THALES

CipherTrust Application Data Protection for Java - WebServices

USER GUIDE

Document Information

Product Version	8.13.1
Document Part Number	007-001770-002
Release Date	12 December 2022

Revision History

Revision	Date	Reason
A	12 December 2022	Initial release

Disclaimer

All information herein is either public information or is the property of and owned solely by Thales and/or its subsidiaries or affiliates who shall have and keep the sole right to file patent applications or any other kind of intellectual property protection in connection with such information.

Nothing herein shall be construed as implying or granting to you any rights, by license, grant or otherwise, under any intellectual and/or industrial property rights of or concerning any of Thales's information.

This document can be used for informational, non-commercial, and personal use only provided that:

> The copyright notice below, the confidentiality and proprietary legend and this full warning notice appear in all copies.

> This document shall not be posted on any publicly accessible network computer or broadcast in any media and no modification of any part of this document shall be made.

Use for any other purpose is expressly prohibited and may result in severe civil and criminal liabilities. The information contained in this document is provided "AS IS" without any warranty of any kind. Unless otherwise expressly agreed in writing, Thales makes no warranty as to the value or accuracy of information contained herein.

The document could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Furthermore, Thales reserves the right to make any change or improvement in the specifications data, information, and the like described herein, at any time.

Thales hereby disclaims all warranties and conditions with regard to the information contained herein, including all implied warranties of merchantability, fitness for a particular purpose, title and non-infringement. In no event shall Thales be liable, whether in contract, tort or otherwise, for any indirect, special or consequential damages or any damages whatsoever including but not limited to damages

resulting from loss of use, data, profits, revenues, or customers, arising out of or in connection with the use or performance of information contained in this document.
Thales does not and shall not warrant that this product will be resistant to all possible attacks and shall not incur, and disclaims, any liability in this respect. Even if each product is compliant with current security standards in force on the date of their design, security mechanisms' resistance necessarily evolves according to the state of the art in security and notably under the emergence of new attacks. Under no circumstances, shall Thales be held liable for any third party actions and in particular in case of any successful attack against systems or equipment incorporating Thales products. Thales disclaims any liability with respect to security for direct, indirect, incidental or consequential damages that result from any use of its products. It is further stressed that independent testing and verification by the person using the product is particularly encouraged, especially in any application in which defective, incorrect or functioning could result in damage to persons or property, denial of service or loss of privacy.

CONTENTS

PREFACE

This guide describes how to use the WebServices for CipherTrust Data Application Protection for Java (CADP for Java). The WebServices include SOAP and Rest APIs.

Release Notes

The Customer Release Notes (CRN) document provides important information about this release that is not included in other customer documentation. It is strongly recommended that you read the CRN to fully understand the capabilities, limitations, and known issues for this release.

Audience

This document is intended for personnel responsible for maintaining your organization's security infrastructure. This includes security officers, the key manager administrators, and network administrators. It is assumed that the users of this document are proficient with security concepts.

All products manufactured and distributed by Thales are designed to be installed, operated, and maintained by personnel who have the knowledge, training, and qualifications required to safely perform the tasks assigned to them. The information, processes, and procedures contained in this document are intended for use by trained and qualified personnel only.

Document Conventions

This section describes the conventions used in this document.

Command Syntax and Typeface Conventions

This document uses the following conventions for command syntax descriptions, and to highlight elements of the user interface.

Convention	Description
bold	The bold attribute is used to indicate the following: > Command-line commands and options (Type **dir /p**.) > Button names (Click **Save As**.) > Check box and radio button names (Select the **Print Duplex** check box.) > Window titles (On the **Protect Document** window, click **Yes**.) > Field names (**User Name:** Enter the name of the user.) > Menu names (On the **File** menu, click **Save**.) (Click **Menu** > **Go To** >

	Folders.) > User input (In the **Date** box, type **April 1**.)
italic	The italic attribute is used for emphasis or to indicate a related document. (See the *Installation Guide* for more information.)
Double quote marks	Double quote marks enclose references to other sections within the document.
<variable>	In command descriptions, angle brackets represent variables. You must substitute a value for command line arguments that are enclosed in angle brackets.
[optional] [<optional>] [a | b | c] [<a> | <b> | <c>]	Square brackets enclose optional keywords or <variables> in a command line description. Optionally enter the keyword or <variable> that is enclosed in square brackets, if it is necessary or desirable to complete the task. Square brackets enclose optional alternate keywords or variables in a command line description. Choose one command line argument enclosed within the braces, if desired. Choices are separated by vertical (OR) bars.
{ a | b | c } { <a> | <b> | <c> }	Braces enclose required alternate keywords or <variables> in a command line description. You must choose one command line argument enclosed within the braces. Choices are separated by vertical (OR) bars.

Notifications and Alerts

Notifications and alerts are used to highlight important information or alert you to the potential for data loss or personal injury.

Tips

Tips are used to highlight information that helps to complete a task more efficiently.

> **TIP:** This is some information that will allow you to complete your task more efficiently.

Notes

Notes are used to highlight important or helpful information.

> **NOTE:** Take note. Contains important or helpful information.

Cautions

Cautions are used to alert you to important information that may help prevent unexpected results or data loss.

> **CAUTION!** Exercise caution. Contains important information that may help prevent unexpected results or data loss.

Warnings

Warnings are used to alert you to the potential for catastrophic data loss or personal injury.

****WARNING**** **Be extremely careful and obey all safety and security measures. In this situation you might do something that could result in catastrophic data loss or personal injury.**

Related Document

- *CipherTrust Application Data Protection for Java User Guide*
- *CipherTrust Application Data Protection – Customer Release Notes*

Note: This document, may at times, abbreviate KeySecure Classic and CipherTrust Manager to Key Manager unless specified.

Support Contacts

If you encounter a problem while installing, registering, or operating this product, please refer to the documentation before contacting support. If you cannot resolve the issue, contact your supplier or Thales Customer Support.

Thales Customer Support operates 24 hours a day, 7 days a week. Your level of access to this service is governed by the support plan arrangements made between Thales and your organization. Please consult this support plan for further information about your entitlements, including the hours when telephone support is available to you.

Customer Support Portal

The Customer Support Portal, at https://supportportal.thalesgroup.com, is where you can find solutions for most common problems. The Customer Support Portal is a comprehensive, fully searchable database of support resources, including software and firmware downloads, release notes listing known problems and workarounds, a knowledge base, FAQs, product documentation, technical notes, and more. You can also use the portal to create and manage support cases.

> **NOTE:** You require an account to access the Customer Support Portal. To create a new account, go to the portal and click the **REGISTER** link.

Telephone Support

The support portal also lists telephone numbers for voice contact (Contact Us).

Email Support

You can also contact technical support by email at technical.support.DIS@thalesgroup.com.

CHAPTER 1: Overview

CipherTrust Application Data Protection for Java (CADP for Java) WebServices provide an interface to the capabilities of the Key Manager's key management and cryptographic operations platform. A very thorough set of operations are provided which include: key creation, pseudo random number generation, import and export of key and certificate, encrypt, decrypt, hashing, signing and hash/sign verification. Access to keys and cryptographic operations are centrally controlled and audited via the Key Manager GUI.

CADP for Java WebServices are deployed on Apache Tomcat 9 utilizing the CADP for Java provider, which enables interaction between the WebService and Key Manager. Both session oriented and stateless interfaces are provided to all capabilities via SOAP and REST protocols. When using the session oriented APIs, the session context must be established and maintained by the HTTP session during access to the appliances, and closed upon termination of the session. The stateless API requires credentials with every invocation and there is no common context between each call. It is also possible to combine access to both session oriented and stateless API wherever suitable as calls to the stateless API do not affect the active sessions.

Authentication to the WebService is controlled by Key Manager credentials stored within the appliances and optional Active Directory (AD) domain entities. This allows the accommodation of WebService specific credentials in combination with access by AD verified authorizations as is most suitable for each application. All WebService's users are represented in the Key Manager as cryptographic users and the WebService access is controlled and audited by the user authorizations defined within Key Manager.

CHAPTER 2: Installing CADP for Java WebServices

CADP for Java WebService is built to run on an existing Apache Tomcat installation. Since you may want to install Tomcat just to test the product, this chapter includes the basic installation procedures and some troubleshooting tips to get the web server running. These instructions should not be used to create a secure, professional, efficient or production-ready web server. These instructions are here to give you a quick way to test our Thales product.

These instructions were tested using Apache Tomcat 9.0.4. The results may vary.

CADP for Java WebService supports Apache Tomcat versions 7, 8, and 9.

Prerequisites

> JRE must be installed and JRE_HOME must be set. Alternatively, you can use JDK and set JAVA_HOME. Oracle Java 8 (minimum 1.8.0_111), 10, 11 (including OpenJDK and Amazon Corretto), 12 (OpenJDK and Azul Java), 14 (OpenJDK), 15 (OpenJDK), 17 (OpenJDK) and IBM Java 8 (minimum 8.0.6.25) are supported.

> Tomcat must be installed, for details refer to http://tomcat.apache.org/

Installing and Configuring CADP for Java

For instructions on how to install and configure CADP for Java, refer to the CipherTrust Application Data Protection for Java User Guide.

Installing CADP for Java WebService

Apache CXF gives flexibility to deploy both REST and SOAP based services on single web app (WAR). In this way we have different end points for both services, for example:

> **REST**: http://localhost:8080/protectappws/services/rest?_wadl

> **SOAP**: http://localhost:8080/protectappws/services/ProtectappwsImpl?wsdl

Installation

To install REST API:

- Copy `protectappws.war`, from the `CADP_for_JAVA\lib\webservices` directory to the `%CATALINA_HOME%\webapps` directory. If Tomcat is running, it may automatically extract the contents to the `%CATALINA_HOME%\webapps\protectappws`.

- Restart the Apache-Tomcat server.

NOTE: While deploying the protectappws war in tomcat, if folder with same name already exists in the `webapps` folder then it should be deleted for clean deployment before starting Tomcat.

To view the list of WebService samples, visit the Github page.

CHAPTER 3: CADP for Java WebService API

The CADP for Java WebService provides both: session oriented and stateless (SOAP and REST) interfaces to Key Manager cryptographic facilities. All session oriented access must be enclosed between Session_Open and Session_Close calls, while all stateless calls are autonomous and do not require any setup or closing context. For repeated operations it is most efficient to utilize the session based API as it has the capability to limit repeated interactions with the Key Manager by caching information and cryptographic objects for reuse within the same session.

Session affinity is managed through maintenance of an “HTTP” session scope. This allows the potential for both SOAP and REST interfaces to use session oriented access should that be desirable. The CADP for Java WebService is configured to use “transport session” scope which establishes an HTTP session cookie that the client must return with all interaction with the web server in order to enable access to the same session.

The APIs provided in CADP for Java WebService can be run over both http and https protocols. The following section describes how to configure the HTTPS support.

Configuring HTTPS Support for Tomcat

To support HTTPS, perform the following steps:

1. Create KeyStore attested with Certificate Authority (CA) or self-signed.

   For self-signed KeyStore using Java, use the following command:

   ```
   keytool -genkey -alias tomcat -keyalg RSA -keystore <keystore_name>.keystore
   ```

2. Edit the `server.xml` file under the `/CATALINA_HOME/conf` directory. Add/uncomment the following changes in the connector port.

   ```
   <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
   maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
   keystoreFile="<keystore_path>" keystorePass="changeit" clientAuth="false"
   sslProtocol="TLS" />
   ```

   Where, `keystoreFile` is the path of keystore file generated in step 1 and `keystorePass` is the password used during the keystore generation.

   NOTE: The changes in the `server.xml` file depends on the Tomcat server. The changes suggested in this step are for Tomcat 8 and 9.

3. Restart the Tomcat server.
4. Check through browser for HTTPS: https://< host-name>:<Port>/

WebService Interface Detail – Session Oriented API

All session oriented API calls have the prefix “Session_” which clearly illustrates that they utilize and require an active HTTP session context. All session operations must be brackets by Session_Open and Session_Close, with all other session methods being available while the session is open.

> **Note**: CADP for Java also allows you to create a NAE session using domain user. For more details, refer to Session Management Tasks on Thalesdocs.

Session_AddUsersToGroup: Add Users to a Group

URL: ./Session_AddUsersToGroup

Input Parameters:

- groupName – name of the group to which users are to be added.
- UserList – list of users to be added to the group. Users may be one or many.

Sample SOAP Parameters:

```
<prot:Session_AddUsersToGroup>
   <groupName>policy</groupName>
   <UserList>
      <!--1 or more repetitions:-->
      <user>cryptouser</user>
   </UserList>
</prot:Session_AddUsersToGroup>
```

Output: User(s) added successfully

<ns2:Session_AddUsersToGroupResponse xmlns:ns2="http://dsws.org/protectappws/">User(s) added successfully</ns2:Session_AddUsersToGroupResponse>

Session_AllUserInfo: Get Information of All the Users

URL: ./Session_AllUserInfo

Input Parameters:

none

Sample SOAP Parameters:

```
<prot:Session_AllUserInfo/>
```

Output: Information of all the users

<ns2:Session_AllUserInfoResponse xmlns:ns2="http://dsws.org/protectappws/"><![CDATA[<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<AllUserInfoResponse>

<UserDataList>

<UserData>

<User>testing</User>

<ModifyUserInfo>true</ModifyUserInfo>

</UserData>

<UserData>

<User>cryptouser</User>

<ModifyUserInfo>false</ModifyUserInfo>

<GroupList>

<Group>policy</Group>

</GroupList>

<CustomAttributeList>

<CustomAttribute>

<Name>twg_attritbute_1</Name>

<Value>twg@encryption.com</Value>

</CustomAttribute>

</CustomAttributeList>

</UserData>

</AllUserInfoResponse>]]></ns2:Session_AllUserInfoResponse>

Session_Cert_Export: Export Certificate (and Optionally Private Key)

URL: ./Session_Cert_Export

Input Parameters:

- certname – name of the certificate to export.
- certformat – this option is required if private key is also to be exported along with the certificate. Valid values: 1 or 8, representing PEM-PKCS#1 or PEM-PKCS#8 format.

Sample SOAP Parameters:

```
<prot:Session_Cert_Export>
      <certname>pkcs1sample</certname>
      <!--Optional:-->
      <certformat></certformat>
</prot:Session_Cert_Export>
```

Output: Certificate and the optional private key in PKCS#1 or PKCS#8 format

<ns1:Session_Cert_ExportResponse xmlns:ns1="http://dsws.org/protectappws/">-----BEGIN CERTIFICATE-----MIIDvzCCAqegAwIBAgIDAJ5EMA0GCSqGSIb3DQEBCwUAMIGbMQswCQYDVQQGEwJV

[… sample truncated for brevity …]
OcqQnevrP4rbUC/5W6+gO0m5ZjMDKryAyW4RiNCboGKtVTVcz68J0+75RTvycjWKibEI-----END CERTIFICATE-----</ns1:Session_Cert_ExportResponse>

Session_Cert_Import: Import certificate (and optional Private Key)

URL: ./Session_Cert_Import

Input Parameters:

- certname – name of the certificate to import.
- certisdeletable – sets whether the certificate can be deleted via the API, default is false.
- certisexportable – sets whether the certificate can be exported via the API, default is false.
- certificate – certificate to be imported, in PKCS1, PKCS#8, or PKCS#12 format.
- certpassword – optional, if password provided certificate must be Hex encoded.

Sample SOAP Parameters:

```
<prot:Session_Cert_Import>
      <certname>pkcs1samplevtN</certname>
      <certisdeletable>true</certisdeletable>
      <certisexportable>true</certisexportable>
      <certificate>-----BEGIN CERTIFICATE-----
      MIIDvzCCAqegAwIBAgIDAJ5EMA0GCSqGSIb3DQEBCwUAMIGbMQswCQYDVQQGEwJV
      [… sample truncated for brevity … ]
      OcqQnevrP4rbUC/5W6+gO0m5ZjMDKryAyW4RiNCboGKtVTVcz68J0+75RTvycjWK
      ibEI
```

```
-----END CERTIFICATE-----
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAhLItSYS7WHe22H+VOyj5GlAkwcylRsCAl/kbLBUW5adSB5K3
[… sample truncated for brevity … ]
mzXC86k6UN7ya29wDFuWwLK+gnwD2THORrdk5U+1B0PywK4JdDOR
-----END RSA PRIVATE KEY-----
</certificate>
<!--Optional:-->
<certpassword></certpassword>
</prot:Session_Cert_Import>
```

Output: boolean – indicates import success

<ns1:Session_Cert_ImportResponse xmlns:ns1="http://dsws.org/protectappws/">true</ns1:Session_Cert_ImportResponse>

NOTE: In order to import a PKCS12 certificate using web services you must provide certpassword field in import request. In this case certificate data i.e. <certificate> tag must be sent in Hex Format.

Session_ChangePassword: Change Password Permission for User

URL: ./Session_ChangePassword

Input Parameters:

- userName
- password
- isPasswordChangeable – optional.

Sample SOAP Parameters:

```
<prot:Session_ChangePassword>
<userName>cryptouser</userName>
<password>asdf1234</password>
<!--Optional:-->
<isPasswordChangeable>true</isPasswordChangeable>
```

```
</prot:Session_ChangePassword>
```

Output: Password/permission changed successfully

<ns2:Session_ChangePasswordResponse xmlns:ns2="http://dsws.org/protectappws/">Password/permission changed successfully</ns2:Session_ChangePasswordResponse>

Session_Close: Close the WebService Session, Release Any Active Context

URL: ./Session_Close

Input Parameters:

none

Sample SOAP Parameters:

```
<prot:Session_Close>
</prot:Session_Close>
```

Output: boolean – if a session was found it is closed, if no session found returns false

<ns1:Session_CloseResponse xmlns:ns1="http://dsws.org/protectappws/">true</ns1:Session_CloseResponse>

Session_CreateGroup: Create a New Group

URL: ./Session_CreateGroup

Input Parameters:

> groupName – name of the group to be created.

Sample SOAP Parameters:

```
<prot:Session_CreateGroup>
      <groupName>cryptogroup</groupName>
</prot:Session_CreateGroup>
```

Output: group created successfully

<ns2:Session_CreateGroupResponse xmlns:ns2="http://dsws.org/protectappws/">group created successfully.</ns2:Session_CreateGroupResponse>

Session_CreateUser: Create a New User

URL: ./Session_CreateUser

Input Parameters:

- userName – name of the new user to be created.
- userPassword – password for the new user.
- isPasswordChangeable – optional, use this parameter to change the password. The default value is `false`
- CustomAttributeList – optional, use this parameter to add custom attributes to the user.

Sample SOAP Parameters:

```
<prot:Session_CreateUser>
      <userName>cryptouser</userName>
      <userPassword>asdf1234</userPassword>
      <!--Optional:-->
      <isPasswordChanageable>true</isPasswordChanageable>
      <!--Optional:-->
       <CustomAttributeList>
         <CustomAttribute>
           <Name>name1</Name>
           <Value>value1</Value>
         </CustomAttribute>
         <CustomAttribute>
           <Name>name2</Name>
           <Value>value2</Value>
        </CustomAttribute>
      </CustomAttributeList>
</prot:Session_CreateUser>
```

Output: user created successfully

<ns2:Session_CreateUserResponse xmlns:ns2="http://dsws.org/protectappws/">user created successfully.</ns2:Session_CreateUserResponse>

Session_Decrypt: Decrypt Data Using Key Specified by Name

URL: ./Session_Decrypt

Input Parameters:

- keyname – name of the key to be used for decryption.
- ciphertext – encrypted data represented in Hex.
- keyiv – optional/blank. If blank, use the key's default IV, else, specify in Hex characters. The number of Hex character must be specific to the transformation used.
- transformation – transformation to be used. For example: AES/ ECIES/CBC/PKCS5Padding. For more information on supported ECIES transformations, refer to the CipherTrust Application Data Protection for Java User Guide.
- outputformat – optional, displays output in following formats: HEX, STR, and BASE64. Default is STR.

Sample SOAP Parameters:

```
<prot:Session_Decrypt>
   <keyname>aes256vt</keyname>
   <ciphertext> 10009046C980ECAFC6A79765A7ABAE01C846C5</ciphertext>
   <!--Optional:-->
   <keyiv></keyiv>
   <transformation>AES/CBC/PKCS5Padding</transformation>
   <!--Optional:-->
   <outputformat>STR</outputformat>
</prot:Session_Decrypt>
```

Output: Plaintext data

<ns1:Session_DecryptResponse xmlns:ns1="http://dsws.org/protectappws/">0000111122223333</ns1:Session_DecryptResponse>

Session_DeleteGroup: Delete a Group

URL: ./Session_DeleteGroup

Input Parameters:

- groupName – name of the group to be deleted.

Sample SOAP Parameters:

```
<prot:Session_DeleteGroup>
      <groupName>cryptogrouptest1</groupName>
</prot:Session_DeleteGroup>
```

Output: group deleted successfully

<ns2:Session_DeleteGroupResponse xmlns:ns2="http://dsws.org/protectappws/">group deleted successfully.</ns2:Session_DeleteGroupResponse>

Session_DeleteUser: Delete a User

URL: ./Session_DeleteUser

Input Parameters:

- userName – name of the user to be deleted.

Sample SOAP Parameters:

```
<prot:Session_DeleteUser>
   <userName>cryptouser</userName>
</prot:Session_DeleteUser>
```

Output: user deleted successfully

<ns2:Session_DeleteUserResponse xmlns:ns2="http://dsws.org/protectappws/">user deleted successfully.</ns2:Session_DeleteUserResponse>

Session_Encrypt: Encrypt Data Using Key Specified by Name

URL: ./Session_Encrypt

Input Parameters:

- keyname – name of the key to be used for encryption.
- plaintext – ASCII text to be encrypted, or hex if binary encryption is desired.
- keyiv – optional/blank. If blank, use the key's default IV, else, specify in Hex characters. The number of Hex character must be specific to the transformation used.
- transformation – transformation to be used. For example: AES/ ECIES/CBC/PKCS5Padding. For more information on supported ECIES transformations, refer to the CipherTrust Application Data Protection for Java User Guide.

Sample SOAP Parameters:

```
<prot:Session_Encrypt>
   <keyname>aes256vt</keyname>
   <plaintext>0000111122223333</plaintext>
   <!--Optional:-->
   <keyiv></keyiv>
   <transformation>AES/CBC/PKCS5Padding</transformation>
</prot:Session_Encrypt>
```

Output: Encrypted data in Hex

<ns1:Session_EncryptResponse xmlns:ns1="http://dsws.org/protectappws/">70C49C826D60E8564EDD51E8BF276C0FFF79D25420251D740D14C3919EC6663D</ns1:Session_EncryptResponse>

NOTE: In almost all cases the keyiv and transformation should not be specified: It is a useful practice to utilize the Key Manager capability to store the IV for the application and AES/CBC/PKCS5Padding – the default - is the most recommended cipher block mode.

Session_FPEDecrypt: Decrypt Data Using Format Preserving Encryption

URL : ./ Session_ FPEDecrypt

Input Parameters:

- ciphertext- encrypted data
- keyname – name of the key to be used for decryption.
- keyiv – optional, Hex encoded 56 bytes if plaintext > 56 Numeric value.
- tweakAlgo – optional, default is none.
- tweakData – If tweak data algorithm is "None" or absent, the value must be HEX encoded string representing 64 bit long (hence, HEX encoding will consume 16 characters). Tweak data is mandatory if Tweak Algo is given, else it is optional.

Sample SOAP parameters

```
<prot:Session_FPEDecrypt>
      <keyname>ghostKey2</keyname>
      <!--Optional:-->
      <keyiv></keyiv>
      <!--Optional:-->
      <tweakAlgo>none</tweakAlgo>
      <tweakData>1a23456712345678</tweakData>
      <ciphertext>393336313538</ciphertext>
</prot:Session_FPEDecrypt>
```

Output: Plaintext data

<ns2:Session_FPEDecryptResponse xmlns:ns2="http://dsws.org/protectappws/">239494</ns2:Session_FPEDecryptResponse>

Session_FPEEncrypt: Encrypt Data Using Format Preserving Encryption

URL : ./ Session_FPEEncrypt

Input Parameters:

- plaintext – data to be encrypted.
- keyname – name of the key.
- keyiv – optional, Hex encoded 56 bytes if plaintext > 56 Numeric value.
- tweakAlgo – optional, default is none.
- tweakData – If tweak data algorithm is "None" or absent, the value must be HEX encoded string representing 64 bit long (hence, HEX encoding will consume 16 characters). Tweak data is mandatory if Tweak Algo is given, else it is optional.

Sample SOAP parameters

```
<prot:Session_FPEEncrypt>
      <keyname>ghostKey2</keyname>
      <!--Optional:-->
      <keyiv></keyiv>
      <!--Optional:-->
      <tweakAlgo>none</tweakAlgo>
      <tweakData>1a23456712345678</tweakData>
```

```
        <plaintext>239494</plaintext>
    </prot:Session_FPEEncrypt>
```

Output: Encrypted data in Hex.

<ns2:Session_FPEEncryptResponse xmlns:ns2="http://dsws.org/protectappws/">393336313538</ns2:Session_FPEEncryptResponse>

Session_FPEFormatDecryption: Decrypt Data Using Format Preserving Encryption While Preserving Format of Ciphertext

URL : ./Session_FPEFormatDecryption

Input Parameters:

- format – format of the ciphertext to be preserved. Valid values are: `LAST_FOUR, FIRST_SIX, FIRST_SIX_LAST_FOUR, FIRST_TWO_LAST_FOUR,` and `NONE`. Also, only the format which was used for encryption, can be used here for decryption.
- keyName – name of the key.
- transformation – override the standard padding.
- data – data to be decrypted.
- tweakData – optional, If tweak data algorithm is "None" or absent, the value must be HEX encoded string representing 64 bit long (hence, HEX encoding will consume 16 characters). Tweak data is mandatory if Tweak Algo is given else it is optional.
- tweakAlgo – optional, default is none.

Sample SOAP parameters

```
    <prot:Session_FPEFormatDecryption>
        <format>LAST_FOUR</format>
        <keyName>cryptokeytest</keyName>
        <transformation>FPE/AES/CARD10</transformation>
        <data>713-456</data>
        <!--Optional:-->
        <tweakData>hello</tweakData>
        <!--Optional:-->
        <tweakAlgo>SHA1</tweakAlgo>
    </prot:Session_FPEFormatDecryption>
```

Output: Plaintext data with last four characters preserved.

<ns2:Session_FPEFormatDecryptionResponse xmlns:ns2="http://dsws.org/protectappws/">453-456</ns2:Session_ FPEFormatDecryptionResponse>

Session_FPEFormatEncryption: Encrypt Data Using Format Preserving Encryption While Preserving Format of Plaintext

URL : ./Session_FPEFormatEncryption

Input Parameters:

- format – the format in which some part of input data is to be kept intact, that is, the selected part of the input data is not encrypted. Valid values are: `LAST_FOUR, FIRST_SIX, FIRST_SIX_LAST_FOUR, FIRST_TWO_LAST_FOUR,` and `NONE.`
- keyName – name of the key.
- transformation – override the standard padding.
- data – data to be encrypted.
- tweakData – optional, If tweak data algorithm is "None" or absent, the value must be HEX encoded string representing 64 bit long (hence, HEX encoding will consume 16 characters). Tweak data is mandatory if Tweak Algo is given, else it is optional.
- tweakAlgo – optional, default is none.

Sample SOAP parameters

```
<prot:Session_FPEFormatEncryption>
      <format>LAST_FOUR</format>
      <keyName>cryptokey</keyName>
      <transformation>FPE/AES/CARD10</transformation>
      <data>713-456</data>
      <!--Optional:-->
      <tweakData>hello</tweakData>
      <!--Optional:-->
      <tweakAlgo>SHA1</tweakAlgo>
</prot:Session_FPEFormatEncryption>
```

Output: Ciphertext data with last four characters preserved.

<ns2:Session_FPEFormatEncryptResponse xmlns:ns2="http://dsws.org/protectappws/">163-456</ns2:Session_ FPEFormatEncryptResponse >

Session_GCMDecrypt: Decrypt Data Using GCM Standards

URL: ./Session_GCMDecrypt

Input Parameters:

- keyname – name of the key to be used for decryption.
- ciphertext – encrypted data represented in Hex.
- iv – specify 1 to 16 bytes IV in Hex format. For example 697473617265616c6c79636f6f6c6976.
- aad – optional/blank, the data to be passed to the recipient in plain text, needs to be “authenticated”.
- authtaglength – the tag length to ensure the data is not accidentally altered, must be 32 to 128 bits and must be multiple of 8.

Sample SOAP Parameters:

```
<prot:Session_GCMDecrypt>
      <keyname></keyname>
      <authtaglength></authtaglength>
      <iv></iv>
      <!--Optional:-->
      <aad></aad>
      <ciphertext></ciphertext>
</prot:Session_GCMDecrypt>
```

Output: Plaintext data

Session_GCMEncrypt: Encrypt Data Using GCM Standards

URL: ./Session_GCMEncrypt

Input Parameters:

- keyname – name of the key to be used for encryption.
- plaintext – ASCII text to be encrypted, or hex if binary encryption is desired.
- iv – specify 1 to 16 bytes IV in Hex format. For example 697473617265616c6c79636f6f6c6976..
- aad – optional/blank, the data to be passed to the recipient in plain text, needs to be “authenticated”.
- authtaglength – the tag length to ensure the data is not accidentally altered, must be 32 to 128 bits and must be multiple of 8.

Sample SOAP Parameters:

```
<prot:Session_GCMEncrypt>
    <keyname>AES</keyname>
    <authtaglength></authtaglength>
    <iv></iv>
    <!--Optional:-->
    <aad></aad>
    <plaintext>1234567</plaintext>
</prot:Session_GCMEncrypt>
```

Output: Encrypted data in Hex

Session_GetAllGroupInfo: Get Information of All Groups

URL: ./Session_GetAllGroupInfo

Input Parameters:

none

Sample SOAP Parameters:

```
<prot:Session_GetAllGroupInfo/>
```

Output: Information about all the groups

<ns2:Session_GetAllGroupInfoResponse xmlns:ns2="http://dsws.org/protectappws/"><![CDATA[<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<AllGroupInfoResponse>

<GroupInfoList>

<GroupInfo>

<Group>cryptogrouptest</Group>

<UserList>

<User>cryptouser</User>

</UserList>

</GroupInfo>

<GroupInfo>

<Group>cryptogroup</Group>

</GroupInfo>

</GroupInfoList>

</AllGroupInfoResponse>]]></ns2:Session_GetAllGroupInfoResponse>

Session_GetGroupInfo: Get Information of All the Users in a Group

URL: ./Session_GetGroupInfo

Input Parameters:

> groupName – name of the group for which information is to be retrieved.

Sample SOAP Parameters:

```
<prot:Session_GetGroupInfo >
      <groupName>cryptogrouptest</groupName>
</prot:Session_GetGroupInfo >
```

Output: Details of the group

<ns2:Session_GetGroupInfoResponse xmlns:ns2="http://dsws.org/protectappws/"><![CDATA[<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<GroupInfoResponse>

<Group>cryptogrouptest</Group>

<UserList>

<User>cryptouser</User>

</UserList>

</GroupInfoResponse>]]></ns2:Session_GetGroupInfoResponse>

Session_GetKeyAttributes: Get Custom Attributes of a Key

URL: ./Session_GetKeyAttributes

Input Parameters:

> keyName – name of the key for which custom attributes are to be retrieved.

Sample SOAP Parameters:

```
<prot:Session_GetKeyAttributes>
      <keyName>123425</keyName>
</prot:Session_GetKeyAttributes>
```

Output:

<ns2:Session_GetKeyAttributesResponse xmlns:ns2="http://dsws.org/protectappws/">{attr1=123456, attrw=234567}</ns2:Session_GetKeyAttributesResponse>

NOTE: The list of attributes that we get in response is comma separated so any attribute (name and value) having special characters (particularly **,** and **{}** and **“”** and **=**) will be displayed as it is and will make the attribute difficult to read. So, it is recommended NOT to use special characters (particularly **,** and **{}** and **“”** and **=**) in the input.

Session_GetKeyNames: Get List of Keys Associated With a User

URL: ./Session_GetKeyNames

Input Parameters:

- username – name of the user for which keys are to be retrieved.
- password – password of the user for which keys are to be retrieved.

Sample SOAP Parameters:

```
<prot:Session_GetKeyNames>
</prot:Session_GetKeyNames>
```

Output:

<ns2:GetKeyNamesResponse xmlns:ns2="http://dsws.org/protectappws/">[vijans_aes_128_nv, vijans_aes_128_v,....< …sample truncated for brevity…>..., vijans_aes_128_persistent_v]</ns2:GetKeyNamesResponse>

Session_HMAC: Perform Keyed Hash

URL: ./Session_HMAC

Input Parameters:

- keyname – name of the key to be used for HMAC.
- messagetext – data on which HMAC is to be performed.

Sample SOAP Parameters:

```
<prot:Session_HMAC>
```

```
         <keyname>hmacsha1vt</keyname>
         <messagetext>This is a message suitable for HMAC
signing...</messagetext>
      </prot:Session_HMAC>
```

Output: HMAC in Hex

<ns1:Session_HMACResponse xmlns:ns1="http://dsws.org/protectappws/">CDBE56D386CDD6D0EBBBC481ACCA84CD60CE2919 </ns1:Session_HMACResponse>

Session_HMAC_Verify: Verify Keyed Hash

URL: ./Session_HMAC_Verify

Input Parameters:

- keyname – name of the key to be used for HMAC.
- messagetext – data on which HMAC is to be performed.
- mac – message authentication code for verification.

Sample SOAP Parameters:

```
      <prot:Session_HMAC_Verify>
         <keyname>hmacsha1vt</keyname>
         <messagetext>This is a message suitable for HMAC
signing...</messagetext>
         <mac>CDBE56D386CDD6D0EBBBC481ACCA84CD60CE2919</mac>
      </prot:Session_HMAC_Verify>
```

Output: boolean – result of HMAC verification

<ns1:Session_HMAC_VerifyResponse xmlns:ns1="http://dsws.org/protectappws/">true</ns1:Session_HMAC_VerifyResponse>

Session_Key_Export: Export Key from the Key Management Appliances

URL: ./Session_Key_Export

Input Parameters:

- keyname – name of the key to export.

> keyformat – optional, format of the RSA or EC key to export. Valid values: PEM-PKCS#1 (default), PEM-PKCS#8, and PEM-SEC1.

> keytype – optional, type of the RSA key to export. Valid values: Public or Private.

NOTE: When exporting private EC keys, the key format must be set to PEM-SEC1 or PEM-PKCS#8.

Sample SOAP Parameters:

Symmetric Key

```
<prot:Session_Key_Export>
      <keyname>aes256vt</keyname>
</prot:Session_Key_Export>
```

Output: Key bytes in Hex for AES Keys and HMAC keys, PEM encoded for RSA keys

<ns1:Session_Key_ExportResponse xmlns:ns1="http://dsws.org/protectappws/">68711F03ABEE8B460509D5D54E7C70D3A78ABE21572746D31C433A797093B2CC</ns1:Session_Key_ExportResponse>

Asymmetric Key (private key)

```
<prot:Session_Key_Export>
      <keyname>RSA_1024_Key_version</keyname>
      <!--Optional:-->
      <keyformat>PEM-PKCS#8</keyformat>
      <!--Optional:-->
      <keytype>private</keytype>
</prot:Session_Key_Export>
```

Output: Key bytes in Hex for AES Keys and HMAC keys, PEM encoded for RSA keys

<ns2:Session_Key_ExportResponse xmlns:ns2="http://dsws.org/protectappws/">

-----BEGIN PRIVATE KEY-----

MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAMouxpL4GPXv09+T

mqwo354Ck08uFJPfSFLdAf+8VjOnTTBSkZuVdj2HLO7DOLsgev5Z4goetQ+YY3Th

GM0veDSIoTZSkZhf341pHSp5hi4/a4uMu/uMfz/Gf5U2jselYduU+AyOG3Pyu1Uw

KhIJDfVrcENNeZwwVUjftXY9NJ+/AgMBAAECgYBu0R6UFZoQyuf1ZoDjle+jOsQl

JRuwRTTt/icHVeJlo3CCaO9YOJ67cKjnIWep4U6otPlZFKWnK55vxKUEad1W6/yG

8pSQS4vF2YNVTWvlRgoD6WATo7Z4XgLsF9IIs/Mv5wEKzBIGhTj2Yvdg6JdspVkl

STfKv6HiX+ezhyiyiQJBAPQbXbmWZi9lJAUd2sPmSEYpuAzPjXjIxC0qJSUCV3cX

L51c19mTPlaa/eN1sIw1Mg14eoXNOjXK2KtpPWxXJmUCQQDUCIImOtMW1793KOss

ITmBu3rrJUXZFl9m8Dy3C4pB+f979vUDO+nQqx2qhdgR5KnDJAee/kpr9zMYCiWR

SSlTAkEAp/KUH3XsxDsP9z09IH81xqA5cwbxzOJEptZSTA46Y0NejSQtlhKKV0e3

2mFEoJKQ51e25Iv4ur3fKL/1dK7ZSQJAM2jEtaQ5niY9ZBTlwG+F+/CLAwyMfPAx

vXGuJuuDYC/PRC1ChsA2FsQGc1v0YZJBnvDTcDdTsA3Fs6RyK2HDxQJAe38evSde

bdsRg1Ke5TK+/x3cTYt9hcQK4IDB+Oadad6bnIxnd41eJBCEavoLm1KnzJZ+paBs

dA/rLbXrmXVQ+w==

-----END PRIVATE KEY-----

</ns2:Session_Key_ExportResponse>

Session_Key_Gen: Generate Keys

URL: ./Session_Key_Gen

Input Parameters:

- keyname – name of the new key to be created.
- keyalgorithm – must be one of: RSA-2048, RSA-3072, RSA-4096, AES-128, AES-192, AES-256, HmacSHA1, HmacSHA256, HmacSHA384, HmacSHA512, EC-secp224k1-225, EC-secp224r1-224, EC-secp256k1-256, EC-secp384r1-384, EC-secp521r1-521, EC-prime256v1-256, EC-brainpoolP224r1-224, EC-brainpoolP224t1-224, EC-brainpoolP256r1-256, EC-brainpoolP256t1-256, EC-brainpoolP384r1-384, EC-brainpoolP384t1-384, EC-brainpoolP512r1-512, and EC-brainpoolP512t1-512.
- keyisdeletable – set if the key will be deletable via the API – boolean, default is false.
- keyisexportable – set if the new key will be exportable via the API – boolean, default is false.
- keyisversioned – set if the new key will be a versioned key – boolean, default is false.
- keytemplate – optional, identify a template to use as a basis for the key generation.
- keyistemplate – optional, set if generating a template rather than a key – boolean, default is false.

Sample SOAP Parameters:

```
<prot:Session_Key_Gen>
      <keyname>testkey</keyname>
      <!--Optional:-->
```

```
            <keytemplate></keytemplate>
            <keyalgorithm>AES-256</keyalgorithm>
            <keyisdeletable>true</keyisdeletable>
            <keyisexportable>true</keyisexportable>
            <keyisversioned>true</keyisversioned>
            <!--Optional:-->
            <keyistemplate>false</keyistemplate>
      </prot:Session_Key_Gen>
```

Output: boolean – indicates if key creation was successful

<ns1:Session_Key_GenResponse xmlns:ns1="http://dsws.org/protectappws/">true</ns1:Session_Key_GenResponse>

Session_Key_Import: Import Key into the Key Management Appliances

URL: ./Session_Key_Import

Input Parameters:

- keyname – name of the key to import.
- keyalgorithm – options are: AES, RSA, EC, and Hmac.
- keyisdeletable – whether the imported key will be deletable via the API – boolean, default is false.
- keyisexportable – whether the imported key will be exportable via the API – boolean, default is false.
- keybytes – the bytes for key to be imported.

Sample SOAP Parameters:

```
      <prot:Session_Key_Import>
         <keyname>aes256vtimported</keyname>
         <keyalgorithm>AES</keyalgorithm>
         <keyisdeletable>true</keyisdeletable>
         <keyisexportable>true</keyisexportable>

<keybytes>68711F03ABEE8B460509D5D54E7C70D3A78ABE21572746D31C433A797093B2CC
         </keybytes>
      </prot:Session_Key_Import>
```

Output: boolean – indicate if key import was successful

<ns1:Session_Key_ImportResponse xmlns:ns1="http://dsws.org/protectappws/">true</ns1:Session_Key_ImportResponse>

Session_ModifyCustomAttributes: Modify the Attributes of a key

URL: ./Session_ModifyCustomAttributes

Input Parameters:

> keyName – name of the key for which attributes are to be modified.

> Attribute – list of attribute names and values. The type parameter lets you optionally specify the type for attribute. The default attribute type is String. The supported attribute types are:

- Date/Time
- Byte String
- Integer
- Long Integer
- Big Integer
- String
- Interval
- Enumeration
- Boolean

Sample SOAP Parameters:

```
<prot:Session_ModifyCustomAttributes>
      <KeyName>testaeskey</KeyName>
      <!--1 or more repetitions:-->
             <Attribute>
                <name>k1</name>
                <value>123467</value>
                <!--Optional:-->
                 <type>Integer</type>
              </Attribute>
              <Attribute>
                <name>k2</name>
                <value>7468616c657367656d616c746f</value>
```

```
                    <!--Optional:-->
                     <type>Byte String</type>
                </Attribute>
</prot:Session_ModifyCustomAttributes>
```

Output:

<ns2:Session_ModifyCustomAttributesResponse xmlns:ns2="http://dsws.org/protectappws/">Attributes added successfully.</ns2:Session_ModifyCustomAttributesResponse>

NOTE: The list of attributes that we get in response is comma separated so any attribute (name and value) having special characters (particularly **,** and **{}** and **“”** and **=**) will be displayed as it is and will make the attribute difficult to read. So, it is recommended NOT to use special characters (particularly **,** and **{}** and **“”** and **=**) in the input.

Session_Open: Establishes the WebService session

URL: ./Session_Open

Input Parameters:

- username
- password
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample SOAP Parameters:

```
<prot:Session_Open>
      <username>cryptouser</username>
      <password>qwerty1234</password>
      <!--Optional:-->
      <certalias>cu</certalias>
      <!--Optional:-->
      <certpassword>q1234</certpassword>
</prot:Session_Open>
```

Output: boolean

<ns1:Session_OpenResponse xmlns:ns1="http://dsws.org/protectappws/">true</ns1:Session_OpenResponse>

Session_PRNG: Pseudo Random Number Generator

URL: ./Session_PRNG

Input Parameters:

> length – the number of random bytes to be generated.

Sample SOAP Parameters:

```
<prot:Session_PRNG>
   <length>4</length>
</prot:Session_PRNG>
```

Output: Hex of random bytes

<ns1:Session_PRNGResponse xmlns:ns1="http://dsws.org/protectappws/">725C241D</ns1:Session_PRNGResponse>

NOTE: The lengths of the returned random bytes are represented in Hex and the length of the Hex value is twice as long as the number of actual bytes.

Session_ RemoveUsersFromGroup: Remove Users from a Group

URL: ./Session_RemoveUsersFromGroup

Input Parameters:

> groupName – name of the group from which user is to be removed.

> UserList – list of users to be removed from the group. Users may be one or many.

Sample SOAP Parameters:

```
<prot:Session_RemoveUsersFromGroup>
   <groupName>cryptogroup</groupName>
   <UserList>
```

```
            <!--1 or more repetitions:-->
            <user>cryptouser</user>
         </UserList>
      </prot:Session_RemoveUsersFromGroup>
```

Output: User(s) removed successfully

<ns2: Session_RemoveUsersFromGroupResponse xmlns:ns2="http://dsws.org/protectappws/">User(s) removed successfully</ns2: Session_RemoveUsersFromGroupResponse>

Session_RSA_Sign: Sign Message Text Using RSA Private Key

URL: ./Session_RSA_Sign

Input Parameters:

- keyname – name of RSA key pair containing private key.
- messagetext – message to sign.
- transformation – provide from one of the following: RSA, SHA1withRSA, SHA256withRSA, SHA384withRSA, SHA512withRSA, SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, and SHA512withRSAPSSPadding. For complete list of supported transformations, refer to the *Supported Algorithm* section of the *CADP for Java User Guide*.
- saltlength – optional, length of salt to be used for sign operation.
- format – optional, supported signing CMS formats are: cms/detached/der/enveloped, cms/detached/der, cms/detached/smime/enveloped and cms/detached/smime.
- messageformat – optional, supported message formats: HEX and STR. Default is STR.

- The saltlength parameter is supported only with: SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, and SHA512withRSAPSSPadding transformations.
- The saltlength and format parameters cannot be used simultaneously.
- The messagetext must be in the same format as specified in the messageformat parameter.
- CMS formats are not supported with the following transformations: SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, SHA512withRSAPSSPadding, RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and

RSAPSSPaddingSHA512.

- For KeySecure 8.12.5 onward, following transformations are added to sign the data based on pre-calculated hash: RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512. These transformations support saltlength parameter.

Session_RSA_Sign Sample with saltlength:

```
<prot:Session_RSA_Sign>
      <keyname>certpkcs12</keyname>
      <messagetext>eqwewewqeqqeqeqwe</messagetext>
      <transformation>SHA1withRSAPSSPadding</transformation>
      <!--Optional:-->
      <saltlength>40</saltlength>
      <!--Optional:-->
      <messageformat>STR</messageformat>
</prot:Session_RSA_Sign>
```

Output: signature in Hex

<ns2:Session_RSA_SignResponse xmlns:ns2="http://dsws.org/protectappws/">3082087B06092A864886F70D010703A082086C3082086802010318201C0308201BC0201003081A330819B310B3009060355040613025553310B3009060355040813024341311530130603550407130C526564776F6F64204369747931143012060355040A130B536166656E657420496E6331143012060355040B130B456E67696E656572696E6731123010060355040314097361 6D706C655F63613128302606092A864886F70D010901161973616D706C655F636140736166656E65742D696E632E636F6D020300A6E9300D06092A864886F70D01010105000482010096B5A17F59CA76C8DB69E280F5A6EC599651385C36A35175A2ED5B8018A2C63EF6EE3FE93C614D6848EAD8AC6AF42F68A921199621A7CDCAB9B886385F476DD9F76DD57F9[… sample truncated for brevity …]E8BD38E8FD7E9057F49F82E5610C32A29A512977F4E191480C639E9333D57F50A042C5D6665423AC42BC12C587C27620D4838045D590309B95318A9395948B38ED1CE76753058E8A17</ns2:Session_RSA_SignResponse>

Session_RSA_Sign Sample with format:

```
<prot:Session_RSA_Sign>
      <keyname>certpkcs12</keyname>
      <messagetext>eqwewewqeqqeqeqwe</messagetext>
      <transformation>SHA1withRSA</transformation>
```

```
        <!--Optional:-->
        <format>cms/detached/der/enveloped</format>
        <!--Optional:-->
        <messageformat>STR</messageformat>
    </prot:Session_RSA_Sign>
```

Output: signature in Hex

<ns2:Session_RSA_SignResponse xmlns:ns2="http://dsws.org/protectappws/">3082087B06092A864886F70D010703A082086C3082086802010318201C0308201BC0201003081A874519B310B30090603550406130255533 10B30090603550408130243413115301306035504071 30C526564776F6F64204369747931143012060355040A130B536166656E657420496E6331143012060355040B130B456E67696E656572696E6731123010060355040314097361 6D706C655F63613128302606092A864886F70D010901161973616D706C655F636140736166656E65742D696E632E636F6D020300A6E9300D06092A864886F70D01010105000482010009 6B5A17F59CA76C8DB69E280F5A6EC599651385C36A35175A2ED5B8018A2C63EF6EE3FE93C614D6848EAD8AC6AF42F68A921199621A7CDCAB9B886385F476DD9F76DD57F9[… sample truncated for brevity …]E8BD38E8FD7E9057F49F82E5610C32A29A512977F4E191480C639E9333D57F50A042C5D6665423AC42BC12C587C27620D4838045D590309B95318A9395948B38ED1CE76753058E8A17</ns2:Session_RSA_SignResponse>

Session_Sign: Sign Message Text Using RSA or EC Private Key

URL: ./Session_Sign

Input Parameters:

> keyname – name of RSA or EC key pair containing private key.

> messagetext – message to sign. It must be in the same format as specified in the messageformat parameter.

> transformation – provide one of the RSA/EC signing transformations supported by CADP for Java.

format – optional, supported signing CMS formats are: cms/detached/der/enveloped, cms/detached/der, cms/detached/smime/enveloped and cms/detached/smime.

> messageformat – optional, supported message formats: HEX and STR. Default is STR.

- The transformations: SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, SHA512withRSAPSSPadding, ECDSA, SHA1withECDSA, SHA256withECDSA, SHA384withECDSA, SHA512withECDSA, RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512 do not support

the CMS formats.

- For KeySecure 8.12.5 onward, following transformations are added to sign the data based on pre-calculated hash: RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512.

Sample SOAP Parameters:

```
<prot:Session_Sign>
      <keyname>certpkcs12</keyname>
      <messagetext>eqwewewqeqqeqeqwe</messagetext>
      <transformation>SHA1withECDSA</transformation>
      <!--Optional:-->
      <messageformat>STR</messageformat>
</prot:Session_Sign>
```

Output: signature in Hex

<ns2:Session_SignResponse xmlns:ns2="http://dsws.org/protectappws/">3082087B06092A864886F70D010703A082086C3082086802010 0318201C0308201BC0201003081A330819B310B3009060355040613025553310B3009060355040813024 3131153013060355040713C526564776F6F64204369747931143012060355040A130B536166656E657420 496E6331143012060355040B130B456E67696E656572696E6731123010060355040314097361 6D706C655F6361312830260 6092A864886F70D010901161973616D706C655F636140736166656E65742D696E632E6 36F6D020300A6E9300D06092A864886F70D01010105000482010009 6B5A17F59CA76C8DB69E280F5A6E C599651385C36A35175A2ED5B8018A2C63EF6EE3FE93C614D6848EAD8AC6AF42F68A921199621A7C DCAB9B886385F476DD9F76DD57F9[… sample truncated for brevity …] E8BD38E8FD7E9057F49F82E5610C32A29A512977F4E191480C639E9333D57F50A042C5D6665423AC42 BC12C587C27620D4838045D590309B95318A9395948B38ED1CE76753058E8A17</ns2:Session_SignRes ponse>

Session_RSA_Verify: Verify the Signature of Message Text Using RSA Public Key

URL: ./Session_RSA_Verify

Input Parameters:

> keyname – name of RSA key pair containing public key.

messagetext – message for signing verification. .

- signature – signature of RSA signing for verification in Hex.
- transformation – provide from one of the following: RSA, SHA1withRSA, SHA256withRSA, SHA384withRSA, SHA512withRSA, SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, and SHA512withRSAPSSPadding. For the complete list of supported transformations, refer to the *Supported Algorithm* section in the *Signing and Verifying* Chapter of the *CADP for Java User Guide*.
- saltlength – optional, length of salt to be used for sign verification operation.
- format – optional, supported signing CMS formats are: cms/detached/der/enveloped, cms/detached/der, cms/detached/smime/enveloped and cms/detached/smime.
- messageformat – optional, supported message formats: HEX and STR. Default is STR.
- caname – optional, name of the CA used for verifying.

- The saltlength parameter is supported only with: SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, and SHA512withRSAPSSPadding transformations.
- The saltlength and format parameters cannot be used simultaneously.
- The messagetext must be in the same format as specified in the messageformat parameter.
- CMS formats are not supported with the following transformations: SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, SHA512withRSAPSSPadding, RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512.
- For KeySecure 8.12.5 onward, following transformations are added to verify the data based on pre-calculated hash: RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512. These transformations support saltlength parameter.

Session_RSA_Verify Sample with saltlength:

```
<prot:Session_RSA_Verify>
      <keyname>certpkcs12</keyname>
      <messagetext>eqwewewqeqqeqeqwe</messagetext>
```

```
        <signature>3082087B06092A864886F70D010703A082086C3082086802010031820 1
        C0308201BC0201003081A330819B310B3009060355040613025553310B30090603550
        40813024341311530130603550407130C526564776F6F642043697479311430120603
        55040A130B536166656E657420496E6331143012060355040B130B456E67696E65657
        2696E67311230100603550403140973616D706C655F6361312830260609 2A864886F7
        0D010901161973616D706C655F636140736166656E65742D696E632E636F6D020300A
        6E9300D06092A864886F70D0101010500048201 00096B5A17F59CA76C8DB69E280F5A
        6EC599651385C36A35175[… sample truncated for brevity … ]
        A512977F4E191480C639E9333D57F50A042C5D6665423AC42BC12C587C27620D48380
        45D590309B95318A9395948B38ED1CE76753058E8A17</signature>

        <transformation>SHA1withRSAPSSPadding</transformation>

        <!--Optional:-->

        <saltlength>40</saltlength>

        <!--Optional:-->

        <messageformat>STR</messageformat>

    </prot:Session_RSA_Verify>
```

Output: boolean – result of verification

<ns2:Session_RSA_VerifyResponse xmlns:ns2="http://dsws.org/protectappws/">true</ns2:Session_RSA_VerifyResponse>

Session_RSA_Verify Sample with format:

```
    <prot:Session_RSA_Verify>

        <keyname>certpkcs12</keyname>

        <messagetext>eqwewewqeqqeqeqwe</messagetext>

        <signature>3082087B06092A864886F70D010703A082086C3082086802010031820 1
        C0308201BC0201003081A330819B310B3009060355040613025553310B30090603550
        40813024341311530130603550407130C526564776F6F642043697479311430120603
        55040A130B536166656E657420496E6331143012060355040B130B456E67696E65657
        2696E67311230100603550403140973616D706C655F6361312830260609 2A864886F7
        0D010901161973616D706C655F636140736166656E65742D696E632E636F6D020300A
        6E9300D06092A864886F70D0101010500048201 00096B5A17F59CA76C8DB69E280F5A
        6EC599651385C36A35175[… sample truncated for brevity … ]
        A512977F4E191480C639E9333D57F50A042C5D6665423AC42BC12C587C27620D48380
        45D590309B95318A9395948B38ED1CE76753058E8A17</signature>

        <transformation>SHA1withRSA</transformation>

        <!--Optional:-->

        <format>cms/detached/der/enveloped</format>

        <!--Optional:-->
```

```
        <messageformat>STR</messageformat>
        <!--Optional:-->
        <caname>sample_ca</caname>
    </prot:Session_RSA_Verify>
```

Output: boolean – result of verification

<ns2:Session_RSA_VerifyResponse xmlns:ns2="http://dsws.org/protectappws/">true</ns2:Session_RSA_VerifyResponse>

Session_SignVerify: Verify the Signature of Message Text Using RSA or EC Public Key

URL: ./Session_SignVerify

Input Parameters:

- keyname – name of RSA or EC key pair containing public key.
- messagetext – message for signing verification. It must be in the same format as specified in the messageformat parameter.
- signature – signature of RSA or EC signing for verification in Hex.
- transformation – provide one of the RSA/EC signverify transformations supported by CADP for Java .
- format – optional, supported signing CMS formats are: cms/detached/der/enveloped, cms/detached/der, cms/detached/smime/enveloped and cms/detached/smime..
- messageformat – optional, supported message formats: HEX and STR. Default is STR.
- caname – optional, name of the CA used for verifying.

- The transformations: SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, SHA512withRSAPSSPadding, ECDSA, SHA1withECDSA, SHA256withECDSA, SHA384withECDSA, SHA512withECDSA, RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512 do not support the CMS formats.
- For KeySecure 8.12.5 onward, following transformations are added to sign/verify the data based on pre-calculated hash: RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512.

Sample SOAP Parameters:

```
<prot:Session_SignVerify>
      <keyname>certpkcs12</keyname>
      <messagetext>eqwewewqeqqeqeqwe</messagetext>
      <signature>3082087B06092A864886F70D010703A082086C3082086802010031820 1
      C0308201BC0201003081A330819B310B3009060355040613025553310B30090603550
      40813024341311530130603550407130C526564776F6F642043697479311430120603
      55040A130B536166656E657420496E6331143012060355040B130B456E67696E65657
      2696E67311230100603550403140973616D706C655F6361312830260609 2A864886F7
      0D010901161973616D706C655F636140736166656E65742D696E632E636F6D020300A
      6E9300D06092A864886F70D0101010500048201000 96B5A17F59CA76C8DB69E280F5A
      6EC599651385C36A35175[… sample truncated for brevity … ]
      A512977F4E191480C639E9333D57F50A042C5D6665423AC42BC12C587C27620D48380
      45D590309B95318A9395948B38ED1CE76753058E8A17</signature>
      <transformation> SHA1withECDSA</transformation>
      <!--Optional:-->
      <messageformat>STR</messageformat>
      <!--Optional:-->
      <caname>sample_ca</caname>
</prot:Session_SignVerify>
```

Output: boolean – result of verification

<ns2:Session_SignVerifyResponse xmlns:ns2="http://dsws.org/protectappws/">true</ns2:Session_SignVerifyResponse>

Session_UserInfo: Get Information of a User

URL: ./Session_UserInfo

Input Parameters:

> userName – name of the user for whom information is to be retrieved.

Sample SOAP Parameters:

```
<prot:Session_UserInfo>
      <userName>cryptouser</userName>
</prot:Session_UserInfo>
```

Output: Details of the user

<ns2:Session_UserInfoResponse xmlns:ns2="http://dsws.org/protectappws/"><![CDATA[<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<UserInfoResponse>

<User>cryptouser</User>

<GroupList>

<Group>cryptogroup</Group>

</GroupList>

<CustomAttributeList>

<CustomAttribute>

<Name>twg_attritbute_1</Name>

<Value>twg@encryption.com</Value>

</CustomAttribute>

<CustomAttribute>

<Name>twg_attritbute_2</Name>

<Value>twg2@encryption.com</Value>

</CustomAttribute>

</CustomAttributeList>

<isChangePasswdPermission>false</isChangePasswdPermission>

</UserInfoResponse>]]></ns2:Session_UserInfoResponse>

Session_WrapKey: Key Wrap

URL: ./Session_WrapKey

Input Parameters:

- keyName – name of the key to export.
- keyUseForWrap – key to be used for wrapping.
- wrapFormatPadding – (optional) padding format to be used for wrapping the key. It is used for PKCS#1v2.1 and one of the following padding is used: SHA256, SHA384, and SHA512.

Sample SOAP Parameters:

```
<prot:Session_WrapKey>
      <keyName>AESKey</keyName>
      <keyUseForWrap>RSAKey</keyUseForWrap>
      <!--Optional:-->
```

```
        < wrapFormatPadding>SHA256</wrapFormatPadding>
    </prot:Session_WrapKey>
```

Output: Wrap key bytes

<ns2: Session_WrapKeyResponse xmlns:ns2="http://dsws.org/protectappws/">36E409F7993906344FA0DC560475086F485163857ACD41752651ACDF236BDDE73F9859CBF42A744D27603F5869D3DBD29C97005B973517DB76761AF8915D0B13</ns2: Session_WrapKeyResponse>

NOTE: cxf-manifest.jar should be included in the Java client build path while calling Session_WrapKey web services.

WebService Interface Detail – Stateless API

The stateless API offers identical functionality to that provided in the session oriented API, with the exception that there is no context established or maintained between calls to the Stateless WebService. In practice this means that credentials must be provided for every call, and access to cryptographic objects cannot be cached for efficiency between them. REST sample URLs are provided below along with SOAP content samples.

Two parameters certAlias and certPassword are optional for all the requests. CertAlias is a client certificate alias for making SSL connections and certPassword is the password for the provided certificate alias.

ProtectApp WebService supports caching of session-pool for a user. In session-pool, a session is cached for a user so that for additional requests no new session is created for the particular user till the time the user-session is cached.

To maintain session for a user, session-pool is created using the following parameters. The parameters are maintained in `cxf-beans` file in the `webapps\protectappws\WEB-INF` directory. User can modify the default values, as required.

Size of cache: The number of different user sessions that can be cached. Default value is `2000`.

Change the default value `2000` in the following code snippet as required in the `cxf-beans` file,

```
<!-- size of cache -->
<constructor-arg type = "int" value = "2000"/>
```

Session expiry time: The time after which the session expires from the last use of the particular cached session. Default value is `3600000` millisecond.

Change the default value `3600000` in the following code snippet as required in the `cxf-beans` file,

```
<!-- session expiry time in milli sec -->
<constructor-arg type = "int" value = "3600000"/>
```

Salt: The string used along with the Message digest algorithm parameter to create the hash of the key to store the user session in the cache. Default string used is `ThisIsIt432@123`.

Change the default string `ThisIsIt432@123` in the following code snippet as required in the `cxf-beans` file,

```
<!-- salt -->
<value>ThisIsIt432@123</value>
```

Message digest algorithm: The algorithm used to calculate the hash of the key used in creating the user session. Default algorithm used is `SHA-256`. Possible values can be `SHA-1`, `SHA-256,` and `SHA-512`.

Change the default algorithm `SHA-256` in the following code snippet as required in the `cxf-beans` file,

```
<!-- message digest algorithm  -->
<value>SHA-256</value>
```

AddUsersToGroup: Add Users to a Group

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/groups/addUsers

Input Parameters:

- adminUser
- adminPassword
- groupName
- UserList
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
  "GroupUpdateRequest": {
    "adminUser": "cryptouser",
    "adminPassword": "asdf1234",
    "groupName": "policy",
    "UserList":
      {
      "user": [
        "abc12",
        "pqr4321",
        "crypto123"
```

```
            ]
      }
  }
}
```

response:

```
{
  "GroupUpdateResponse": {
    "description": "[abc12, pqr4321, crypto123] added in the group."
  }
}
```

AllUserInfo: Get Information of All Users

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/users/info

Input Parameters:

- adminUser
- adminPassword
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
"AllUserInfoRequest":{
   "adminUser": "testadmin",
   "adminPassword": "test1234",
   "certAlias": "privatekey",
   "certPassword": "test1234"
   }
}
```

Sample REST response

response:

```
{"AllUserInfoResponse": {"UserDataList": {"UserData": [
```

```
    {
    "User": "testuser",
    "ModifyUserInfo": true,
    "GroupList": {"Group": "testGroup"},
   "CustomAttributeList": {"CustomAttribute": [
        {
          "Name": "twg_attritbute_1",
          "Value": "twg@encryption.com"
       },
       {
          "Name": "twg_attritbute_2",
          "Value": "twg2@encryption.com"
       }
    ]}
},
    {
    "User": "dbusr",
    "ModifyUserInfo": true,
    "GroupList": {"Group":       [
       "SQLGroup",
       "testGroup"
    ]}
},
    {
    "User": "encrypt_decrypt_user",
    "ModifyUserInfo": true
},
  ]}}}
```

Cert_Export: Export Certificate (and Optionally Private Key)

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/certExport

Input Parameters:

- username
- password
- certname – name of the certificate to export.
- certformat – this option is required if private key is to be included along with the certificate. Valid values: 1 or 8, representing PEM-PKCS#1 or PEM-PKCS#8 format.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
  "Cert_Export": {
    "username": "cryptouser",
    "password": "safenet123",
    "certname": "safmplded343",
    "certformat": "1"
  }
}
```

response:

```
{
  "certExportResponse": "       -----BEGIN CERTIFICATE-----
        MIIDvzCCAqegAwIBAgIDAKLeMA0GCSqGSIb3DQEBCwUAMIGbMQswCQYDVQQGEwJV
        UzELMAkGA1UECBMCQ0ExFTATBgNVBAcTDFJlZHdvb2QgQ2l0eTEUMBIGA1UEChML
        U2FmZW5ldCBJbmMxFDASBgNVBAsTC0VuZ2luZWVyaW5nMRIwEAYDVQQDFAlzYW1w
        bGVfY2ExKDAmBgkqhkiG9w0BCQEWGXNhbXBsZV9jYUBzYWZlbmV0LWluYy5jb20w
        HhcNMTMwOTE4MDUyNDQ1WhcNMjMwOTE2MDUyNDQ1WjBnMQswCQYDVQQGEwJ1czEL
        MAkGA1UECBMCY2ExFTATBgNVBAcTDHJlZHdvb2QgY2l0eTEQMA4GA1UEChMHc2Fm
        ZW5ldDEUMBIGA1UECxMLZW5naW5lZXJpbmcxDDAKBgNVBAMTA2JhcjCCASIwDQYJ
        KoZIhvcNAQEBBQADggEPADCCAQoCggEBAOedPA5y5loz6nYkPUHr3LOJlohcppq+
        zrCbIIJy3PmgXseRZHHIVno7H1UttZjXkGY0W7fDrsm+L7MBiVhZMq0j63Nu12M/
        ySE5UWKBKKd8OesVdUX//cUTgPESmwku5ERxD3Lohl4s+Vl9GbwTVFGW8etHkm1J
        A/1VrcRXaYWfWLh9d05XNr1brJ9SbY2lAqaiME16ibXYdqAc2BmwgWXJ5QYYAMvh
        pn+kfm3XRzI1bU1cVrHqTD1EMrTYn0cIHQIDxBL/uZWlzGLJ0gv88IErcZurW5PY
```

```
        nOUoQJNsQ/uaxmg/jDd5bRqZZ8p8zwYaipINzz6aA6l3ucJbYfa6zr8CAwEAAaM/
        MD0wCQYDVR0TBAIwADARBglghkgBhvhCAQEEBAMCB4AwHQYDVR0OBBYEFD/MQE1b
        NQKUSHmQysu/a/NLcWoqMA0GCSqGSIb3DQEBCwUAA4IBAQAwbGiFWdSwkgyjr62/
        Pwb8A8ASAA17F5YD2QRSEh9vKL1Ff7A55/alYey5WvQvIiNSOMNajHJzZCsmI6HW
        w4RrcMo5EAhg+9sCgnFaBlnsXpTP7UC6vq9YoDaYIQ1qX/vFK9PMvV2KYA9zGuxP
        OX5KiMQ1oMv2y9J19SXeH4bC2bFzWlktQBy4OAHOmc9gAwgRd+irMWPGBV5Dv9eM
        TM6yvM5Lis2cuBsXHcpmmA8nWUGiEVKsXF7y4PsvZuA+izyJQQ1Can9NlA6keblx
        4GcMTdFet6WLZlrrtfNF2aYZzaT8Z6Xq5WzhZI0aCprrXxJX3LD1yLniYtv+I3sn
        tPec
        -----END CERTIFICATE-----"
}
```

Sample SOAP Parameters:

```
        <prot:Cert_Export>
          <username>cryptouser</username>
          <password>qwerty1234</password>
          <certname>pkcs1sample</certname>
          <!--Optional:-->
          <certformat></certformat>
       </prot:Cert_Export>
```

Output: Certificate and the optional private key in PKCS#1 or PKCS#8 format

<ns1:Cert_ExportResponse xmlns:ns1="http://dsws.org/protectappws/">-----BEGIN CERTIFICATE-----

MIIDvzCCAqegAwIBAgIDAJ5EMA0GCSqGSIb3DQEBCwUAMIGbMQswCQYDVQQGEwJV

[… sample truncated for brevity …]

OcqQnevrP4rbUC/5W6+gO0m5ZjMDKryAyW4RiNCboGKtVTVcz68J0+75RTvycjWK

ibEI

-----END CERTIFICATE-----</ns1:Cert_ExportResponse>

CertImport: Import certificate (and Optionally Private Key)

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/certImport

Input Parameters:

> username

- password
- certname – name of the certificate to import.
- certisdeletable (for SOAP)/isdeletable (for REST) – sets whether the certificate can be deleted via the API.
- certisexportable (for SOAP)/isexportable (for REST) – sets whether the certificate can be exported via the API.
- certificate – certificate to be imported, in PKCS1, PKCS#8, or PKCS#12 format.
- certpassword – optional, if password provided for certificate it must be Hex encoded.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf:

request:

```
{
  "Cert_Import": {
    "certname": "naanq",
    "username": "jcetest",
    "password": "asdf1234",
    "isdeletable": "true",
    "isexportable": "true",
    "certificate": "-----BEGIN CERTIFICATE-----
MIIDvzCCAqegAwIBAgIDAJ5EMA0GCSqGSIb3DQEBCwUAMIGbMQswCQYDVQQGEwJV
[… sample truncated for brevity … ]
OcqQnevrP4rbUC/5W6+gO0m5ZjMDKryAyW4RiNCboGKtVTVcz68J0+75RTvycjWK
ibEI
-----END CERTIFICATE-----
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAhLItSYS7WHe22H+VOyj5GlAkwcylRsCAl/kbLBUW5adSB5K3
[… sample truncated for brevity … ]
mzXC86k6UN7ya29wDFuWwLK+gnwD2THORrdk5U+1B0PywK4JdDOR
-----END RSA PRIVATE KEY-----
  }
}
```

response:

```
{
  "CertImportResponse": { "CertImportResponse": "true" }
}
```

Sample SOAP Parameters:

```
<prot:Cert_Import>
      <username>cryptouser</username>
      <password>qwerty1234</password>
      <certname>pkcs1samplevtN</certname>
      <certisdeletable>true</certisdeletable>
      <certisexportable>true</certisexportable>
      <certificate>-----BEGIN CERTIFICATE-----
            MIIDvzCCAqegAwIBAgIDAJ5EMA0GCSqGSIb3DQEBCwUAMIGbMQswCQYDVQQGEwJ
            V
            [… sample truncated for brevity … ]
            OcqQnevrP4rbUC/5W6+gO0m5ZjMDKryAyW4RiNCboGKtVTVcz68J0+75RTvycjW
            K
            ibEI
            -----END CERTIFICATE-----
            -----BEGIN RSA PRIVATE KEY-----
            MIIEowIBAAKCAQEAhLItSYS7WHe22H+VOyj5GlAkwcylRsCAl/kbLBUW5adSB5K
            3
            [… sample truncated for brevity … ]
            mzXC86k6UN7ya29wDFuWwLK+gnwD2THORrdk5U+1B0PywK4JdDOR
            -----END RSA PRIVATE KEY-----
      </certificate>
      <!--Optional:-->
      <certpassword></certpassword>
</prot:Cert_Import>
```

Output: boolean – indicates import success

<ns1:Cert_ImportResponse xmlns:ns1="http://dsws.org/protectappws/">true</ns1:Cert_ImportResponse>

NOTE: In order to import a PKCS12 certificate using web services you must provide certpassword field in import request. In this case certificate data i.e. <certificate> tag must be sent in Hex Format.

The sample certificates are included as a convenience. You can also use your own certificates, just be sure that your PKCS#12 certificate is encrypted using 3DES, otherwise you will see the error: “1559: Certificate could not be verified”.

If you need to create your own PKCS#12 certificate using 3DES, you can use openssl, with the following statement as a guide:

```
openssl pkcs12 -keypbe PBE-SHA1-3DES -certpbe PBE-SHA1-3DES -export -in cert.txt -inkey privateKeyInPKCS1 -out thepkcs12.cert -name someCommonNameForCert
```

CertificateSigningRequest: Sign a Certificate

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/signRequest

Input Parameters:

- userName
- password
- caName
- noOfExpireDays
- csrRequest
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
  "CertificateSigningRequest": {
    "userName": "abc",
    "password": "safenet@123",
```

```
    "caName": "xyz",
    "noOfExpireDays": "1",
    "csrRequest": "-----BEGIN CERTIFICATE REQUEST-----
\nMIICoDCCAYgCAQAwWzEPMA0GA1UEAxMGdGVzdENOMQkwBwYDVQQKEwAxCTAHBgNV\nBAsTADEJMAcG
A1UEBxMAMQkwBwYDVQQIEwAxCzAJBgNVBAYTAlVTMQ8wDQYJKoZI\nhvcNAQkBFgAwggEiMA0GCSqGSI
b3DQEBAQUAA4IBDwAwggEKAoIBAQDvn7+CireT\nJ620bg/mXmShkiZlmh71I1VsYQskvQK9i3wrFn6d
GEOAD5nhMPV5RPFelPcZYUnf\nIymzALj0W43YVn1UT8Y+g5KFWfGLZKeAa1myqXxB8VeYiPtcMFbdg1
lhRhGEvuZT\nNfFxSZ6hNLWyO6AuIHByHsh3yKU/jAQ1gQ4UIW0GLA/mqbqb73PNcJ4aoyNKLKN1\nJ+
Gw2ozL3wyFvw0NeJpfWmIA6ZkCbCBTT80cOxiQT9YMDtjCZfsZ3CyUyGzP5oLt\nPmbLzK1ZeXWUemPh
31CEPN5Fx7ju4Rp68XBaFs2XR6f8yIsw6e+FVd8rUC6psFDR\nTfla3vPgriO/AgMBAAGgADANBgkqhk
iG9w0BAQsFAAOCAQEAzCdipiN9JINe2y4o\nvHnNYG8nxPUBoJxgYbSxwvTGhnHEYEEEKheGGL19V0zL
vVgvb776ZsSoYS9KVSOg\njzsc6g1riI2aol1Pe/1eqdfT6f1prl8Q4yr7YAtFSAos1lqHOxXD91gvmU
/2496c\nU1OzLtjXuryyZvrqd6HwRaZloZw3Jf7japEr+1JT3h0O2CMQueWYxsB2cx7n2Tyj\nzK4bc8
GXeA2nt3kTTr+K2B3Cxs8cin8A0G+Zj5CPN9xNRMJQuQ5oz+V8zU6MD2XG\nm+TFs4wpmnk223glIP4G
l+oHGi9fqbiVkacGYt3uOdFEL6MK3A+UnNU+JcM5XOC+\nDQuwvA==\n-----END CERTIFICATE
REQUEST-----\n"
}
}
```

response:

```
{
  "CertificateSigningResponse": {
    "signedData": "
----BEGIN CERTIFICATE----
MIICyTCCAbGgAwIBAgIDAyLeMA0GCSqGSIb3DQEBCwUAMIGYMQswCQYDVQQGEwJV
----END CERTIFICATE----
"
  }
}
```

ChangePassword: Change Password Permission for User

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/users/changePassword

Input Parameters:

- adminUser
- adminPassword
- userName
- userPassword

- isPasswordChangeable – optional
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
  "ChangePasswordRequest": {
    "adminUser": "cryptouser",
    "adminPassword": "safenet@123",
    "userName": "cryptouser",
    "userPassword": "safenet@1234",
    "isPasswordChangeable": "false",
    }
}
```

response:

```
{
  "ChangePasswordResponse": {
    "description": "Information updated successfully"
  }
}
```

CreateCSRRequest: Create a New Cerfiticate Request

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/createCertRequest

Input Parameters:

- userName
- password
- keyName – RSA key.
- cnName – common name for the certificate.
- countryName – two letter code of the country.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
  "CreateCSRRequest": {
    "userName": "abc",
    "password": "safenet@123",
    "keyName": "RSA",
    "cnName": "xyz",
    "countryName": "IN"
  }
}
```

response:

```
{"CreateCSRResponse":
{"csrRequest":"----BEGIN CERTIFICATE REQUEST---
\nMIIBFTCBwAIBADBbMQ8wDv1cn4XdLHQ2g=\n---END CERTIFICATE REQUEST----\n"}
}
```

CreateGroup: Create a New Group

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/groups/createGroup

Input Parameters:

- adminUser
- adminPassword
- groupName – name of the group to be created.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
  "CreateGroupRequest": {
    "adminUser": "cryptouser",
    "adminPassword": "safenet@123",
```

```
    "groupName": "abcd",
  }
}
```

response:

```
{
  "CreateGroupResponse": {
    "groupName": "abcd",
    "description": "group created successfully."
  }
}
```

CreateUser: Create a New User

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/users/createUser

Input Parameters:

- adminUser
- adminPassword
- userName
- userPassword
- isPasswordChangeable – optional. Use this parameter to change the password, default value is `false`.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.
- CustomAttributeList – optional, use this parameter to add custom attributes to the user.

Sample REST call for cxf

request:

```
{
  "CreateUserRequest": {
    "adminUser": "cryptouser",
    "adminPassword": "safenet@123",
    "userName": "crypto14",
    "userPassword": "safenet@123",
    "isPasswordChangeable": "true",
```

```
    "CustomAttributeList":  {
    "CustomAttribute": [
      {
        "Name": "test1",
       "Value": "value1"
     },
     {
       "Name": "test2",
       "Value": "value2"
     }
   ]
  }
}
```

response:

```
{
  "CreateUserResponse": {
    "description": "user created successfully.",
    "userName": "crypto14"
  }
}
```

Sample SOAP Parameters:

```
<prot:CreateUser>
   <adminUser>adminUser</adminUser>
   <adminPassword>asdf1234</adminPassword>
    <userName>userToCreate</userName>
    <userPassword>asdf1234</userPassword>
    <!--Optional:-->
    <isPasswordChangeable>true</isPasswordChangeable>
    <!--Optional:-->
    <CustomAttributeList>
    <CustomAttribute>
   <Name>test1</Name>
   <Value>value1</Value>
```

```
    </CustomAttribute>
   </CustomAttributeList>
    <!--Optional:-->
    <certAlias></certAlias>
    <!--Optional:-->
    <certPassword></certPassword>
  </prot:CreateUser>
```

Output: User created successfully

<ns2:CreateUserResponse xmlns:ns2="http://dsws.org/protectappws/">User created successfully.</ns2:CreateUserResponse>

Decrypt: Decrypt Data Using Key Specified by Name

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/decrypt

Input Parameters:

- username
- password
- keyname – name of the key to be used for decryption.
- ciphertext – encrypted data represented in Hex.
- keyiv – optional/blank. If blank, use the key's default IV, else, specify in Hex characters. The number of Hex character must be specific to the transformation used.
- transformation – transformation to be used. For example: AES/ ECIES/CBC/PKCS5Padding. For information on supported ECIES transformations, refer to the CipherTrust Application Data Protection for Java User Guide.
- outputformat – optional, displays output in following formats: HEX, STR, and BASE64. Default is STR.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf:

request:

```
{
  "Decrypt": {
    "username": "cryptouser",
    "password": "safenet123",
```

```
    "keyname": "testKey",
    "keyiv": "12345678123456781234567812345678",
    "transformation": "AES/CBC/PKCS5Padding",
    "ciphertext": "100010EBF1B02D8CFAC300502398BD7FD5DB1A",
    "outputformat": "STR"
  }
}
```

response :

```
{
  "DecryptResponse": { "plainText": "hello world!" }
}
```

Sample SOAP Parameters:

```
      <prot:Decrypt>
         <username>cryptouser</username>
         <password>qwerty1234</password>
         <keyname>aes256vt</keyname>
         <ciphertext>10009046C980ECAFC6A79765A7ABAE01C846C5</ciphertext>
         <!--Optional:-->
         <keyiv></keyiv>
         <transformation>AES/CBC/PKCS5Padding</transformation>
         <!--Optional:-->
         <outputformat>STR</outputformat>
      </prot:Decrypt>
```

Output: Plaintext data

<ns1:DecryptResponse xmlns:ns1="http://dsws.org/protectappws/">0000111122223333</ns1:DecryptResponse>

NOTE: In almost all cases the keyiv and transformation should not be specified: It is a useful practice to utilize the Key Manager appliances capability to store the IV for the application and AES/CBC/PKCS5Padding – the default - is the most recommended cipher block mode.

DeleteGroup: Delete a Group

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/groups/deleteGroup

Input Parameters:

> adminUser

> adminPassword

> groupName – name of the group to be deleted.

> certAlias – optional, is a client certificate alias for making SSL connections.

> certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
  "DeleteGroupRequest": {
    "adminUser": "cryptouser",
    "adminPassword": "asdf1234",
    "groupName": "abcd",
  }
}
```

response:

```
{
  "DeleteGroupResponse": {
    "description": "group deleted successfully."
  }
}
```

DeleteUser: Delete a User

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/users/deleteUser

Input Parameters:

> adminUser

> adminPassword

> userName

- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
  "DeleteUserRequest": {
    "adminUser": "cryptouser",
    "adminPassword": "safenet@123",
    "userName": "cryptouser"
  }
}
```

response:

```
{
  "CreateUserResponse": {
    "description": "user created successfully.",
    "userName": "crypto14"
  }
}
```

Encrypt: Encrypt Data Using Key Specified by Name

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/encrypt

Input Parameters:

- username
- password
- keyname – name of the key to be used for encryption.
- plaintext – ASCII text to be encrypted, or hex if binary encryption is desired.
- keyiv – optional/blank. If blank, use the key's default IV, else, specify in Hex characters. The number of Hex character must be specific to the transformation used.
- transformation – transformation to be used. For example: AES/ ECIES/CBC/PKCS5Padding. For information on supported ECIES transformations, refer to the CipherTrust Application Data Protection for Java User Guide.
- certAlias – optional, is a client certificate alias for making SSL connections.

> certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf:

request:

```
{
  "Encrypt": {
    "username": "cryptouser",
    "password": "safenet123",
    "keyname": "testKey",
    "keyiv": "12345678123456781234567812345678",
    "transformation": "AES/CBC/PKCS5Padding",
    "plaintext": "hello"
  }
}
```

response:

```
{
  "EncryptResponse": { "cipherText": "100010C9CE1F70A6663BF3B9A2F62CD852F437" }
}
```

Sample SOAP Parameters:

```
      <prot:Encrypt>
         <username>cryptouser</username>
         <password>qwerty1234</password>
         <keyname>aes256vt</keyname>
         <plaintext>0000111122223333</plaintext>
         <!--Optional:-->
         <keyiv></keyiv>
         <!--Optional:-->
         <transformation>AES/CBC/PKCS5Padding</transformation>
      </prot:Encrypt>
```

Output: Encrypted data in Hex

<ns1:EncryptResponse xmlns:ns1="http://dsws.org/protectappws/">70C49C826D60E8564EDD51E8BF276C0FFF79D25420251D740D14C3919EC6663D</ns1:EncryptResponse>

NOTE: In almost all cases the keyiv and transformation should not be specified: It is a useful practice to utilize the Key Manager appliances capability to store the IV for the application and AES/CBC/PKCS5Padding – the default - is the most recommended cipher block mode.

FPE_Decrypt: Decrypt Data Using Format Preserving Encryption

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/fpe_decrypt

Input Parameters:

- username
- password
- keyname – name of the key.
- keyiv – optional, Hex encoded 56 bytes.
- tweakAlgo – optional, default is none.
- tweakData – If tweak data algorithm is "None" or absent, the value must be HEX encoded string representing 64 bit long (hence, HEX encoding will consume 16 characters). Tweak data is mandatory if Tweak Algo is given, else it is optional.
- cipherText – encrypted data represented in Hex.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
  "FPE_Decrypt": {
    "username": "jcetest",
    "password": "asdf1234",
    "keyname": "aes256vt",
```

```
    "keyiv":
"000102010405060708090001020304050607080900010203040506070809000102030405060708
900010203040506070809000102030405",
    "tweakAlgo": "none",
    "tweakData": "1a23456712345678",
    "cipherText": "363631343133"
  }
}
```

response:

```
{
  "DecryptFPEResponse": { "plainText": "23232332" }
}
```

Sample SOAP Parameters:

```
<prot:FPE_Decrypt>
        <username>jcetest</username>
        <password>asdf1234</password>
        <keyname>aes256vt</keyname>
<!--Optional:-->
<keyiv>000102010405060708090001020304050607080900010203040506
07080900010203040506070809000102030405</keyiv>
<!--Optional:-->
        <tweakAlgo>none</tweakAlgo>
        <tweakData>1a23456712345678</tweakData>
        <cipherText>363631343133</cipherText>
 </prot:FPE_Decrypt>
```

Output: Plaintext data

<ns2:FPE_DecryptResponse xmlns:ns2="http://dsws.org/protectappws/">34373038</ns2:FPE_DecryptResponse>

FPE_Encrypt: Encrypt Data Using Format Preserving Encryption

URL : <http/https>://<host-name>:<Port>/protectappws/services/rest/fpe_encrypt

Input Parameters:

- username
- password
- keyname – name of the key
- keyiv – optional, Hex encoded 56 bytes.
- tweakAlgo – optional, default is none.
- tweakData – If tweak data algorithm is "None" or absent, the value must be HEX encoded string representing 64 bit long (hence, HEX encoding will consume 16 characters). Tweak data is mandatory if Tweak Algo is given else it is optional.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.
- plaintext – data to be encrypted.

NOTE: In case if tweak data algorithm represents any valid algorithm, the tweak data value can be any ASCII string (not necessarily HEX). Tweak Data is first processed using Tweak Hash Algorithm and the result is truncated to 64 bits for input to the FPE algorithm.

Sample REST call for cxf:

request:

```
{
  "FPE_Encrypt": {
    "username": "jcetest",
    "password": "asdf1234",
    "keyname": "aes256vt",
     "keyiv":
      "000102010405060708090001020304050607080900010203040506
      0708090001020304050607080900010203040506",
    "tweakAlgo": "none",
    "tweakData": "1a23456712345678",
    "plaintext": "239494"
  }
}
```

response:

```
{
  "EncryptFPEResponse": { "cipherText": "100010C9CE1F70A6663BF3B9A2F62CD852F437"
}
}
```

Sample SOAP Parameters:

```
<prot:FPE_Encrypt>
    <username>jcetest</username>
    <password>asdf1234</password>
    <keyname>AESFPE</keyname>
    <tweakData>1234567812345678</tweakData>
    <plaintext>1234</plaintext>
</prot:FPE_Encrypt>
```

Output: Encrypted data in Hex

<ns2:FPE_EncryptResponse xmlns:ns2="http://dsws.org/protectappws/">34373038</ns2:FPE_EncryptResponse>

FPEFormatDecryption: Decrypt data using FPE While Preserving Format of Ciphertext

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/fpeFormat/decrypt

Input Parameters:

- userName
- password
- format – the format in which some part of ciphertext is to be kept intact. Valid values are: `LAST_FOUR`, `FIRST_SIX`, `FIRST_SIX_LAST_FOUR`, `FIRST_TWO_LAST_FOUR`, and `NONE`. Also, only the format which was used for encryption of the plaintext, can be used here for decryption. Default value is `NONE`.
- keyName – name of the key to be used for decryption.
- transformation – override the standard padding.
- data – data to be decrypted.
- tweakAlgo – optional, default is none.
- tweakData – optional, If tweak data algorithm is "None" or absent, the value must be HEX encoded string representing 64 bit long (hence, HEX encoding will consume 16 characters). Tweak data is mandatory if Tweak Algo is given, else it is optional.

- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
  "FPEFormatDecryptionRequest": {
    "userName": "cryptouser",
    "password": "safenet@123",
    "format": "LAST_FOUR",
    "keyName": "AESKey",
    "transformation": "FPE/AES/CARD10",
    "data": "713-456",
    "tweakData": "hello",
    "tweakAlgo": "SHA1",
  }
}
```

response:

```
{
  "FPEFormatDecryptionRequest": {
    "data": "453-456"
  }
}
```

FPEFormatEncryption: Encrypt Data Using FPE While Preserving Format of Plaintext

URL : <http/https>://<host-name>:<Port>/protectappws/services/rest/fpeFormat/encrypt

Input Parameters:

- userName
- password
- format – the format in which some part of input data is to be kept intact, that is, the selected part of the input data is not encrypted. Valid values are: `LAST_FOUR`, `FIRST_SIX`, `FIRST_SIX_LAST_FOUR`, `FIRST_TWO_LAST_FOUR`, and `NONE`. Default value is `NONE`.

- keyName – name of the key to be used for encryption.
- transformation – override the standard padding.
- plainText – data to be encrypted.
- tweakAlgo – optional, default is none.
- tweakData – optional, If tweak data algorithm is "None" or absent, the value must be HEX encoded string representing 64 bit long (hence, HEX encoding will consume 16 characters). Tweak data is mandatory if Tweak Algo is given, else it is optional.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

NOTE: In case if tweak data algorithm represents any valid algorithm, the tweak data value can be any ASCII string (not necessarily HEX). Tweak Data is first processed using Tweak Hash Algorithm and the result is truncated to 64 bits for input to the FPE algorithm.

Sample REST call for cxf

request:

```
{
  "FPEFormatEncryptionRequest": {
    "userName": "cryptouser",
    "password": "safenet@123",
    "format": "last_four",
    "keyName": "AESKey",
    "transformation": "FPE/AES/CARD10",
    "plainText": "713-456",
    "tweakData": "hello",
    "tweakAlgo": "SHA1",
  }
}
```

response:

```
{
  "FPEFormatEncryptionResponse": {
    "encryptedData": "163-456"
  }
```

```
}
```

GCM_Decrypt: Decrypt Data Using GCM Standards

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/gcmdecrypt

Input Parameters:

- username
- password
- keyname – name of the key.
- authtaglength – tag length to ensure the data is not accidentally altered, must be 32 to 128 bits and must be multiple of 8.
- iv – specify 1 to 16 bytes IV in Hex format. For example 697473617265616c6c79636f6f6c6976.
- aad – optional, the data to be passed to the recipient in plain text, needs to be “authenticated”.
- ciphertext – optional, encrypted data represented in Hex.
- transformation – optional, override the standard AES/GCM/NoPadding.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
"GCM_Decrypt":
      {
      "username": "crypto",
      "password": "asdf1234",
      "keyname": "testaesgcm",
      "authtaglength":"72",
      "iv": "31323334353637383132331212",
      "aad":"1115",
      "ciphertext": "2BDC88892AAB9EF9074860028433"
      }
}
```

response:

```
{
```

```
GCMDecryptResponse:
      {
      plainText: "hello"
      }-
}
```

Sample SOAP Parameters:

```
<prot:GCM_Decrypt>
      <username>crpto</username>
      <password>asdf1234</password>
      <keyname>testaesgcm</keyname>
      <authtaglength>72</authtaglength>
      <iv>313233343536373831323312 31212</iv>
<!--Optional:-->
      <aad>122131</aad>
      <ciphertext>A17769B0000714ECA6839172D719C8783BAEC0390438DE3F</ciphertext>
</prot:GCM_Decrypt>
```

Output: Plaintext data

<ns2:GCM_DecryptResponse xmlns:ns2="http://dsws.org/protectappws/">hello</ns2:GCM_DecryptResponse>

GCM_Encrypt: Encrypt Data Using GCM Standards

URL : <http/https>://<host-name>:<Port>/protectappws/services/rest/gcmencrypt

Input Parameters:

- username
- password
- keyname – name of the key.
- authtaglength – tag length to ensure the data is not accidentally altered, must be 32 to 128 bits and must be multiple of 8.
- iv – specify 1 to 16 bytes IV in Hex format. For example 697473617265616c6c79636f6f6c6976.
- aad – optional, the data to be passed to the recipient in plain text, needs to be “authenticated”.
- plaintext – optional, ASCII text to be encrypted, or hex if binary encryption is desired.

- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf:

request:

```
{
"GCM_Encrypt":
      {
      "username": "crypto",
      "password": "asdf1234",
      "keyname": "testaesgcm",
      "authtaglength":"72",
      "iv": "31323334353637383132331212",
      "aad":"1115",
      "plaintext": "hello"
      }
}
```

response:

```
{
GCMEncryptResponse:
      {
      cipherText: "2BDC88892AAB9EF9074860028433"
      }-
}
```

Sample SOAP Parameters:

```
      <prot:GCM_Encrypt>
            <username>crypto</username>
            <password>asdf1234</password>
            <keyname>testaesgcm</keyname>
            <authtaglength>72</authtaglength>
            <iv>31323334353637383132331212</iv>
      <!--Optional:-->
```

```
            <aad>122131</aad>
            <plaintext>hello</plaintext>
        </prot:GCM_Encrypt>
```

Output: Encrypted data in Hex

<ns2:GCM_EncryptResponse xmlns:ns2="http://dsws.org/protectappws/">A0387706C73C700F4DA3409057E30BBB1BB280562F2C4729</ns2:GCM_EncryptResponse>

GetAllGroupInfo: Get Information of All Groups

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/groups/info

Input Parameters:

- adminUser
- adminPassword
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf:

request:

```
{
"AllGroupInfoRequest":{
   "adminUser": "testadmin",
   "adminPassword": "test1234",
   "certAlias": "privatekey",
   "certPassword": "test1234"
   }
}
```

Sample REST response for cxf

response:

```
{"AllGroupInfoResponse": {"GroupInfoList": {"GroupInfo": [
      {
      "Group": "SQLGroup",
      "UserList": {"User":       [
```

```
         "dbusr",
         "nparasher"
      ]}
   },
      {
      "Group": "testGroup",
      "UserList": {"User":       [
         "testuser",
         "dbusr",
         "testdb"
      ]}
   },

      {
      "Group": "twgGroup",
      "UserList": {"User": "twguser"}
   }
]}}}
```

GetGroupInfo: Get Information of a Group

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/groups/groupInfo

Input Parameters:

- adminUser
- adminPassword
- groupName – name of the group for which information is to be requested.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf:

request:

```
{
"GroupInfoRequest":{
```

```
   "adminUser": "aduser",
   "adminPassword": "test1234",
   "groupName": "twgGroup",
   "certAlias": "privatekey",
   "certPassword": "test1234"
   }
}
```

Sample REST response for cxf

response:

```
{"GroupInfoResponse": {
   "Group": "twgGroup",
   "UserList": {"User": "twguser"}
}}
```

GetKeyAttributes: Get Custom Attributes of a Key

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/key/getkeyattributes

Input Parameters:

- userName
- password
- keyName – name of the key for which custom attributes are to be retrieved.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
      "GetKeyAttributes": {
            "userName": "test1",
            "password": "abcd1234",
            "keyName": "123425",
  }
```

```
}
```

response:

```
{"GetKeyAttributesResponse":
      {
            "keyAttributes": "{attr1=123456, attrw=234567}"
      }
}
```

Sample SOAP Parameters:

```
      <prot:GetKeyAttributes>
            <username>test1</username>
            <password>abcd1234</password>
            <keyName>123425</keyName>
      </prot:GetKeyAttributes>
```

Output:

<ns2:GetKeyAttributesResponse xmlns:ns2="http://dsws.org/protectappws/">{attr1=123456, attrw=234567}</ns2:GetKeyAttributesResponse>

NOTE: The list of attributes that we get in response is comma separated so any attribute (name and value) having special characters (particularly **,** and **{}** and **“”** and **=**) will be displayed as it is and will make the attribute difficult to read. So, it is recommended NOT to use special characters (particularly **,** and **{}** and **“”** and **=**) in the input.

HMAC: Perform Keyed Hash

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/hmac

Input Parameters:

- username
- password
- keyname – name of the new key to be used for HMAC.

- messagetext – data on which to perform HMAC.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf:

request:

```
{
  "HMAC": {
    "username": "cryptouser",
    "password": "safenet123",
    "keyname": "testHMACKey",
    "messagetext": "akhi"
  }
}
```

response:

```
{
  "HMACResponse": { "hmac_Response":
"100010F331F2F515D8D2A2CF3D84A23F82CDB4B19C6661" }
}
```

Sample SOAP Parameters:

```
      <prot:HMAC>
         <username>cryptouser</username>
         <password>qwerty1234</password>
         <keyname>hmacsha1vt</keyname>
         <messagetext>This is a message suitable for HMAC
signing...</messagetext>
      </prot:HMAC>
```

Output: HMAC in Hex

<ns1:HMACResponse xmlns:ns1="http://dsws.org/protectappws/">CDBE56D386CDD6D0EBBBC481ACCA84CD60CE2919 </ns1:HMACResponse>

HMACVerify: Verify Keyed Hash

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/hmacVerify

Input Parameters:

- username
- password
- keyname – name of the key to be used for HMAC.
- messagetext – data on which to perform HMAC.
- mac – message authentication code for verification.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
  "HMAC_Verify": {
    "username": "cryptouser",
    "password": "safenet123",
    "keyname": "testHMACKey",
    "messagetext": "akhi",
    "mac": "100010F331F2F515D8D2A2CF3D84A23F82CDB4B19C6661"
  }
}
```

response:

```
{
  "HMACVerifyResponse": { "hmac_Verify_Result": "true" }
}
```

Sample SOAP Parameters:

```
      <prot:HMAC_Verify>
         <username>cryptouser</username>
         <password>qwerty1234</password>
         <keyname>hmacsha1vt</keyname>
```

```
         <messagetext>This is a message suitable for HMAC
signing...</messagetext>
         <mac>CDBE56D386CDD6D0EBBBC481ACCA84CD60CE2919</mac>
      </prot:HMAC_Verify>
```

Output: boolean – result of HMAC verification

<ns1:HMAC_VerifyResponse
xmlns:ns1="http://dsws.org/protectappws/">true</ns1:HMAC_VerifyResponse>

keyExport: Export Key from the Key Management Appliances

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/keyExport

Input Parameters:

- username
- password
- keyname – name of the key to export.
- keyformat – optional, format of the RSA or EC key to export. Valid values: PEM-PKCS#1 (default), PEM-PKCS#8, and PEM-SEC1.
- keytype – optional, type of the RSA key to export. Valid value: public or private.

> **NOTE:** When exporting private EC keys, the key format must be set to PEM-SEC1 or PEM-PKCS#8.

- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf

Symmetric key

request: {

```
  "Key_Export": {
    "username": "cryptouser",
    "password": "safenet123",
    "keyname": "testKey"
    "keyformat":"PEM-PKCS#8"
    "keytype":"Public"
```

```
  }
}
```

response:

```
{
  "KeyExportResponse": {
    "keyExportResponse":
"0B052C31FADE927B245E415C731EB5E9EF17665D72755515F7501C8C5104DE80"
  }
}
```

Asymmetric key (public key)

request: {

```
  "Key_Export": {
      "username": "NewUser",
      "password": "abcd1234",
      "keyname": "RSA_2048_Key_version#2",
      "keyformat" : "PEM-PKCS#1",
      "keytype" : "public"
      }
}
```

response:

```
{
      "KeyExportResponse": {
            "keyExportResponse":
                "-----BEGIN RSA PUBLIC KEY-----
                \nMIIBCgKCAQEAnVHvexwnwh/Ryzg7jJN5V0M97Y+F6VQEHyX9CsiJemQaS8Xhs6f
                t\n3PyhEZ168WVDoqk6RZcTvX+4VkaemU/BkrRPUwR4FT5gWqVV6h+tP7zO3tcyCG
                3B\nejchO8091IGMgb4e6/HvPV1ozn94ZY5n7uXNpZucRbunn6B3+fMCTs1jcquF8
                vEm\nVZ7cbIU6EHwxScNHAXffmgQDZuXOa57PhU38GX81Min/mbpyGPif9ivkBuAt
                25yw\norQo/AqaoRNmWnFnlVTXwXL/iwYycD94P9Mkbq2IOXprWB8A7/Al73lHfYh
                y058T\nJv0o+bh01i8pRVMrILzNnqTQ5uELGPUSoQIDAQAB\n
                -----END RSA PUBLIC KEY-----"
                }
}
```

Sample SOAP Parameters:

```
      <prot:key_Export>
```

```
        <username>cryptouser</username>
        <password>qwerty1234</password>
        <keyname>aes256vt</keyname>
        <keyformat>PEM-PKCS#8</keyformat>
        <keytype>private</keytype>
    </prot:key_Export>
```

Output: Key bytes in Hex for AES Keys and HMAC keys, PEM encoded for RSA keys

<ns2:Key_ExportResponse xmlns:ns2="http://dsws.org/protectappws/">

-----BEGIN PRIVATE KEY-----

MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAJ0YqKNY1BEgctx2

je8QNie25JIHiODsmNqN3gwDRSehWhJB6neFawkslEhCLzTI+uJPcWquRirs2yo3

k9AkqnY1/7FvzcnsRgaPmcN2jolEs+DB8VWEQPMQP1IM/cF11grly5bgIXbHZww5

ZPI+EqyR2fk7UE3wS1GTA2E5ZSDZAgMBAAECgYAgJj+8zhz4NE/NVmXNmt+6WO+G

Awy8Gy9c6tOaWcD5T0gDnIHDXAZ/piPidcj7YHYXgnKPhoLNnhNsHFufqSFJJcLL

+dpm+oLW/6n43prltd8m4Yv5WytbhVzurv/j00hpptaE9qPNOpnqs42+mwMFH9gp

vB6TcbPLTue6nNsiAQJBAMySGeCMpMed8zzd0ojLoFxzn4MVYTzbZcCP8bSHg2+b

jOYLYhn30++f4gSmSrfucnYFfIZiQ1UdkjTJ9S1zr0kCQQDElytcIZjLmSiTsGb0

QWo6ihIMVnCXoF27VEpbsh9siEZHNfDgkKJWw3moZ+FOYBUZ0gOI3sekbnj/pUZl

KZURAkAKdCjYXbeJ79tfL+KzVNyD/PCYzT3j0z5W4cQzOn2P9X30v34q34w4AY6W

OD03OvskZtFfOKRCNDmlcvCRewjpAkAObbHm/yhBuSn7PPUxdCOhzERq12VIG6MF

frg+ZqJbytytmU3mjXb8uzgQnh8xK2ghDwLnZsJsVRdYW/8Aqb6hAkEAg9avKm7B

skk6IC178u3QD2XLSu/G8x0BJwmcr32qNdw7Wg4nK8NlPCLKeyNZ8CWNokTjYilr

nfdWKZiotldtYQ==

-----END PRIVATE KEY-----

</ns2:Key_ExportResponse>

KeyGen: Generate Keys

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/keyGen

Input Parameters:

- username
- password

- keyname – name of the new key to be created.
- keyalgorithm – must be one of: RSA-2048, RSA-3072, RSA-4096, AES-128, AES-192, AES-256,

  HmacSHA1, HmacSHA256, HmacSHA384, HmacSHA512, EC-secp224k1-225, EC-secp224r1-224, EC-secp256k1-256, EC-secp384r1-384, EC-secp521r1-521, EC-prime256v1-256, EC-brainpoolP224r1-224, EC-brainpoolP224t1-224, EC-brainpoolP256r1-256, EC-brainpoolP256t1-256, EC-brainpoolP384r1-384, EC-brainpoolP384t1-384, EC-brainpoolP512r1-512, and EC-brainpoolP512t1-512.
- keyisdeletable – set if the key will be deletable via the API – boolean, default is false.
- keyisexportable – set if the new key will be exportable via the API – boolean, default is false.
- keyisversioned – set if the new key will be a versioned key – boolean, default is false.
- keytemplate – identify a template to use as a basis for the key generation.
- keyistemplate – set if generating a template rather than a key - boolean, default is false.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf:

request:

```
{
  "Key_Gen": {
    "username": "cryptouser",
    "password": "safenet123",
    "keyname": "TestKey",
    "keyalgorithm": "AES-256",
    "keyisdeletable": "false",
    "keyisexportable": "false",
    "keyisversioned": "false",
    "keyistemplate": "true"
  }
}
```

response:

```
{
  "KeyGenResponse": { "keyGenResponse": "true" }
}
```

Sample SOAP Parameters:

```
<prot:Key_Gen>
      <username>cryptouser</username>
      <password>qwerty1234</password>
      <keyname>testkey</keyname>
      <keyalgorithm>AES-256</keyalgorithm>
      <keyisdeletable>true</keyisdeletable>
      <keyisexportable>true</keyisexportable>
      <keyisversioned>true</keyisversioned>
      <!--Optional:-->
      <keytemplate></keytemplate>
      <!--Optional:-->
      <keyistemplate>false</keyistemplate>
</prot:Key_Gen>
```

Output: boolean – indicate if key creation was successful

<ns1:Key_GenResponse xmlns:ns1="http://dsws.org/protectappws/">true</ns1:Key_GenResponse>

ChangeKeyOwner: Change the Owner of a Key

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/users/changeKeyOwner

Input Parameters:

- adminUser – name of the admin user initiating the key ownership change request. The admin user must also be the key owner.
- adminPassword – password of the admin user.
- userName – name of the new owner of the key.
- keyName – name of the key to be reassigned.

Sample REST call for cxf:

request:

```
{
  "ChangeKeyOwnerRequest": {
    "adminUser": "testadmin_original_owner",
```

```
    "adminPassword": "test123",
    "userName": "Testuser1_new_owner",
    "keyName": "Test-AES-256",
    }
}
```

response:

```
{
  " ChangeKeyOwnerResponse": {" description": " Information updated successfully"}
}
```

GetKeyNames: Get Key Names Associated with a User

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/getkeynames

Input Parameters:

> username

> password

Sample REST call for cxf:

request:

```
{
  "Get_KeyNames": {
    "username": "testuser",
    "password": "testpasscode",
  }
}
```

response:

```
{
  "GetKeyNamesResponse": {"keyNames": "[hmacSHA512keyv, ariakeyv,
testaeskeyonce,… <…sample truncated for brevity…>… aesoncekeyv,
aesoncekeyvImport]"}
}
```

Sample SOAP Parameters:

```
<prot:GetKeyNames>
          <username>testuser</username>
          <password>testpasscode</password>
        </prot:GetKeyNames>
```

Output: boolean

<ns2:GetKeyNamesResponse xmlns:ns2="http://dsws.org/protectappws/">[secp256k1_ecc_key_5v, <…sample truncated for brevity…>,aesoncekeyvImport]</ns2:GetKeyNamesResponse>

GenerateKeyVersion: Generate Version of a Key

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/key/generatekeyversion

Input Parameters:

- userName
- password
- keyName – name of the key to generate its new version.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf:

request:

```
{
"GenerateKeyVersion" : {
    "userName": "NewUser",
    "password": "asdf1234",
    "keyName": "GenVersionofKey"
    }
}
```

response:

```
{
  "GenerateKeyVersionResponse": {"generateKeyVersionResponse": true}
}
```

Sample SOAP Parameters:

```
<prot:GenerateKeyVersion>
        <userName>NewUser</userName>
        <password>asdf1234</password>
        <keyName>GenVersionofKey</keyName>
</prot:GenerateKeyVersion>
```

Output: boolean – indicate if key version generation was successful

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>

<ns2:GenerateKeyVersionResponse xmlns:ns2="http://dsws.org/protectappws/">true</ns2:GenerateKeyVersionResponse>

</soap:Body>

</soap:Envelope>

modifyKeyPermission: Modify the Group Permissions of a Key

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/key/modifyKeyPermission

Input Parameters:

- userName – name of the user which owns the key.
- password – password associated with the userName.
- keyName – name of the key, usage permissions for which to be modified for the associated groups.
- permission – tag using which the key permissions are set for each group. Permission for multiple groups can be set in a single request.
- groupName – name of the group for which the key permissions are to be set.
- export – set it to `true` to have key export permission for the group using the key. Possible values: true or false. Default value is `false`.
- encrypt – set it to `true` to have encrypt permission for the group using the key. Possible values: true or false. Default value is `false`.
- decrypt– set it to `true` to have decrypt permission for the group using the key. Possible values: true or false. Default value is `false`.
- mac– set it to `true` to have MAC operation permission for the group using an Hmac key. Possible values: true or false. Default value is `false`.
- macv– set it to `true` to have MAC Verify permission for the group using an Hmac key. Possible values: true or false. Default value is `false`.

- sign– set it to `true` to have signing operation permission for the group using an Hmac key. Possible values: true or false. Default value is `false`.
- signv– set it to `true` to have sign verifying operation permission for the group using an Hmac key. Possible values: true or false. Default value is `false`.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

NOTE: Values for all the XML tags are to be provided in the XML request, even if a particular permission is not to be modified. Example: enter `true` for the encrypt parameter even if the group has encryption permission for the key, as leaving it `blank` will set the permission to `false`.

Sample REST call for cxf:

request:

```
{
   "GroupPermissionsUpdateRequest": {
     "userName": "twguser",
     "password": "asdf1234",
     "keyName": "Test_key_JCE",
     "permissions": {
         "permission": [
                 {
                       "groupName": "Group1",
                        "export": "true",
                        "encrypt": "false",
                        "decrypt": "true",
                        "mac": "",
                       "macv": "",
                        "sign": "false",
                        "signv": "false"
                  },
                  {
                      "groupName": "Group2",
```

```
                        "export": "false",
                        "encrypt": "false",
                        "decrypt": "true",
                        "mac": "",
                        "macv": "",
                        "sign": "false",
                        "signv": "false"
                    }
                ]
            }
        }
}
```


response:


```
{
 " GroupPermissionUpdateResponse":{"result": "permissions are updated
successfully."}
}
```


KeyImport: Import Key into the Key Management Appliances

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/keyImport

Input Parameters:

- username
- password
- keyname – name of the key to import.
- keyalgorithm – options are: AES, RSA, EC, and Hmac.
- keyisdeletable – whether the imported key will be deletable via the API – boolean, default is false.
- keyisexportable – whether the imported key will be exportable via the API - boolean, default is false.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.
- keybytes – the bytes for key to be imported.

Sample REST call for cxf:

request:

```
{
  "Key_Import": {
    "username": "cryptouser",
    "password": "safenet123",
    "keyname": "gddh",
    "keyalgorithm": "AES",
    "keyisdeletable": "true",
    "keyisexportable": "true",
    "keybytes": "110DAB8A90056E5255ECD84AF224543A"
  }
}
```

response:

```
{
  "KeyImportResponse": { "keyImportResult": "true" }
}
```

Sample SOAP Parameters:

```
      <prot:Key_Import>
         <username>cryptouser</username>
         <password>qwerty1234</password>
         <keyname>aes256vtimported</keyname>
         <keyalgorithm>AES</keyalgorithm>
         <keyisdeletable>true</keyisdeletable>
         <keyisexportable>true</keyisexportable>

<keybytes>68711F03ABEE8B460509D5D54E7C70D3A78ABE21572746D31C433A797093B2CC
        </keybytes>
      </prot:Key_Import>
```

Output: boolean – indicate if key import was successful

<ns1:Key_ImportResponse xmlns:ns1="http://dsws.org/protectappws/">true</ns1:Key_ImportResponse>

PRNG: Pseudo Random Number Generator

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/prng

Input Parameters:

- username
- password
- length – the number of random bytes to be generated.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf:

request:

```
{
  "prng": {
    "username": "cryptouser",
    "password": "safenet123",
    "length": "20"
  }
}
```

response:

```
{
  "PRNGResponse": { "prngResponse": "8F542DB70F12FE3CED68E76FDCBF32BC84696905" }
}
```

Sample SOAP Parameters:

```
<prot:PRNG>
      <username>cryptouser</username>
      <password>qwerty1234</password>
      <length>4</length>
</prot:PRNG>
```

Output: Hex of random bytes

<ns1:PRNGResponse xmlns:ns1="http://dsws.org/protectappws/">725C241D</ns1:PRNGResponse>

NOTE: The length of the returned random bytes is represented in Hex and the length of the Hex value is twice as long as the number of actual bytes.

RSASign: Sign Message Text Using RSA Private Key

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/rsaSign

Input Parameters:

- username
- password
- keyname – name of RSA key pair containing private key.
- messagetext – message to sign.
- transformation – provide from one of the following: RSA, SHA1withRSA, SHA256withRSA, SHA384withRSA, SHA512withRSA, SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, and SHA512withRSAPSSPadding. For complete list of supported transformations, refer to the *Supported Algorithm* section in the *Signing and Verifying* Chapter of the *CADP for Java User Guide*.
- saltlength – optional, length of salt to be used for sign operation.
- format – optional, supported signing CMS formats are: cms/detached/der/enveloped, cms/detached/der, cms/detached/smime/enveloped and cms/detached/smime.
- messageformat – optional, supported message formats: HEX and STR. Default is STR.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

- The saltlength parameter is supported only with: SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, and SHA512withRSAPSSPadding transformations.
- The saltlength and format parameters cannot be used simultaneously.
- The messagetext must be in the same format as specified in the messageformat parameter.
- CMS formats are not supported with the following transformations: SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding,

SHA512withRSAPSSPadding, RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512.

- For KeySecure 8.12.5 onward, following transformations are added to sign the data based on pre-calculated hash: RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512. These transformations support saltlength parameter.

RSA_Sign Sample REST call with format

request:

```
{
  "RSA_Sign": {
    "username": "cryptouser",
    "password": "safenet123",
    "keyname": "certpkcs12",
    "messagetext": "akhi",
    "transformation": "SHA1withRSA",
    "format": "cms/detached/der/enveloped",
    "messageformat": "STR"
  }
}
```

response:

```
{
  "RSASignResponse": { "rsa_SignResponse":
"100010A72DFA9CD79662C60F3B6C75FE6EC9893F35C1B03670FA5AE8354E3E290C1468151530F2B
EC6F7F4801A063F216D0D9D48CD8734C6F31739E7E00C0815698232B89BAFE8B43979E9C0D591A33
3C41B9EE28E55A2FFB8EED5A6359931D9DB48A6765EDFBB8F2AB2800DF75EA7A53C6F586FAFC6107
9703AB28A58A306931DC6C22CD3771B6F7CE001CDC441BB7A7629BECC3F92EB959031D6483717FD8
CF4266B4B6335D84A0EA0A087093F3C4778B65E7AA0FD86A1C8BF111FCF7CE4B96A12B3C05131ABB
CDDF33530DE6EDF889EF65739B6CBFD1722A25BCAD6657C098F2A2642AB5D478CBC1E0433635C3C4
C495703042032F6EA91E477EDBA2D204A1DADD8" }
}
```

RSA_Sign Sample REST call with saltlength

request:

```
{
```

```
  "RSA_Sign": {
    "username": "cryptouser",
    "password": "safenet123",
    "keyname": "certpkcs12",
    "messagetext": "akhi",
    "transformation": "SHA1withRSAPSSPadding",
    "saltlength": "40",
    "messageformat": "STR"
  }
}
```

response:

```
{
  "RSASignResponse": { "rsa_SignResponse":
"100010A72DFA9CD79662C60F3B6C75FE6EC9893F35C1B03670FA5AE8354E3E290C1468151530F2B
EC6F7F4801A063F216D0D9D48CD8734C6F31739E7E00C0815698232B89BAFE8B43979E9C0D591A33
3C41B9EE28E55A2FFB8EED5A6359931D9DB48A6765EDFBB8F2AB2800DF75EA7A53C6F586FAFC6107
9703AB28A58A306931DC6C22CD3771B6F7CE001CDC441BB7A7629BECC3F92EB959031D6483717FD8
CF4266B4B6335D84A0EA0A087093F3C4778B65E7AA0FD86A1C8BF111FCF7CE4B96A12B3C05131ABB
CDDF33530DE6EDF889EF65739B6CBFD1722A25BCAD6657C098F2A2642AB5D478CBC1E0433635C3C4
C495703042032F6EA91E477EDBA2D204A1DADD8" }
}
```

RSA_Sign SOAP Sample with format:

```
      <prot:RSA_Sign>
         <username>cryptouser</username>
         <password>qwerty1234</password>
         <keyname>certpkcs12</keyname>
         <messagetext>13123sdssdadasddsdsada</messagetext>
         <transformation>SHA1withRSA</transformation>
         <!--Optional:-->
         <format>cms/detached/der/enveloped</format>
         <!--Optional:-->
         <messageformat>STR</messageformat>
      </prot:RSA_Sign>
```

Output: signature in Hex

<ns1:RSA_SignResponse xmlns:ns1="http://dsws.org/protectappws/">32D4F93F540E465FE0DA63C29BFD28053F75C131598CC7B85493BF7BACEF5E14A1F2787F8D284E33269462ACA047DF1BF47442D70151D9471B9782B61477D514263B8FFFA0174650B4E8E7901A124B49C1E50796CECC1E83685124132855FF7BFB175F108BB026AEAA185660BB2CD4C7C85DFB557DDE58692A1C7EA769F38D877D220DC24AA5C5116E46E82E847FEDB99B45FA21128A94E86ACF8926E39D795F0C51D919DE9D9EC3647B22C26EB2A67931E52F294AE96FECE577DF9823F7A4EE69216166FF3940616253BF03ACDB32E414E29C0FFE8D5C2C0C0C1098CE2FCFA173EDCCDF3939FD5E691CBF2CF638EA26ED1A0E8E2ED2F727E300B3FCFA17ED18</ns1:RSA_SignResponse>

RSA_Sign SOAP Sample with saltlength:

```
<prot:RSA_Sign>
   <username>cryptouser</username>
   <password>qwerty1234</password>
   <keyname>certpkcs12</keyname>
   <messagetext>13123sdssdadasddsdsada</messagetext>
   <transformation>SHA1withRSAPSSPadding</transformation>
   <!--Optional:-->
   <saltlength>40</saltlength>
   <!--Optional:-->
   <messageformat>STR</messageformat>
</prot:RSA_Sign>
```

Output: signature in Hex

<ns1:RSA_SignResponse xmlns:ns1="http://dsws.org/protectappws/">32D4F93F540E465FE0DA63C29BFD28053F75C131598CC7B85493BF7BACEF5E14A1F2787F8D284E33269462ACA047DF1BF47442D70151D9471B9782B61477D514263B8FFFA0174650B4E8E7901A124B49C1E50796CECC1E83685124132855FF7BFB175F108BB026AEAA185660BB2CD4C7C85DFB557DDE58692A1C7EA769F38D877D220DC24AA5C5116E46E82E847FEDB99B45FA21128A94E86ACF8926E39D795F0C51D919DE9D9EC3647B22C26EB2A67931E52F294AE96FECE577DF9823F7A4EE69216166FF3940616253BF03ACDB32E414E29C0FFE8D5C2C0C0C1098CE2FCFA173EDCCDF3939FD5E691CBF2CF638EA26ED1A0E8E2ED2F727E300B3FCFA17ED18</ns1:RSA_SignResponse>

Sign: Sign Message Text Using RSA or EC Private Key

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/Sign

Input Parameters:

- username
- password
- keyname – name of RSA or EC key pair containing private key.
- messagetext – message to sign. It must be in the same format as specified in the messageformat parameter.
- transformation – provide one of the RSA/EC signing transformations supported by CADP for Java.
- format – optional, supported signing CMS formats are: cms/detached/der/enveloped, cms/detached/der, cms/detached/smime/enveloped and cms/detached/smime

- The transformations: SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, SHA512withRSAPSSPadding, ECDSA, SHA1withECDSA, SHA256withECDSA, SHA384withECDSA, SHA512withECDSA, RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512 do not support the CMS formats.
- For KeySecure 8.12.5 onward, following transformations are added to sign the data based on pre-calculated hash: RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512.

- messageformat – optional, supported message formats: HEX and STR. Default is STR.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
"Sign": {
"username": "admin",

"password": "admin",

"keyname": "ecc_testkey",

"messagetext": "126352152154",

"transformation": "SHA512withECDSA"

}
}
```

"SignResponse":

```
{
"signResponse":

"10001030818402400212 89F54A83BADACC8FD634A7606531204FECCC834AB649C86BF7D4A3612FC101D
FF1BE16E5354406D934A655705D8D13642763EBD4AECA84C283D399E573AA02405EEFEC37FB16D3CFDCB
F8ACE69E55C8A5F9EA224326EEF0FAF32CE2FA0DCBE7C063A572FA2F209D92A022945F1ED75A80B29404
9D499567CBA8D5993962FCC21"
    }
}
```

Sample SOAP Parameters:

```
<prot:_Sign>
   <username>cryptouser</username>
   <password>qwerty1234</password>
   <keyname>certpkcs12</keyname>
   <messagetext>13123sdssdadasddsdsada</messagetext>
   <transformation> SHA384withECDSA</transformation>
   <!--Optional:-->
   <messageformat>STR</messageformat>
</prot:_Sign>
```

Output: signature in Hex

<ns1:_SignResponse xmlns:ns1="http://dsws.org/protectappws/">32D4F93F540E465FE0DA63C29BFD28053F75C131598CC7B85493BF7BACEF5E14A1F2787F8D284E33269462ACA047DF1BF47442D70151D9471B9782B61477D514263B8FFFA0174650B4E8E7901A124B49C1E50796CECC1E83685124132855FF7BFB175F108BB026AEAA185660BB2CD4C7C85DFB557DDE58692A1C7EA769F38D877D220DC24AA5C5116E46E82E847FEDB99B45FA21128A94E86ACF8926E39D795F0C51D919DE9D9EC3647B22C26EB2A67931E52F294AE96FECE577DF9823F7A4EE69216166FF3940616253BF03ACDB32E414E29C0FFE8D5C2C0C0C1098CE2FCFA173EDCCDF3939FD5E691CBF2CF638EA26ED1A0E8E2ED2F727E300B3FCFA17ED18</ns1:_SignResponse>

RSAVerify: Verify the Signature of Message Text Using RSA Public key

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/rsaVerify

Input Parameters:

> username

- password
- keyname – name of RSA key pair containing public key.
- messagetext – message for signing verification.
- messageformat – supported message formats: HEX and STR. Default is STR
- signature – signature of RSA signing for verification in Hex.
- transformation – provide from one of the following: RSA, SHA1withRSA, SHA256withRSA, SHA384withRSA, SHA512withRSA, SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, and SHA512withRSAPSSPadding. For complete list of supported transformations, refer to the Supported Algorithm section in the Signing and Verifying Chapter of the CADP for Java User Guide.
- saltlength – optional, length of salt to be used for sign verification operation.
- format – optional, supported signing CMS formats are: cms/detached/der/enveloped, cms/detached/der, cms/detached/smime/enveloped and cms/detached/smime
- caname – optional, name of the CA.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

- The saltlength parameter is supported only with: SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, and SHA512withRSAPSSPadding transformations.
- The saltlength and format parameters cannot be used simultaneously.
- The messagetext must be in the same format as specified in the messageformat parameter.
- CMS formats are not supported with the following transformations: SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, SHA512withRSAPSSPadding, RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512.
- For KeySecure 8.12.5 onward, following transformations are added to verify the data based on pre-calculated hash: RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512. These transformations support saltlength parameter.

RSA_Verify Sample REST call with format

request:

```
{
  "RSA_Verify": {
    "username": "cryptouser",
    "password": "abcd123",
    "keyname": "certpkcs12",
    "messagetext": "13123sdssdadasddsdsada",
    "signature": "
3082087B06092A864886F70D010703A082086C30820868020100318201C0308201BC0201003081A3
30819B310B30090603550406130255533110B30090603550408130243413115301306035504071 30C
526564776F6F642043697479311430120603550 40A130B536166656E657420496E63311430120603
55040B130B456E67696E656572696E67311230100603550403140973616D706C655F636131283026
06092A864886F70D010901161973616D706C655F636140736166656E65742D696E632E636F6D0203
00A6E9300D06092A8648
[… sample truncated for brevity … ]
90E82E619499D6A0B54E084540A2B23AEC0E1943738433A0EA0E4FC7DF1449F9E55849302A17180E
4A678F435EF21F638C5A2746250DDDDD73B19953B03DA286ADB4BBDAF9C77EFC4E5032",
    "transformation": "SHA1withRSA",
    "format": "cms/detached/der/enveloped",
    "caname": "sample_ca"
  }
}
```

response:

```
{
  "RSA_Verify_Response": { "rsaVerifyResponse": "true" }
}
```

RSA_Verify Sample REST call with saltlength

request:

```
{
  "RSA_Verify": {
    "username": "cryptouser",
    "password": "abcd123",
    "keyname": "certpkcs12",
    "messagetext": "13123sdssdadasddsdsada",
```

```
    "signature": "
3082087B06092A864886F70D010703A082086C308208680201003182 01C0308201BC0201003081A3
30819B310B3009060355040613025553310B3009060355040813024341311530130603550407130C
526564776F6F64204369747931143012060355040A130B536166656E657420496E63311430120603
55040B130B456E67696E656572696E6731123010060355040314097361 6D706C655F636131283026
06092A864886F70D010901161973616D706C655F636140736166656E65742D696E632E636F6D0203
00A6E9300D06092A8648
[… sample truncated for brevity … ]
90E82E619499D6A0B54E084540A2B23AEC0E1943738433A0EA0E4FC7DF1449F9E55849302A17180E
4A678F435EF21F638C5A2746250DDDDD73B19953B03DA286ADB4BBDAF9C77EFC4E5032",
    "transformation": "SHA1withRSAPSSPadding",
    "saltlength": "40"
  }
}
```

response:

```
{
  "RSA_Verify_Response": { "rsaVerifyResponse": "true" }
}
```

RSA_Verify SOAP Sample with format:

```
      <prot:RSA_Verify>
            <username>cryptouser</username>
            <password>qwerty1234</password>
            <keyname>certpkcs12</keyname>
            <messagetext>13123sdssdadasddsdsada</messagetext>
            <signature>3082087B06092A864886F70D010703A082086C3082086802010031820 1
            C0308201BC0201003081A330819B310B3009060355040613025553310B30090603550
            4081302434131153013060355040713 0C526564776F6F64204369747931143012060 3
            55040A130B536166656E657420496E6331143012060355040B130B456E67696E65657
            2696E6731123010060355040314097361 6D706C655F6361312830260 6092A864886F7
            0D010901161973616D706C655F636140736166656E65742D696E632E636F6D020300A
            6E9300D06092A86488[… sample truncated for brevity … ]
            DACAAE8733FB6A9CD52E3B673C8E0DBB4E100A7EDE8B6DA586ADFAF38DA8975616F0B
            E035CB8B6D1514F1A6F3423180F402508A2F8E89C564</signature>
            <transformation>SHA1withRSA</transformation>
            <!--Optional:-->
            <format>cms/detached/der/enveloped</format>
            <!--Optional:-->
```

```
        <caname>sample_ca</caname>
    </prot:RSA_Verify>
```

Output: boolean – result of verification

<ns1:RSA_VerifyResponse xmlns:ns1="http://dsws.org/protectappws/">true</ns1:RSA_VerifyResponse>

RSA_Verify SOAP Sample with saltlength:

```
    <prot:RSA_Verify>
        <username>cryptouser</username>
        <password>qwerty1234</password>
        <keyname>certpkcs12</keyname>
        <messagetext>13123sdssdadasddsdsada</messagetext>
        <signature>3082087B06092A864886F70D010703A082086C30820868020100318201
        C0308201BC0201003081A330819B310B300906035504061302553310B30090603550
        40813024341311530130603550407130C526564776F6F64204369747931143012060 3
        55040A130B536166656E657420496E6331143012060355040B130B456E67696E65657
        2696E6731123010060355040314097361 6D706C655F6361312830260 6092A864886F7
        0D010901161973616D706C655F636140736166656E65742D696E632E636F6D020300A
        6E9300D06092A86488[… sample truncated for brevity … ]
        DACAAE8733FB6A9CD52E3B673C8E0DBB4E100A7EDE8B6DA586ADFAF38DA8975616F0B
        E035CB8B6D1514F1A6F3423180F402508A2F8E89C564</signature>
        <transformation>SHA1withRSAPSSPadding</transformation>
        <!--Optional:-->
        <saltlength>40</saltlength>
    </prot:RSA_Verify>
```

Output: boolean – result of verification

<ns1:RSA_VerifyResponse xmlns:ns1="http://dsws.org/protectappws/">true</ns1:RSA_VerifyResponse>

SignVerify: Verify the Signature of Message Text Using RSA or EC Public key

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/SignVerify

Input Parameters:

- username
- password

- keyname – name of RSA or EC key pair containing public key.
- messagetext – message for sign verification. It must be in the same format as specified in the messageformat parameter.
- messageformat – supported message formats: HEX and STR. Default is STR
- signature – signature of RSA or EC signing for verification in Hex.
- transformation – provide one of the RSA/EC signing transformations supported by CADP for Java.
- format – optional, supported signing CMS formats are: cms/detached/der/enveloped, cms/detached/der, cms/detached/smime/enveloped and cms/detached/smime

- The transformations: SHA1withRSAPSSPadding, SHA256withRSAPSSPadding, SHA384withRSAPSSPadding, SHA512withRSAPSSPadding, ECDSA, SHA1withECDSA, SHA256withECDSA, SHA384withECDSA, SHA512withECDSA, RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512 do not support the CMS formats.
- For KeySecure 8.12.5 onward, following transformations are added to sign/verify the data based on pre-calculated hash: RSA, RSAPSSPaddingSHA1, RSAPSSPaddingSHA256, RSAPSSPaddingSHA384, and RSAPSSPaddingSHA512. These transformations support saltlength parameter.

- caname – optional, name of the CA.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
"SignVerify": {
"username": "admin",

"password": "admin",

"keyname": "ecc_testkey",

"messagetext": "126352152154",

"signature":
"10001030818402403 59CBBCDDECC68E423F56B10D151F5F3E6B2C15DE3F250864AAC26A5B8564BDFDC2
22E6172C476D91418959D4889A7DF03C3F5A6423B105D50E76C2061AC9FF602407F8EF0AC0D34C9B71B3
```

```
F13B58C32A7CF1440607C26FE03F99991FD330BA3B31FB1A1F5ECD602BBA890B073935B4414F5EE242E9
DD8299FADE146210916F2588D",
"transformation": "SHA512withECDSA"
}
}

{
    "SignVerify_Response": {
        "signVerifyResponse": true
  }
}
```

Sample SOAP Parameters:

```
<prot:_SignVerify>
      <username>cryptouser</username>
      <password>qwerty1234</password>
      <keyname>certpkcs12</keyname>
      <messagetext>13123sdssdadasddsdsada</messagetext>
      <signature>3082087B06092A864886F70D010703A082086C3082086802010031820 1
      C0308201BC0201003081A330819B310B300906035504061302553310B30090603550
      40813024341311530130603550407130C526564776F6F642043697479311430120603
      55040A130B536166656E657420496E6331143012060355040B130B456E67696E65657
      2696E67311230100603550403140973616D706C655F6361312830260 6092A864886F7
      0D010901161973616D706C655F636140736166656E65742D696E632E636F6D020300A
      6E9300D06092A86488[… sample truncated for brevity … ]
      DACAAE8733FB6A9CD52E3B673C8E0DBB4E100A7EDE8B6DA586ADFAF38DA8975616F0B
      E035CB8B6D1514F1A6F3423180F402508A2F8E89C564</signature>
      <transformation> SHA256withECDSA </transformation>
      <!--Optional:-->
      <caname>sample_ca</caname>
</prot:_SignVerify>
```

Output: boolean – result of verification

<ns1:_SignVerifyResponse xmlns:ns1="http://dsws.org/protectappws/">true</ns1:_SignVerifyResponse>

RemoveUsersFromGroup: Remove User(s) from a Group

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/groups/deleteUsers

Input Parameters:

- adminUser
- adminPassword
- groupName – group from which user is to be removed.
- UserList – list of the users to be removed.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
  "GroupUpdateRequest": {
    "adminUser": "cryptouser",
    "adminPassword": "asdf1234",
    "groupName": "cryptogrouptest",
    "UserList":
      {
      "user": [
        "crypto12",
             ]
      },
  }
}
```

response:

```
{
  "GroupUpdateResponse": {
    "description": "[crypto12] removed from the group."
  }
}
```

UserInfo: Get User Information

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/users/userInfo

Input Parameters:

- adminUser
- adminPassword
- userName – user whose information is required.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
"UserInfoRequest":{
   "adminUser": "testadmin",
   "adminPassword": "asdf1234",
   "userName":"twguser"
   }
}
```

Sample REST response for cxf

response:

```
{"UserInfoResponse": {
   "User": "twguser",
   "GroupList": {"Group": "twgGroup"},
   "CustomAttributeList": {"CustomAttribute": [
       {
         "Name": "twg_attritbute_1",
         "Value": "twg@encryption.com"
       },
       {
         "Name": "twg_attritbute_2",
         "Value": "twg2@encryption.com"
       }
   ]},
   "isChangePasswdPermission": true
}}
```

WrapKey: Wrap a Key

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/key/wrapKey

Input Parameters:

- userName
- password
- keyName – name of the key to export.
- keyUseForWrap – key to be used for wrapping.
- wrapFormatPadding – (optional) padding format to be used for wrapping the key. It is used for PKCS#1v2.1 and one of the following padding is used: SHA256, SHA384, and SHA512.
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf

request:

```
{
  "WrapKeyRequest": {
    "userName": "cryptouser",
    "password": "asdf1234",
    "keyName": "AESKey",
    "keyUseForWrap": "RSAKey",
    "wrapFormatPadding": "SHA256"
  }
}
```

response:

```
{
  "WrapKeyResponse": {
    "wrapKeyData":
    "36E409F7993906344FA0DC560475086F485163857ACD41752651ACDF236BDDE73F9859CBF42
    A744D27603F5869D3DBD29C97005B973517DB76761AF8915D0B13"
  }
}
```

Delete_Key: Delete Keys

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/Key_Gen

Input Parameters:

- username
- password
- keyname
- keyalgorithm – Del.
- keyisdeletable
- keyisexportable
- keyisversioned
- keytemplate
- keyistemplate
- certAlias – optional, is a client certificate alias for making SSL connections.
- certPass – optional, is the password for the provided certificate alias.

Sample SOAP Parameters:

```
<prot:Key_Gen>
   <username>cryptouser</username>
   <password>qwerty1234</password>
   <keyname>testkey</keyname>
   <keyalgorithm>Del</keyalgorithm>
   <keyisdeletable></keyisdeletable>
   <keyisexportable></keyisexportable>
   <keyisversioned></keyisversioned>
   <!--Optional:-->
   <keytemplate></keytemplate>
   <!--Optional:-->
   <keyistemplate></keyistemplate>
</prot:Key_Gen>
```

Output: boolean – indicate if key deletion was successful

<ns1:Key_GenResponse xmlns:ns1="http://dsws.org/protectappws/">true</ns1:Key_GenResponse>

ModifyCustomAttributes: Modify Attributes of a Key

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/key/modifyCustomAttributes

Input Parameters:

> username

> password

> KeyName – name of the key for which attributes are to be modified.

> Attribute – name and value of the attribute for the key. The type parameter lets you optionally specify the type for attribute. The default attribute type is String. Following are the supported attribute types:

- Date/Time
- Byte String
- Integer
- Long Integer
- Big Integer
- String
- Interval
- Enumeration
- Boolean

> certAlias – optional, is a client certificate alias for making SSL connections.

> certPassword – optional, is the password for the provided certificate alias.

Sample REST call for cxf:

request:

```
{
  "ModifyCustomAttributes": {
     "username": "test1",
     "password": "abcd1234",
     "KeyName": "test_key",
     "Attributes": {
          "Attribute": [
               {
               "name": "attr1",
               "value": "123456",
               "type" : "Integer"
               },
```

```
                {
                "name": "attr2",
                "value": " 7468616c657367656d616c746f",
                "type" : "Byte String"
                }
                     ]
                 }
                       }
}
```

response:

```
{"ModifyCustomAttributesResponse": {"result": "Attributes added successfully."}}
```

Sample SOAP Parameters:

```
      <prot:ModifyCustomAttributes>
            <username>test1</username>
            <password>abcd1234</password>
            <keyName>testaeskey</keyName>
            <!--1 or more repetitions:-->
            <Attribute>
             <name>testattr</name>
             <value>124567</value>
             <!--Optional:-->
             <type>Integer</type>
             </Attribute>
      </prot:ModifyCustomAttributes>
```

Output:

<ns2:ModifyCustomAttributesResponse xmlns:ns2="http://dsws.org/protectappws/">Attributes added successfully.</ns2:ModifyCustomAttributesResponse>

NOTE: The list of attributes that we get in response is comma separated so any attribute (name and value) having special characters (particularly **,** and **{}** and **“”** and **=**) will be displayed as it is and will make the attribute difficult to read. So, it is recommended NOT to use special characters (particularly **,** and **{}** and **“”** and **=**) in the input.

AddSecretData: Add KMIP Secret Data to Key Manager

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/kmip/addSecretData

Input Parameters:

- certAlias – is a client certificate alias for making SSL connections.
- certPass – is the password for the provided certificate alias.
- secretData – secret data to be added to the Key Manager.
- dataType – type of secret data being added to Key Manager. Valid values are `Password` and `Seed`. It is an optional parameter. If no value is provided, default value `Password` is used.
- secretDataName – name for the secret data being added to Key Manager.

Sample REST call for cxf

request:

```
{
  "AddSecretDataRequest": {
    "certAlias": "privatekey",
    "certPassword": "asdf1234",
    "secretData": "2E1717D057F2BA2CBA7F6EFF6D37F846CE2848C92C1A0BCBF4F4D3011304BD57",
    "dataType": "Password",
    "secretDataName": "secdata1"
  }
}
```

response:

```
{
  "AddSecretDataResponse": {
    "secretDataResponse":
    "4747C796C5A5BF5D19F9EB8621CF9F304DCF646E515B292250BF2B5BB9FA43B8"
  }
}
```

The unique identification number (UID) of the secret data is returned in the response

NOTE: Caching of session-pool for a user is not supported for KMIP requests.

GetSecretData: Retrieve KMIP Secret Data from Key Manager

URL: <http/https>://<host-name>:<Port>/protectappws/services/rest/kmip/getSecretData

Input Parameters:

- certAlias – is a client certificate alias for making SSL connections.
- certPass – is the password for the provided certificate alias.
- dataType – type of data being retrieved from Key Manager. Valid values are `Password` and `Seed`. It is an optional parameter.
- secretDataName – name for the secret data being retrieved from Key Manager.
- UID – unique identification number for the secret data being retrieved. UID is generated when the secret data is added to the Key Manager.

NOTE: Either of `secretDataName` or `UID` parameter is used to retrieve the secret data from the Key Manager. If both these parameters are specified, then preference is given to `secretDataName`.

Sample REST call for cxf using secretDataName

request:

```
{
  "GetSecretDataRequest": {
    "certAlias": "privatekey",
    "certPassword": "asdf1234",
    "dataType": "Password",
    "secretDataName": "secdata1"
  }
}
```

response:

```
{
  "GetSecretDataResponse": {
    "secretDataResponse":
    "2E1717D057F2BA2CBA7F6EFF6D37F846CE2848C92C1A0BCBF4F4D3011304BD57"
  }
}
```

Sample REST call for cxf using UID

request:

```
{
  "GetSecretDataRequest": {
    "certAlias": "privatekey",
    "certPassword": "asdf1234",
    "dataType": "Password",
    "UID": "09B5A600CF2B84EEE87A294750536FDCD66442122817BE"
  }
}
```

response:

```
{
  "GetSecretDataResponse": {
    "secretDataResponse":
    "2E1717D057F2BA2CBA7F6EFF6D37F846CE2848C92C1A0BCBF4F4D3011304BD57"
  }
}
```

CHAPTER 4: CADP for Java Mobile App Samples

The Mobile App sample is located in `CADP_for_JAVA\lib\webservices\AndroidApp`.

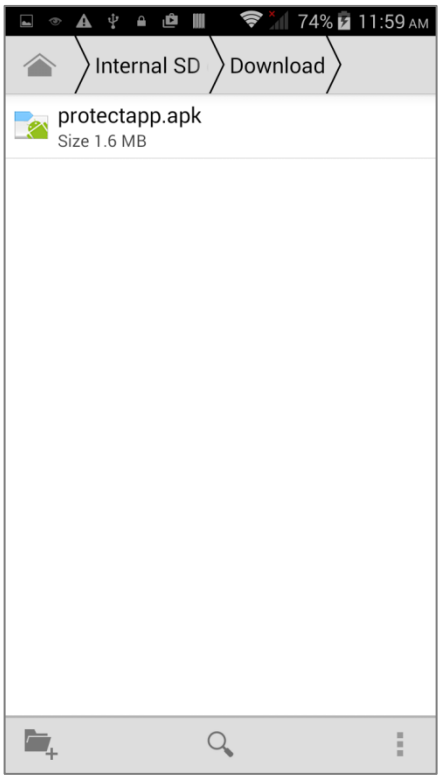
Overview

CADP for Java Mobile App is an android based application. It is integration between CADP for Java WebServices and android application.

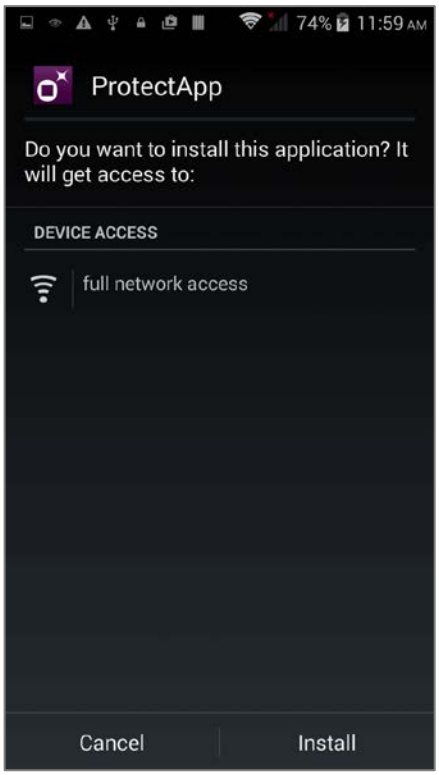
Installation

To install CADP for Java Mobile App follow the steps below:

1. Download the **protectapp.apk** file from the latest CADP for Java build and copy the apk to your android mobile handset.

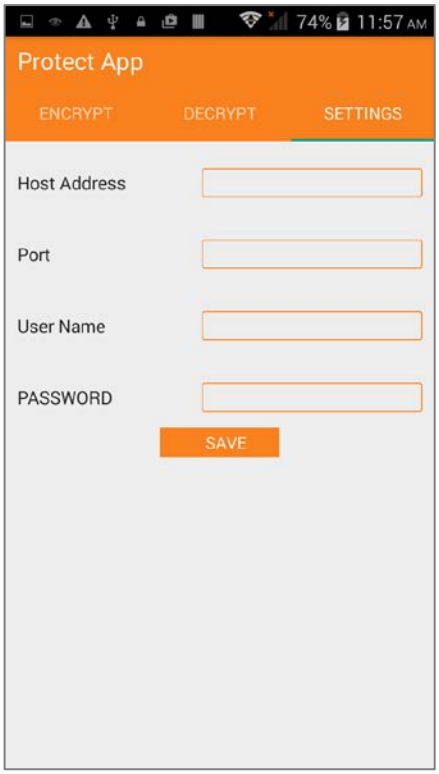


2. Run the **protectapp.apk** setup to install the CADP for Java Mobile App.



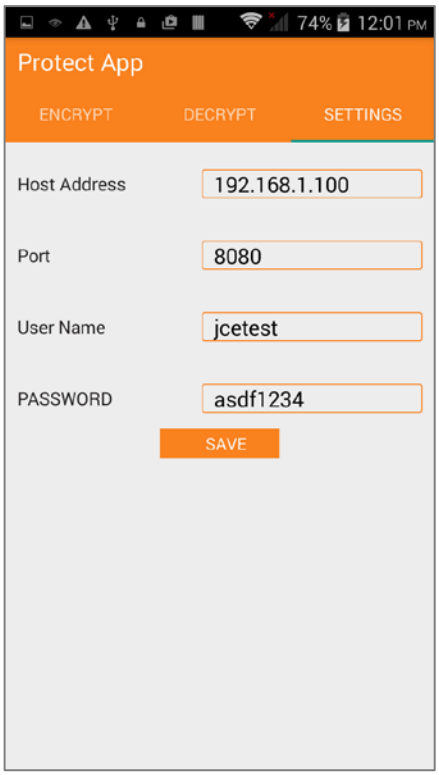
3. Click **Install** to start the installation. The following screen appears displaying installation in progress:

4. Once the installation is complete, the following screen appears:



To start using the mobile app, configure the mobile app.

5. On **SETTINGS** tab, add the following configuration:
   - **Host Address** – IP address of the server.
   - **Port** – Port of the server.
   - **User Name** – Username to connect to the server.
   - **Password** – Password to connect to the server.



6. Click **Save** to save the configuration.

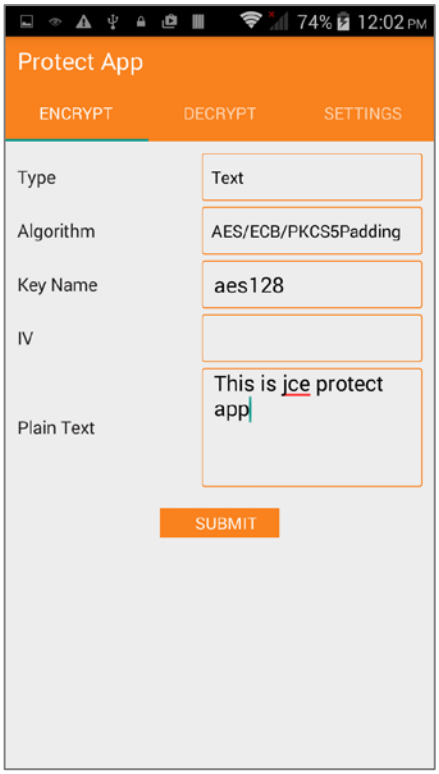
Encryption with Mobile App

To encrypt a plaintext with mobile app, follow the steps below:

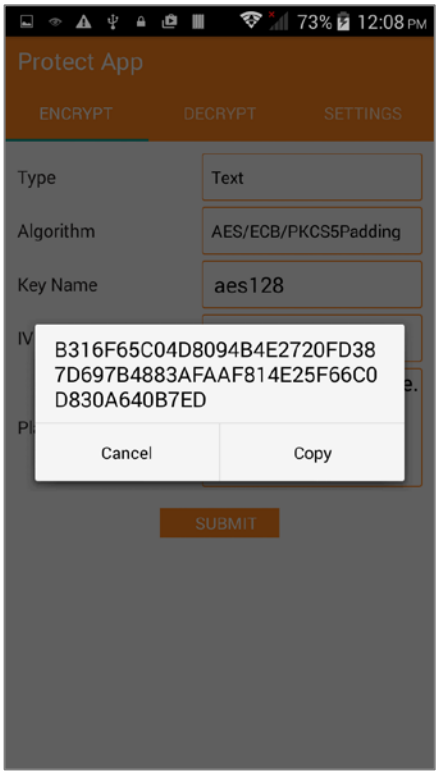
1. On the android mobile handset, run the CADP for Java application.
2. On the **ENCRYPT** tab, enter the following details:
   - **Type** – Input data type.
   - **Algorithm** – Algorithm to be used for encryption.
   - **Key Name** – Name of the key to be used for encryption. **Version with all** (encryption with all the versions of the key) is not supported.

     NOTE: Version with all: When you append #all in the key name, the system performs encryption of the plaintext with all the available versions of the key.

   - **IV** – Optional, a random data.
   - **Plain Text** – The plaintext data to be encrypted.



3. Click **Submit**. The encrypted ciphertext output is displayed in a pop-up message box.

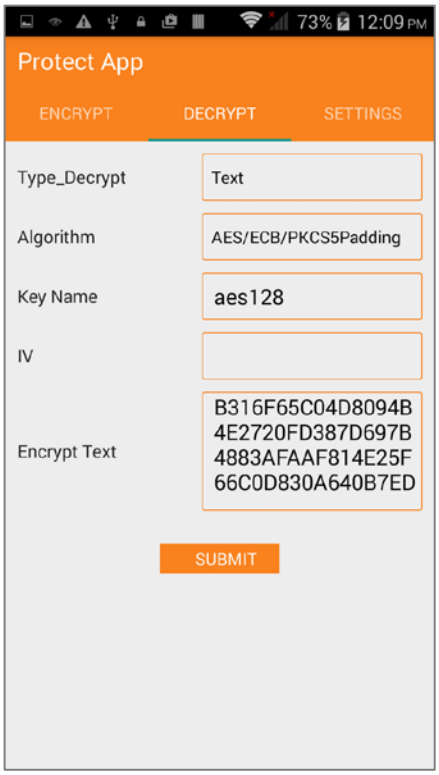


4. Click **Copy** to copy the ciphertext or click **Cancel** to end.

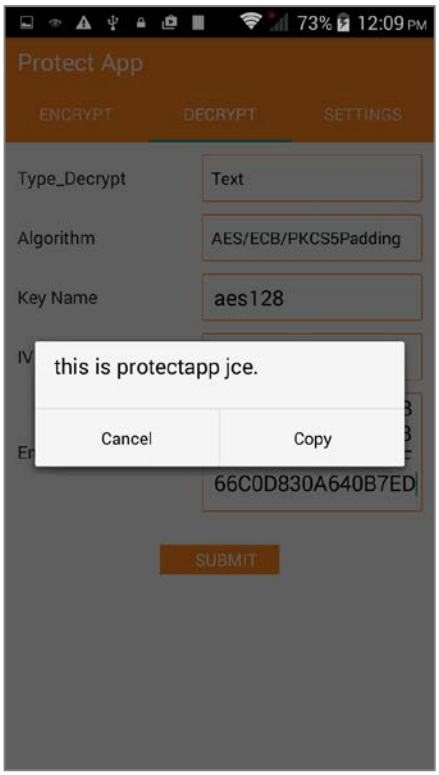
Decryption with Mobile App

To decrypt a plaintext with mobile app, follow the steps below:

1. On the android mobile handset, run the CADP for Java application.
2. On the **DECRYPT** tab, enter the following details:
   - **Type_Decrypt** – Input data type.
   - **Algorithm** – Algorithm to be used for decryption.
   - **Key Name** – Name of the key to be used for decryption.
   - **IV** – optional, a random data. It should be same that was used to encrypt the plaintext.
   - **Encrypt Text** – The ciphertext data to be decrypted.



3. Click **Submit**. The decrypted plaintext output is displayed in a pop-up message box.



4. Click **Copy** to copy the plaintext or click **Cancel** to end.

Exit Mobile App

To exit CADP for Java mobile application, press back or exit button on your android mobile handset.