the RACF user ID associated with Liberty server

2 Modify Server. This step updates the server.xml file to define certain server attributes.

It accepts the following user variables:

- Server Name: name of the Liberty server instance to be modified
- WLP_USER_DIR: the path to the WAS Liberty Profile user area
- HTTP Port: the port used for client HTTP requests
- Host Name: host name to use for the server

3 Define Security Profiles. This step issues RACF commands to define security profiles for the client and server user IDs.

It comprises the following steps:

3.1 Define Client User ID. This substep issues RACF commands to define a user ID to RACF for the client that is using the z/OS Cloud Services.

It accepts the following user variables:

- Client ID: the RACF user ID associated with the client using the z/OS Cloud Services
- Client UID: the OMVS UID for the client user ID being defined to RACF
- Profile Owner: the RACF-defined user or group to be assigned as the owner of the RACF profile being defined

3.2 Define Group Profile. This substep issues RACF commands to define a group to RACF. This group is used to bundle the client and server user IDs together in order to simplify their management.

It accepts the following user variables:

- Group Name: name of the RACF-defined group to which the client and server IDs belong
- Group GID: the OMVS GID for the group to which the client and server IDs belong
- Superior Group: name of an existing RACF-defined group. This group becomes the superior group of the group profile being defined.

3.3 Add Client ID to Group. This substep issues RACF commands to connect the client user ID to the group.

It accepts the following user variables:

- Client ID: the RACF user ID associated with the client using the z/OS Cloud Services
- Group Name: name of the RACF-defined group the client is connecting to

3.4 Add Server ID to Group. This substep issues RACF commands to connect the server user ID to the group.

It accepts the following user variables:

- Server ID: the RACF ID associated with the Liberty server
- Group Name: name of the RACF-defined group the server is connecting to

4 Set up SSL. This step updates the server.xml file and issues the RACF commands needed for the server to accept an SSL connection.

Specifically, the script performs the following tasks:

- Issue RACF RACDCERT commands to create a ring, a Certificate Authority certificate, and a user certificate for the server instance, and to connect the certificates to the ring
- Issue RACF RACDCERT commands to export the Certificate Authority certificate to a dataset
- Modify the server.xml file to include z/OS Security and SSL features within the featureManager element
- Modify the server.xml file to set up the sslDefault element
- Modify the server.xml file to use the new SSL port number

It accepts the following user variables:

- Server ID: the RACF ID associated with the Liberty server
- Server Name: name of the Liberty server instance to be modified
- WLP_USER_DIR: the path to the WAS Liberty Profile user area
- SSL Port: SSL port number to use for the server

5 Create Client Certificate. This step issues RACF commands to create and export a new certificate for the client using the z/OS Cloud Services.

It accepts the following user variables:

- Client ID: the RACF user ID associated with the client
- Certificate DSN: the fully qualified name of the dataset to which the new certificate is to be exported

6 Authorize Server. This step sets up the authorizations required to permit the Liberty server access to the z/OS Cloud Services. It comprises the following substeps:

6.1 Liberty Server RACF Authorization. This substep issues RACF commands to authorize the Liberty server user ID to the z/OS Cloud Services prefix.

Specifically, the script performs the following tasks:

- Issue RACF commands to permit the Liberty server to use the Liberty angel process services
- Issue RACF commands to permit the Liberty server to use the z/OS authorized services
- Issue RACF commands to permit the Liberty server to check SAF credentials
- Issue RACF commands to permit the Liberty server to list SAF keyrings
- Issue RACF commands to grant the Liberty server READ access to the z/OS Cloud

Services security prefix

- Issue RACF commands to authorize the Liberty server to synchronize requests to the z/OS thread
- Modify the server.xml file to set up SAF elements
- Modify the server.xml file to enable syncToOSThread

It accepts the following user variables:

- Profile Owner: the RACF-defined user or group to be assigned as the owner of the RACF profile being defined
- z/OS Cloud Services Prefix: profile prefix to be assigned to the z/OS Cloud Services
- Server ID: the RACF ID associated with the Liberty server
- WLP_USER_DIR: the path to the WAS Liberty Profile user area
- Server Name: name of the Liberty server instance

6.2 ACSP Binary Authorization. This substep modifies the extended attributes of the ACSP libraries so that they can be controlled by Liberty server. The substep also changes the group associated with the ASCP libraries to the group the server user ID belongs to.

It accepts the following user variables:

- ACSP_INSTALL_DIR - the path to the ACSP install
- Group Name: name of the RACF-defined group to which the server user ID belong

6.3 Liberty Server Directory Authorization. This substep modifies the owner and permission bits of all files and folders of the Liberty server directory to ensure the Liberty server started task has access.

It accepts the following user variables:

- Server Name: name of the Liberty server instance to be modified
- WLP_USER_DIR: the path to the WAS Liberty Profile user area
- Server ID: the RACF ID associated with the Liberty server

7 Authorize Client. This step executes a REXX exec to issue RACF commands to authorize the client group to the z/OS Cloud Services profile prefix. It also executes a shell script to allow the client WRITE access to the jcca.log file in the server directory. It comprises the following substeps:

7.1 Authorize Client to Profile Prefix. This substep sets up the authorizations required to permit the client group access to the z/OS Cloud Services in order to use the REST APIs.

Specifically, the script performs the following tasks:

- Define a RACF EJBROLE class profile
- Permit the client group READ authority to the EJBROLE class profile
- Optionally permits the default Liberty Server unauthenticated user ID (WSGUEST) READ access to the EJBROLE class profile
- Define a RACF APPL class profile

- Permit the client group READ access to the APPL class profile
- Optionally permits the default Liberty Server unauthenticated user ID (WSGUEST) READ access to the APPL class profile

It accepts the following user variables:

- Profile Owner: the RACF-defined user or group to be assigned as the owner of the RACF profile being defined
- z/OS Cloud Services Prefix: profile prefix to be assigned to the z/OS Cloud Services
- Group Name: name of the RACF-defined group to which the client user IDs belong
- Authorize WSGUEST Option: a checkbox indicating whether to permit the default Liberty Server unauthenticated user ID (WSGUEST) access to both the EJBROLE and APPL class profiles.

13.1 Authorize Client to Log File. This substep modifies permission bits of the jcca.log file in the Liberty server directory to ensure the client user ID has WRITE access.

It accepts the following user variables:

- Server Name: name of the Liberty server instance to be modified
- WLP_USER_DIR: the path to the WAS Liberty Profile user area

14 Deploy and Configure Encryption and Decryption Services. This step deploys and configures the ACSP-REST Encryption and Decryption Services. It comprises the following substeps:

14.1 Deploy Encryption and Decryption Services. This substep deploys the ACSP-REST Encryption and Decryption Services. It performs the following tasks:

- Copy application WAR file to the deployment directory
- Create or replace the jvm.options file to define environmental variables
- Create or replace the server.env file
- Modify the server.xml file to include JSP and servlet features
- Modify the server.xml file to include application information

It accepts the following user variables:

- WLP_USER_DIR: the path to the WAS Liberty Profile user area
- Server Name: name of the Liberty server instance where the services are to be deployed
- ACSP_REST_INSTALL_DIR: the path to the ACSP-REST install
- ACSP_REST_OUTPUT_DIR: the path to the ACSP-REST output directory
- JAVA_HOME: the path to the Java Runtime Environment install

14.2 Configure Encryption and Decryption Services Settings. This substep sets up the default ACSP-REST Services settings in the acsp-rest.properties file. It performs the following task:

- Replace the following default key value pairs with user inputs:
  - serviceBias
  - cipherKeyLabel

- cipherKeyLabel2
- cipherKeyType
- cipherMechanism

It accepts the following user variables:

- Service Bias: PERFORMANCE (CPACF) or SECURITY (CEX)
- Cipher Key Label: secret/symmetric or public/asymmetric key label
- Cipher Key Label 2: private/asymmetric key label
- Cipher Key Type: cryptographic key algorithm
- Cipher Mechanism: processing rule

15 Start Server. This step starts the Liberty server instance as a z/OS started task by invoking the z/OS START command. In addition, a REXX exec is run to determine if the Liberty server instance started successfully by monitoring the standard output.

It accepts the following user variables:

- Started Task Name: name of the started task assigned to the Liberty server
- Server Name: name of the Liberty server instance to be started
- WLP_USER_DIR: the path to the WAS Liberty Profile user area
- WLP_INSTALL_DIR: the path to the WAS Liberty Profile install
- Jobname: the jobname to be assigned to the Liberty server started task

16 Stop Server. This step stops the Liberty server instance by invoking the z/OS STOP command. In addition, a REXX exec is run to determine if the Liberty server instance stopped successfully by monitoring the standard output.

It accepts the following user variables:

- Server Name: name of the Liberty server instance to be stopped
- WLP_USER_DIR: the path to the WAS Liberty Profile user area
- Jobname - the jobname of the Liberty server started task to be stopped

**Note:** This sample is only intended to assist in the development of a working program. IBM does not warrant that the content of this sample will meet your requirements or that it is error-free.

Illustrations 2-4 below show sample screenshots of using this workflow:

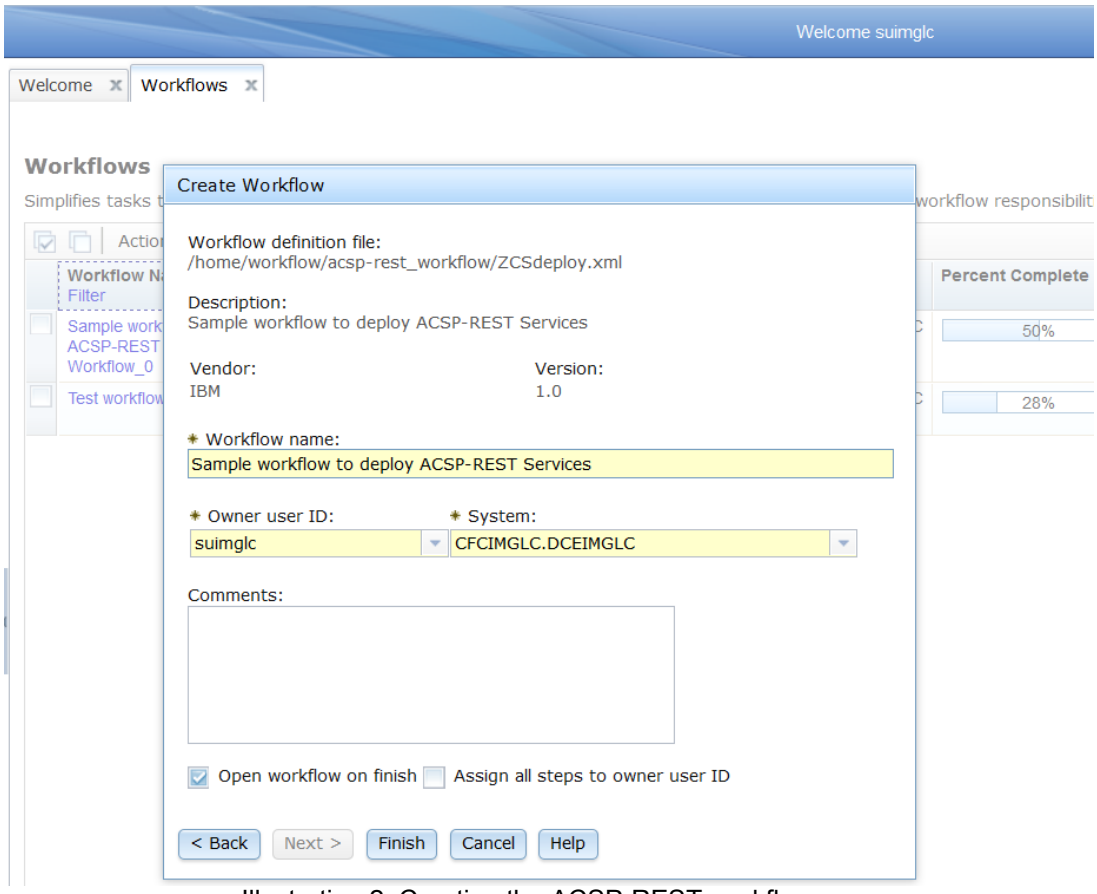


Illustration 2: Creating the ACSP-REST workflow

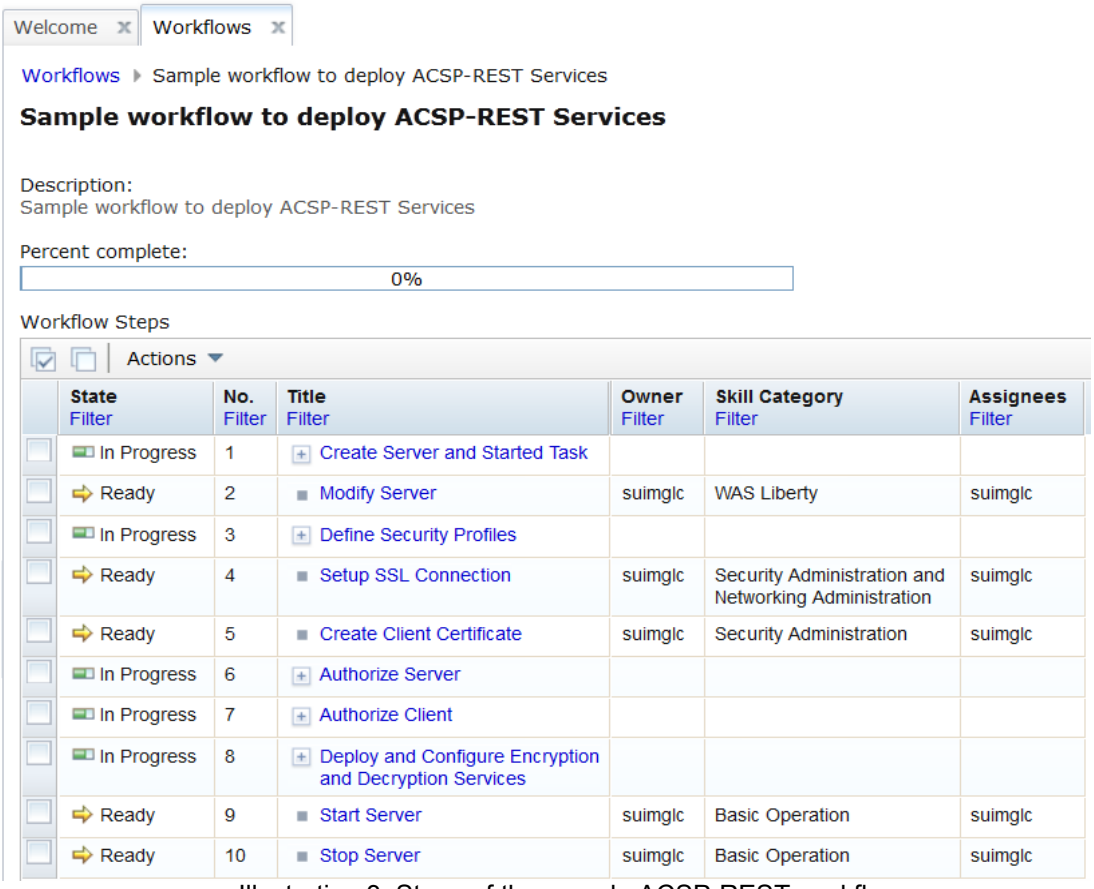


Illustration 3: Steps of the sample ACSP-REST workflow

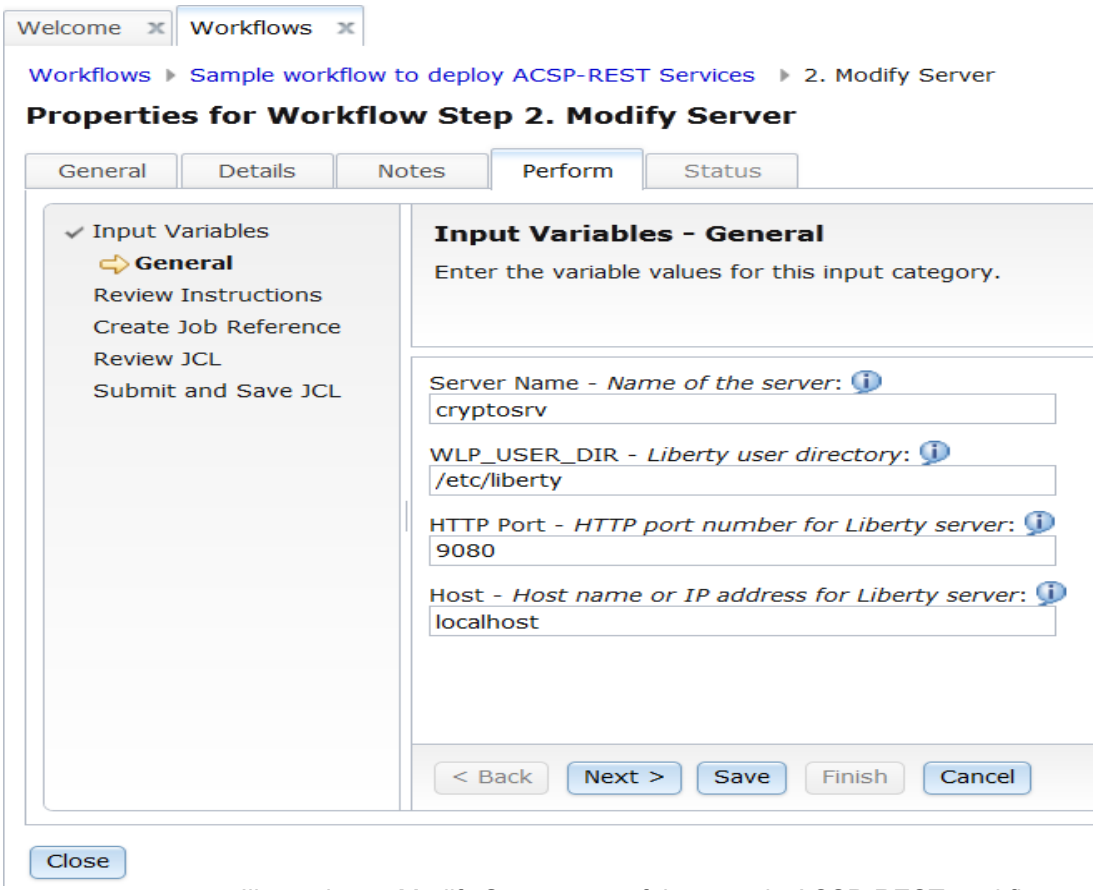


Illustration 4: Modify Server step of the sample ACSP-REST workflow

# Troubleshooting

Common installation problems are listed below.

**Note:** Be sure to restart the Liberty server after applying the following solutions.

| Problem | Solution(s) |
|---|---|
| ”Context Root Not Found” | • Verify that the acsp-rest.war is in the /apps/ folder. |
| No output / Blank results | • Verify the acsp-rest.properties file exists.<br>• Verify the LIBPATH in the server.env file is correct.<br>• Look at messages.log in the server logs for more information. |

# 5. Using the ACSP-REST Interface

The format for the **ACSP-REST**APIs requests and responses is based on the HTTP/1.1 protocol. Conceptually, your application (the client) issues requests to the target system (z/OS) in the form of request messages. Each request message consists of a request line, optionally followed by request headers (HTTP headers), an empty line, and an optional message body. The request line includes the HTTP method, such as GET, a Universal Resource Identifier (URI), and parameters that further qualify the request, where appropriate. For requests that return data, the services define an expected response in the form of a JavaScript Object Notation (JSON) object or as Extensible Markup Language (XML) elements.

It is assumed that users of these services are familiar with JSON (or XML) standard and coding practices. The following references provide additional helpful information:

- Hypertext Transfer Protocol 1.1: http://www.w3.org/Protocols/rfc2616/rfc2616.html
- Multipurpose Internet Mail Extensions (MIME) media types: http://www.iana.org/assignments/media-types/index.html
- Introducing JSON: http://www.json.org
- XML Specification: http://www.w3.org/XML

The **ACSP-REST** services are used to exploit z/OS cryptographic functions.

This release provides support for the following cryptographic functions:

- Cipher: Encrypt/Decrypt
- Hash
- Random Number Generate

**Default Values**

The default values may be set in the acsp-rest.properties file. For more information, see Chapter 4: Deployment.

## Processing Overview

The **ACSP-REST** can be invoked by any HTTPS client application, running on the z/OS local system or a remote system. Such an application will have to be authorized using the RACF/certificate security steps outlined earlier in this document.

The client program initiates an HTTPS request to the **ACSP-REST** REST API. If the interface determines that the request is valid, it performs the requested service. After performing the service, the **ACSP-REST** interface creates an HTTP response. If the request is successful, this response takes the form of an HTTP 200 (OK) response and an object containing a JSON (or XML) object which must be parsed by your program. If the request is not successful, the response includes a JSON response object indicating service failure.

## URI Format

The URIs of the **ACSP-REST** REST interface have the format

```
https://{host}:{port}/zCloud/crypto/{resource}
```

where:

- "https://{host}:{port}" specifies the target system address and port
- "/zCloud/crypto" identifies the **ACSP-REST** interface
- "{resource}" represents the service, e.g. Encrypt or Decrypt

## Content Types

**ACSP-REST** supports the JSON content type (”content-type: application/json”) and XML content type (”content-type: text/xml”). Either format may be used for the request and response data. For the detailed structure of each response object, see the individual JSON/XML schemas in the installation package (See Chapter 2).

Generally the JSON media type provides better overall performance as compared to XML and therefore is recommended instead of XML unless otherwise restricted. The performance differences tend to be minimal when dealing with small payload sizes but can become significant as the size of the payload grows.

## Error Handling

If an HTTP error occurs during the processing of a request, the **ACSP-REST** interface returns an appropriate HTTP status code to the calling client. An error is indicated by a 4*nn* code or a 5*nn* code. For example, HTTP/1.1 400 Bad Request or HTTP/1.1 500 Internal Server Error

Should an error occur in processing of the JSON payload, the **ACSP-REST** Interface will return a JSON object that contains a message that describes the error. You can use this information to diagnose the problem or provide it to IBM Support, if required.

The following HTTP status codes are valid:

**HTTP 200 OK**

Success.

**HTTP 400 Bad Request**

Request contained incorrect parameters.

**HTTP 401 Unauthorized**

Submitter of the request did not authenticate to **ACSP-REST** or is not authorized to use the service.

**HTTP 404 Bad URL**

Target of the request (a URL) was not found.

**HTTP 500 Internal server error**

Programming error.

## Error Logging

The logging for **ACSP-REST** is built into the existing WebSphere Liberty logging framework and all configurations are handled directly through the server.xml.

The basic configuration to enable tracing is achieved by adding the following line to the server definition inside your server.xml.

```
<logging traceSpecification="*=info"/>
```

This sets the trace granularity for the entire Liberty component to info. Additional fine grain control can be setup by appending additional components separated by a colon. To enable full diagnostic tracing for the ACSP-REST crypto services update the configuration to the following:

```
<logging traceSpecification="*=info:com.ibm.zcloud.*=all"/>
```

For more information please refer to the WebSphere Liberty logging documentation found here:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_logging.html?cp=SSAW57_8.5.5%2F3-17-0-0

## Error Diagnostics

A diagnostic object is always returned on failure. It is also returned when the verbose property is set to TRUE. (See individual JSON/XML Schemas in the installation package mentioned in Chapter 2.) The diagnostic object provides z/OS-specific information such as the underlying service name (e.g. CSNBSYE), the service return code, and the service reason code. All default and caller-specified properties are also returned in the diagnostic object.

```
{
   "cipherResponse": {
      "text": {
         "textType": "BASE64",
         "textValue": "NhqR6VVxBDcglUVwAA2nCA=="
      },
      "mechanismParams": {
         "initVector": "pQoi+gjcyNna9l17FcOaOQ=="
      }
      "serviceResponse": {
         "serviceMsgs": [
            "SUCCESS"
         ],
         "serviceDiag": {
            "operation": "ENCRYPT",
            "serviceRS": 0,
            "serviceRC": 0,
            "textType": "UTF8",
            "“textValue": "hello world",
            "chainOperation": "SINGLE",
            "mechanism": "CBC_PAD",
            "serviceBias": "PERFORMANCE",
            "keyType": "AES",
            "keyLabel": "AES.DATA.CLEAR",
            "serviceName": "CSNBSYE",
            "initVector": "pQoi+gjcyNna9l17FcOaOQ=="
         }
      }
   }
}
```

*Table 1: Sample Diagnostic Response (JSON)*

## Cipher: Encrypt /Decrypt

Use the cipher service to encrypt or decrypt data.

### Resource URI

https://{host}:{port}/zCloud/crypto/cipher

### HTTP Methods

The cipher service supports the following HTTP methods:

| | |
|---|---|
| **POST** | Submit a cipher(encrypt/decrypt) request |
| **PUT** | Not supported |
| **GET** | Not supported |

**DELETE** Not supported

## Required Authorizations

Cipher operations require authorization to ICSF services and key labels. For detailed information about ICSF authorization, please refer to the SC14-7506-00 Cryptographic Services ICSF Administrator's Guide.

To submit requests through the cipher service, the z/OS user ID associated with the client certificate must have READ authority to the following resources in the CSFSERV class:

| Resource | Service | Service Description |
|---|---|---|
| CSFENC | CSNBENC | CCA - Secure key encryption for DES |
| CSFSAE | CSNBSAE | CCA - Secure key encryption for AES |
| CSFPKE | CSNDPKE | CCA - PKA encryption for RSA |
| CSF1SKE | CSFPSKE | PKCS#11 - Secret key encryption for AES |
| CSF1PKV | CSFPPKV | PKCS#11 - PKA encryption for RSA |
| CSFDEC | CSNBDEC | CCA - Secure key decryption for DES |
| CSFSAD | CSNBSAD | CCA - Secure key decryption for AES |
| CSFPKD | CSNDPKD | CCA - PKA decryption for RSA |
| CSF1SKD | CSFPSKD | PKCS#11 - Secret key decryption for AES |
| CSF1PKS | CSFPPKS | PKCS#11 - PKA decryption for RSA |

**Note:** The CSNBSYE/CSNBSYD services processes requests for CCA - Clear and protected key encryption for AES and DES but does not require SAF authorization.

To use ICSF key labels, the z/OS user ID associated with the client certificate requires authorization to the specific key label.

- For CCA keys, READ access to the key label in the CSFKEYS class.
- For PKCS#11 keys, READ access to the token containing the key label in the CRYPTOZ class.

## JSON/XML Schemas

The JSON and XML schemas are provided in the ACSP-RESTvrmm.pax file.

- CipherRequestSchema_v1.0.0.json
- CipherRequestSchema_v1.0.0.xsd
- CipherResponseSchema_v1.0.0.json
- CipherResponseSchema_v1.0.0.xsd

## Sample Encrypt Request

An encrypt request can be minimal (uses defaults) or custom configured (some options are explicitly specified). The request must always include the operation and text to be

encrypted. The following tables show an example of a minimal and custom configured encrypt request.

```
{
   "cipherRequest": {
      "operation": "ENCRYPT",
      "text": {
         "textType": "UTF8",
         "textValue": "hello world"
      },
   }
}
```

*Table 2: Sample Minimal Encrypt Request (JSON)*

```
<cipherRequest>
    <operation>ENCRYPT</operation>
    <text>
        <textType>UTF8</textType>
        <textValue>hello world</textValue>
    </text>
</cipherRequest>
```

*Table 3: Sample Minimal Encrypt Request (XML)*

```
{
   "cipherRequest": {
      "operation": "ENCRYPT",
      "text": {
         "textType": "UTF8",
         "textValue": "hello world"
      },
      "chain": {
         "chainOperation": "SINGLE"
      },
      "mechanism": "CBC_PAD",
      "serviceVerbose": "false",
      "serviceBias": "PERFORMANCE",
      "key": {
         "keyType": "AES",
         "keyLabel": "AES.DATA.CLEAR"
      },
      "mechanismParams": {
         "initVector": "pQoi+gjcyNna9l17FcOaOQ=="
      }
   }
}
```

*Table 4: Sample Custom Configured Encrypt Request (JSON)*

## Sample Encrypt Response

An encrypt response will contain the results of the requested encrypt operation. The contents may vary depending on the success or failure of the operation but will always include a service response object which holds any generated service information. The following table shows a sample encrypt response.

```
{
   "cipherResponse": {
      "text": {
         "textType": "BASE64",
         "textValue": "NhqR6VVxBDcglUVwAA2nCA=="
      },
      "serviceResponse": {
         "serviceMsgs": [
            "SUCCESS"
         ]
      },
      "mechanismParams": {
         "initVector": "pQoi+gjcyNna9l17FcOaOQ=="
      }
   }
}
```

*Table 5: Sample Encrypt Response (JSON)*

```
<cipherResponse>
    <text>
        <textType>BASE64</textType>
        <textValue>NhqR6VVxBDcglUVwAA2nCA==</textValue>
    </text>
    <serviceResponse>
        <serviceMsgs>
            <string>SUCCESS</string>
        </serviceMsgs>
    </serviceResponse>
    <mechanismParams>
        <initVector>pQoi+gjcyNna9l17FcOaOQ==</initVector>
    </mechanismParams>
</cipherResponse>
```

*Table 6: Sample Encrypt Response (XML)*

**Note:** The initialization vector for a minimal request is generated randomly so the resulting cipher text will differ depending on the initialization vector used.

## Sample Decrypt Request

A decrypt request can be minimal (uses defaults) or custom configured (some options are

explicitly specified). The request must always include the operation, text to be decrypted, and the initialization vector used during encryption. The following tables show an example of a minimal and custom configured decrypt request.

```
{
   "cipherRequest": {
      "operation": "DECRYPT",
      "text": {
         "textType": "UTF8",
         "textValue": "NhqR6VVxBDcglUVwAA2nCA=="
      },
      "mechanismParams": {
         "initVector": "pQoi+gjcyNna9l17FcOaOQ=="
      }
   }
}
```

*Table 7: Sample Minimal Decrypt Request (JSON)*

```
<cipherRequest>
    <operation>DECRYPT</operation>
    <text>
        <textType>UTF8</textType>
        <textValue>NhqR6VVxBDcglUVwAA2nCA==</textValue>
    </text>
    <mechanismParams>
        <initVector>pQoi+gjcyNna9l17FcOaOQ==</initVector>
    </mechanismParams>
</cipherRequest>
```

*Table 8: Sample Minimal Decrypt Request (XML)*

```
{
   "cipherRequest": {
      "operation": "DECRYPT",
      "text": {
         "textType": "UTF8",
         "textValue": "NhqR6VVxBDcglUVwAA2nCA=="
      },
      "chain": {
         "chainOperation": "SINGLE"
      },
      "mechanism": "CBC_PAD",
      "serviceVerbose": "false",
      "serviceBias": "PERFORMANCE",
      "key": {
         "keyType": "AES",
         "keyLabel": "AES.DATA.CLEAR"
      },
      "mechanismParams": {
         "initVector": "pQoi+gjcyNna9l17FcOaOQ=="
      }
   }
}
```

*Table 9: Sample Custom Configured Decrypt Request (JSON)*

**Note:** The input textValue is always expected to be encoded as a Base64 string. The textType actually represents the desired encoding of the decrypted text returned in the response.

## Sample Decrypt Response

A decrypt response will contain the results of the requested decrypt operation. The contents may vary depending on the success or failure of the operation but will always include a service response object which holds any generated service information. The following table shows a sample decrypt response.

```
{
   "cipherResponse": {
      "text": {
         "textType": "UTF8",
         "textValue": "hello world"
      },
      "serviceResponse": {
         "serviceMsgs": [
            "SUCCESS"
         ]
      },
   }
}
```

*Table 10: Sample Decrypt Response (JSON)*

```
<cipherResponse>
    <text>
        <textType>UTF8</textType>
        <textValue>hello world</textValue>
    </text>
    <serviceResponse>
        <serviceMsgs>
            <string>SUCCESS</string>
        </serviceMsgs>
    </serviceResponse>
</cipherResponse>
```

*Table 11: Sample Decrypt Response (XML)*

# Random Number Generate (RNG)

Use the RNG service to generate a string of random bytes.

## Resource URI

https://{host}:{port}/zCloud/crypto/rng

## HTTP Methods

The RNG service supports the following HTTP methods:

| | |
|---|---|
| **POST** | Submit an RNG request |
| **PUT** | Not supported |
| **GET** | Not supported |
| **DELETE** | Not supported |

## Required Authorizations

RNG operations require authorization to ICSF services. For detailed information about ICSF authorization, please reference the SC14-7506-00 Cryptographic Services ICSF Administrator's Guide.

To submit requests through the RNG service, the z/OS user ID associated with the client certificate must have READ authority to the following resources in the CSFSERV class:

| Resource | Service | Service Description |
|---|---|---|
| CSFRNGL | CSNBRNGL | Random Number Generate |

## JSON/XML Schemas

The JSON and XML schemas are provided in the ACSP-RESTvrmm.pax file.

- RNGRequestSchema_v1.0.0.json
- RNGRequestSchema_v1.0.0.xsd
- RNGResponseSchema_v1.0.0.json
- RNGResponseSchema_v1.0.0.xsd

## Sample RNG Request

An RNG request must always be fully specified as defined by the RNGRequestSchema's. The following table shows an example of a fully specified RNG request.

```
{
   "RNGRequest": {
      "serviceVerbose": false,
      "RNGLength": 8
   }
}
```

*Table 12: Sample RNG request (JSON)*

```
<RNGRequest>
    <serviceVerbose>UTF8</serviceVerbose>
    <RNGLength>message</RNGLength>
</RNGRequest>
```

*Table 13: Sample RNG Request (XML)*

## Sample RNG Response

An RNG response will contain the results of the requested RNG operation. The contents may vary depending on the success or failure of the operation but will always include a service response object which holds any generated service information. The following table shows a sample RNG response.

```
{
   "RNGResponse": {
      "serviceResponse": {
         "serviceMsgs": [
            "SUCCESS"
         ]
      },
      "RNGValue": "HSl/dqH0UvU="
   }
}
```

*Table 14: Sample RNG Response (JSON)*

```
<RNGResponse>
    <serviceResponse>
        <serviceMsgs>
            <string>SUCCESS</string>
        </serviceMsgs>
    </serviceResponse>
    <RNGValue>HSl/dqH0UvU=</RNGValue>
</RNGResponse>
```

*Table 15: Sample RNG Response (XML)*

**Note:** Since the string of random bytes returned will always be Base64 encoded, the length of the string may actually be greater than the number of random bytes requested.

# Hash

Use the Hash service to generate a hash of the input text.

## Resource URI

https://{host}:{port}/zCloud/crypto/hash

## HTTP Methods

The RNG service supports the following HTTP methods:

| | |
|---|---|
| **POST** | Submit a hash request |
| **PUT** | Not supported |
| **GET** | Not supported |
| **DELETE** | Not supported |

## Required Authorizations

Hash operations require authorization to ICSF services. For detailed information about ICSF authorization, please reference the SC14-7506-00 Cryptographic Services ICSF Administrator's Guide.

To submit requests through the Hash service, the z/OS user ID associated with the client certificate must have READ authority to the following resources in the CSFSERV class:

| Resource | Service | Service Description |
|---|---|---|
| CSFOWH | CSNBOWH | One-Way Hash Generate |

## JSON/XML Schemas

The JSON and XML schemas are provided in the ACSP-RESTvrmm.pax file.

- HashRequestSchema_v1.0.0.json
- HashRequestSchema_v1.0.0.xsd
- HashResponseSchema_v1.0.0.json
- HashResponseSchema_v1.0.0.xsd

## Sample Hash Request

A hash request can be minimal (uses defaults) or custom configured (some options are explicitly specified). A request must always include the text to be hashed. The following tables show an example of a minimal and custom configured hash request.

```
{
   "hashRequest": {
      "text": {
         "textType": "UTF8",
         "textValue": "message"
      }
   }
}
```

*Table 16: Sample Minimal Hash Request (JSON)*

```
<hashRequest>
    <text>
        <textType>UTF8</textType>
        <textValue>message</textValue>
    </text>
</hashRequest>
```

*Table 17: Sample Minimal Hash Request (XML)*

```
{
   "hashRequest": {
      "serviceVerbose": true,
      "text": {
         "textType": "UTF8",
         "textValue": "message"
      }
      "mechanism": "SHA256"
   }
}
```

*Table 18: Sample Custom Configured Hash Request (JSON)*

## Sample Hash Response

A hash response will contain the results of the requested hash operation. The contents may vary depending on the success or failure of the operation but will always include a service response object which holds any generated service information. The following table shows a sample hash response.

```
{
   "hashResponse": {
      "serviceResponse": {
         "serviceMsgs": [
            "SUCCESS"
         ]
      },
      "hash":"q1MKE+RZFJgrefm34/uplM/R8/si9xzqGvvwK0YMbR0="
   }
}
```

*Table 19: Sample Hash Response (JSON)*

```
<hashResponse>
    <serviceResponse>
        <serviceMsgs>
            <string>SUCCESS</string>
        </serviceMsgs>
    </serviceResponse>
    <hash>q1MKE+RZFJgrefm34/uplM/R8/si9xzqGvvwK0YMbR0=</hash>
</hashResponse>
```

*Table 20: Sample Hash Response (XML)*

# 6. Appendix

## Glossary of Terms:

**REST (Representational State Transfer):** A software architecture style that is stateless, a client-server model, and cacheable.

**RESTful:** A program or service constructed with REST architecture.