Cybersource B2C Commerce - SOAP Authentication Guide

Contents

Introduction

This document provides a step-by-step guide for managing P12 certificates and code changes for MLE implementation without upgrading the cartridge. It highlights that P12 Authentication now supports both JKS and PKCS12 keystore types. Also includes the changes in configuration to make it general for both Authentication and MLE. It covers the process of generating a P12 file and converting it to the JKS format and managing it. Additionally, it outlines the essential code modifications required to ensure seamless integration and functionality.

Message-Level Encryption (MLE) enables you to store information or communicate with other parties while helping to prevent uninvolved parties from understanding the stored information. MLE is optional and supported only for payments services.

Meta key is a specialized API key that a portfolio or merchant account user can create for the purposes of processing transactions on behalf of multiple of their transacting MID accounts. Meta keys are useful for organizations whose transacting MID users do not manage or store their own individual API keys. Instead of having to create and assign a unique API key for each of your transacting MIDs, you can create and assign a single meta key to dozens or hundreds of your transacting MIDs simultaneously.

1. Merchants using ENT cartridge v21.1.0 and above

1.1.Steps for managing P12 certificate

Below are the steps to generate a P12 file and converting it to the JKS format and managing it.

Step 1: Create P12 file

Option 1: Create a p12 key.

We can use p12 key generated for a specific MID from which it is created.

1. Follow steps mentioned in the link to generate a P12 certificate in Business Center.
2. Make a note of password set to the P12 key.
3. Download the generated P12 file.

Option 2: Create Meta Key.

We can assign a single meta key to dozens or hundreds of transacting MIDs simultaneously.

1. Follow steps mentioned in the link to generate a meta key from Business Center.
2. Make a note of password set to the meta key.
3. Download the generated p12 file.

Step 2: JKS creation

To convert the p12 file to JKS follow the steps mentioned below. Open the terminal in the folder where the P12 file is stored.

1. **These commands will extract all the certs from the p12 file.**

```
openssl pkcs12 -in <Merchant_ID>.p12 -nocerts -out <Merchant_ID>.key
openssl pkcs12 -in <Merchant_ID>.p12 -clcerts -nokeys -out <Merchant_ID>.crt
openssl pkcs12 -in <Merchant_ID>.p12 -cacerts -nokeys -out CyberSourceCertAuth.crt
openssl pkcs12 -in <Merchant_ID>.p12 -cacerts -nokeys -out CyberSource_SJC_US.crt
```

2. **Create a new p12. Here Identity.p12 is the name of the new p12 file**

```
openssl pkcs12 -export -certfile CyberSourceCertAuth.crt -in <Merchant_ID>.crt -inkey
<Merchant_ID>.key -out identity.p12 -name <Merchant_ID>
```

3. **Create JKS from p12 using keytool. Here, <SrcStorePassword> is the password for Identity.p12**

```
keytool -importkeystore -destkeystore <Your_keystore_name>.jks -deststorepass <your_password>
-srckeystore identity.p12 -srcstoretype PKCS12 -srcstorepass <SrcStorePassword>
```

4. **Now import the CyberSource_SJC_US.crt to your keystore**

```
keytool -importcert -trustcacerts -file CyberSource_SJC_US.crt -alias CyberSource_SJC_US -keystore
<Your_keystore_name>.jks
```

You will be prompted “Trust this certificate? [no]:”. Type ‘yes’

5. **List the entries of your keystore**

```
keytool -list -v -keystore <Your_keystore_name>.jks
```

It should have two entries.

a. **CyberSource_SJC_US** certificate with alias name as cybersource_sjc_us. **This certificate is used for MLE.**
b. The other entry should contain a chain of two certificates - CyberSourceCertAuth and <Merchant_ID> with alias name <Merchant_ID>. **This is used for Authentication.**

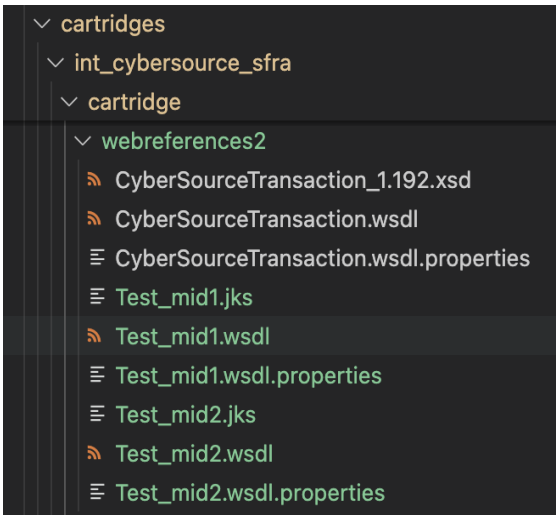
Step 3: Place the Keystore file in our cartridge

Place the file/files in the webreferences2 folder of the same cartridge as the WSDL file.

Path: cartridges\int_cybersource_sfra\cartridge\webreferences2

In case of multiple merchant Ids, duplicate the "**CyberSourceTransaction.wsdl"** file, **"CyberSourceTransaction.wsdl.properties"** file and rename them with the same name as your respective Keystore files.

Example: Test_mid1 and Test_mid2 are generated keystore JKS files added to webreferences2 folder.



NOTE:

1. It's mandatory to use JKS as the Keystore type if MLE is enabled.
2. For P12 Authentication alone, Keystore type can be either PKCS12 or JKS.

Step 4: Configurations in Business Manager

Refer section Metadata changes to create configurations to create required configurations.

Go to **Merchant Tools** > **Site Preferences** > **Custom Preferences** > **Cybersource** and set values for the following parameters

Field	Description
Cybersource Merchant ID	Cybersource Merchant ID (mention the merchant ID not the account or portfolio id)
CsKeystore_Name	Name of the keystore file added in webreferences2 folder.
CsAuth_Alias	If MLE is enabled, use the Alias of the client certificate in JKS file for Authentication (<Merchant_ID>). If MLE is disabled and you are choosing to use PKCS12 keystore for Authentication, use Friendly name from p12 file.
CsKeystore_Password	The password of the keystore file.
CsAuth_KeystoreType	Type of keystore for Authentication (PKCS12 or JKS).

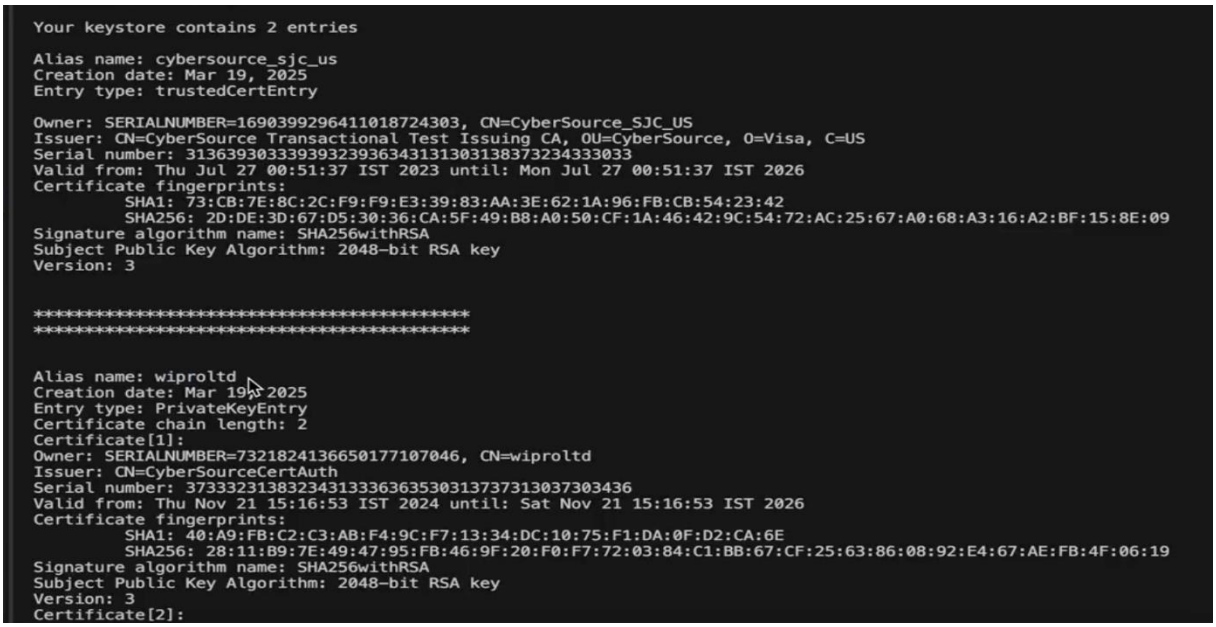
	NOTE: Use only JKS type if MLE is enabled.
CsMLE_Enabled	Enable or Disable Message-Level Encryption
CsJKS_MLEAlias	Alias of the certificate in JKS file for MLE

MLE Enabled

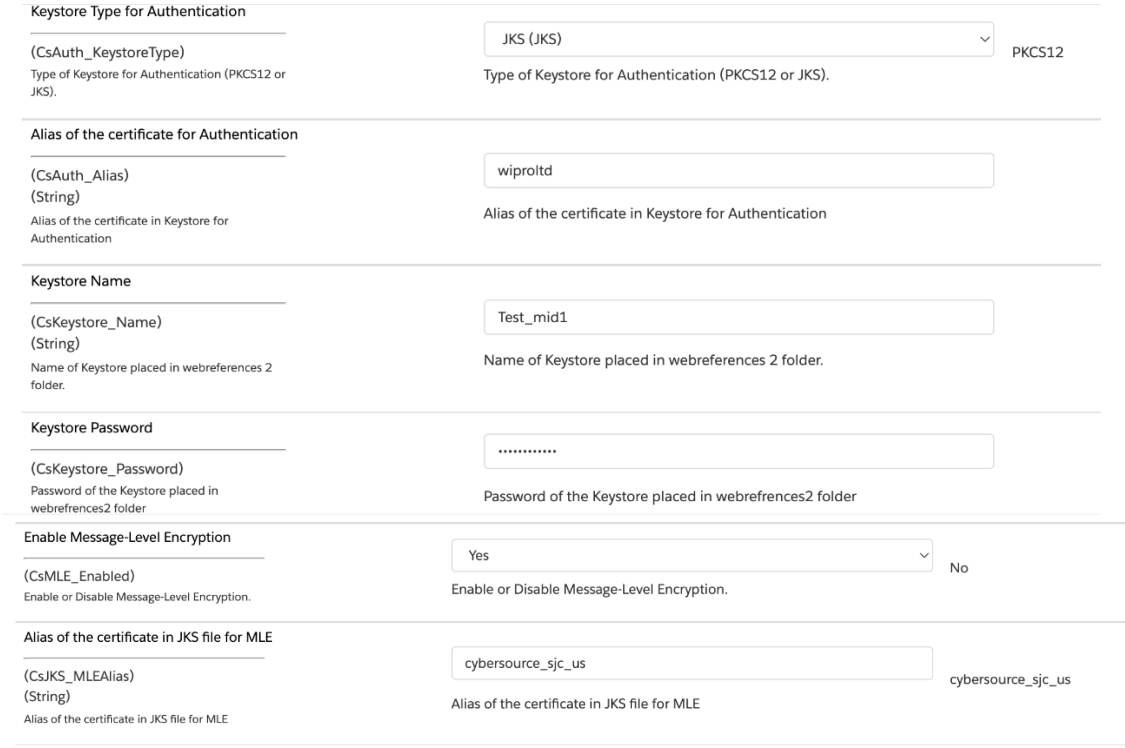
If you are opting for MLE, use the JKS Keystore. To obtain the alias for MLE and Authentication, run the following key tool command for the JKS file.

```
keytool -list -v -keystore <Your_keystore_name>.jks
```

Refer to the below example.



Make note of both the aliases. In this example **'cybersource_sjc_us'** will be used in "**Alias of the certificate in JKS file for MLE"** and **'wiproltd'** will be used in **"Alias of the certificate for Authentication"** configuration in Business Manager.



MLE Disabled and using PKCS12 keystore type for Authentication

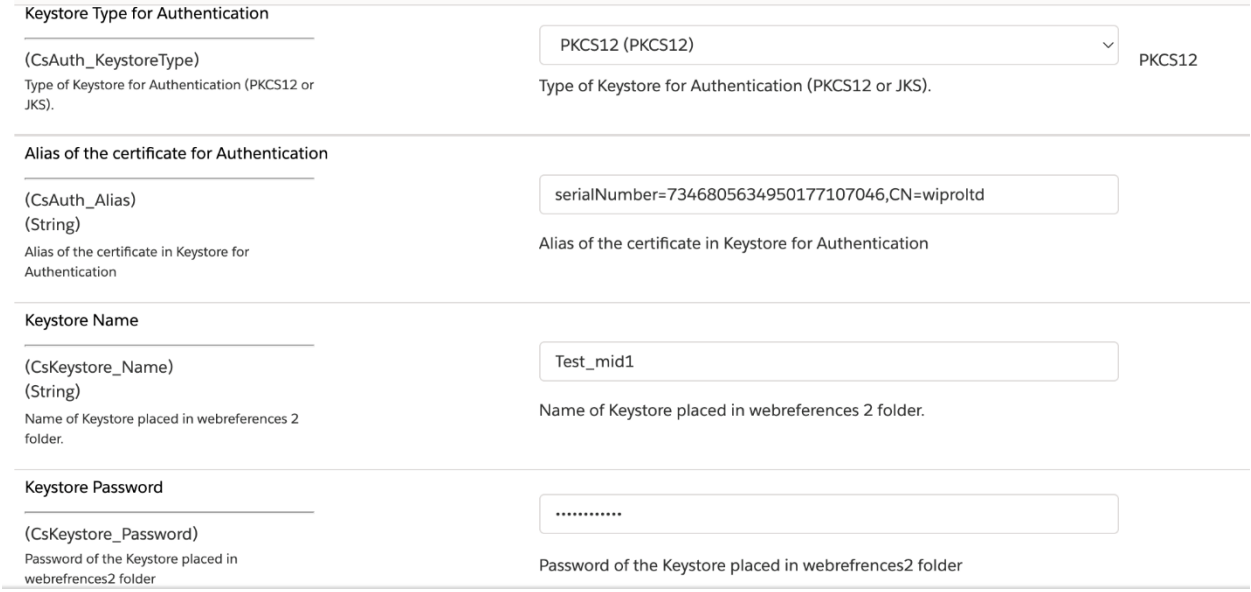
If you choose not to use the MLE feature and prefer to directly use the PKCS12 keystore type for Authentication, obtain the friendly name of the client certificate using the OpenSSL command.

```
openssl pkcs12 -info -in <Keystore_name>.p12
```

Note the friendly name of the certificate and use it in “Alias of the certificate for Authentication” configuration in Business Manager.

Refer the below example.

```
Certificate bag
Bag Attributes
    localKeyID: 01
    friendlyName: serialNumber=734680563495017710 7046,CN=wiproltd
subject=/CN=wiproltd/serialNumber=73468056349501771 07046
issuer=/CN=CyberSourceCertAuth
-----BEGIN CERTIFICATE-----
MIICXzCCAcigAwIBAgIWNzM0NjgwNTYzNDk1MDE3NzEwNzA0NjANBgkqhkiG9w0B
AQsFADAeMRwwGgYDVQQDDBNDeWJlclNvdXJjZUNlcnRBdXRoMB4XDTI0MTIyMDA3
NDI0M1oXDTI2MTIyMDA3NDI0M1owNDERMA8GA1UEAwwId2lwcm9sdGQxHzAdBgNV
BAUTFjczNDY4MDU2MzQ5NTAxNzcxMDcwNDYwggEiMA0GCSqGSIb3DQEBAQUAA4IB
DwAwggEKAoIBAQCLqlODeZrf7VJspXKVYPQmolkzfQS0LfcmaZabxBfAHzALgCqX
+VZ5nNPEcyZWZBi4p35Chfz4V5PAjw8zEFNa5XyTGXYhacMsFCtMEPkV7H0vxB7a
90JlP43z3/TNHQWxTtiqLQ/sQyN5duAnYuI409ak/r3igNPQ7eVvI9xs8rOJ5dum
xpfQoVG8atIunoXc+X6jDgLWi8e1XdeHaAEs4xRk3APor1G0SpmesWSdAuYrKm3J
LbsccGWyAnrKLAPJS+qF1KRuSGgvHxZtl9Jh3Cp1iOuhUbAqvYKusv8LV6Xte+Ym
```



1.2.Code changes for MLE

Step 1: Changes to read the newly added configurations from Business Manager

Make the following changes to **libCybersource.j**s file.

Path: "cartridges/int_cybersource_sfra/cartridge/scripts/cybersource/libCybersource.js"

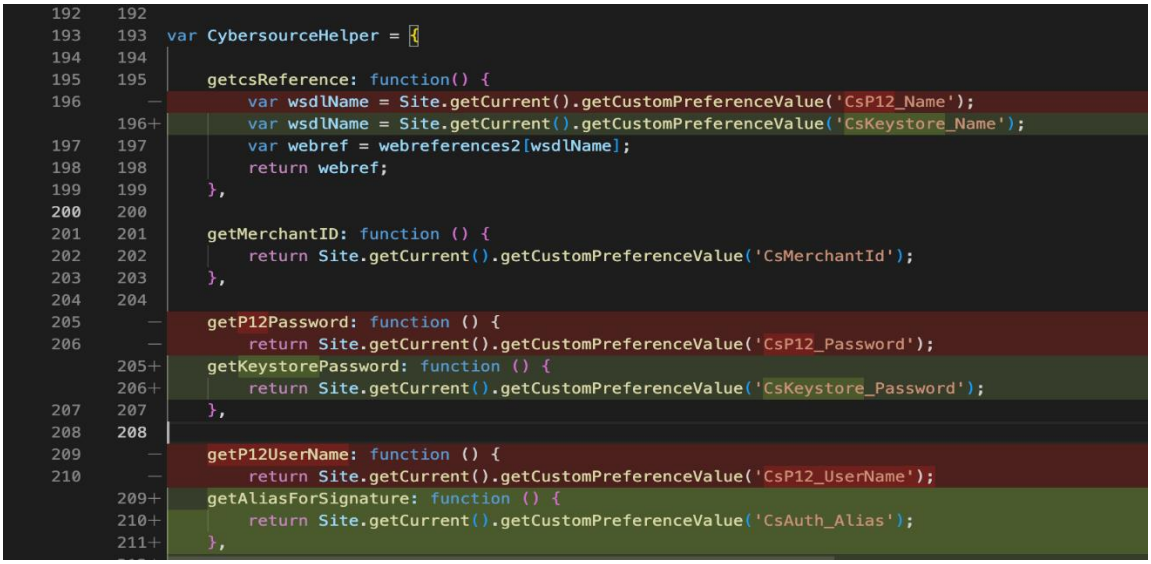
1. Add the following functions to read the new configurations in "**CybersourceHelper**" object.

```
isMLEEnabled: function () {
    return Site.getCurrent().getCustomPreferenceValue('CsMLE_Enabled');
},

getAliasForMLEinJKSfile: function () {
    return Site.getCurrent().getCustomPreferenceValue('CsJKS_MLEAlias');
},

getKeystoreTypeforAuthentication: function () {
    return Site.getCurrent().getCustomPreferenceValue('CsAuth_KeystoreType');
},
```

2. Add the following changes to existing configurations for Authentication.

```
var wsdlName = Site.getCurrent().getCustomPreferenceValue('CsKeystore_Name');
getKeystorePassword: function () {
    return Site.getCurrent().getCustomPreferenceValue('CsKeystore_Password');
},

getAliasForSignature: function () {
    return Site.getCurrent().getCustomPreferenceValue('CsAuth_Alias');
},
```

Refer the following screenshot:



```
192  192
193  193  var CybersourceHelper = {
194  194
195  195      getcsReference: function() {
196       -       var wsdlName = Site.getCurrent().getCustomPreferenceValue('CsP12_Name');
         196+       var wsdlName = Site.getCurrent().getCustomPreferenceValue('CsKeystore_Name');
197  197          var webref = webreferences2[wsdlName];
198  198          return webref;
199  199      },
200  200
201  201      getMerchantID: function () {
202  202          return Site.getCurrent().getCustomPreferenceValue('CsMerchantId');
203  203      },
204  204
205       -      getP12Password: function () {
206       -          return Site.getCurrent().getCustomPreferenceValue('CsP12_Password');
         205+      getKeystorePassword: function () {
         206+          return Site.getCurrent().getCustomPreferenceValue('CsKeystore_Password');
207  207      },
208  208
209       -      getP12UserName: function () {
210       -          return Site.getCurrent().getCustomPreferenceValue('CsP12_UserName');
         209+      getAliasForSignature: function () {
         210+          return Site.getCurrent().getCustomPreferenceValue('CsAuth_Alias');
         211+      },
```

Step 2: Changes in SoapServiceInit.js file

Path: "**cartridges/int_cybersource_sfra/cartridge/scripts/init/SoapServiceInit.js**"

Replace the existing **execute()** method with the following method in "**CyberSourceTransactionService**" service.

```
execute: function (svc, parameter) {
    var WSU_NS = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd";
    var libCybersource =
require('*/cartridge/scripts/cybersource/libCybersource');
    var CybersourceHelper = libCybersource.getCybersourceHelper();
    var passwordOfKeystore = CybersourceHelper.getKeystorePassword();
    var alisForSignature = CybersourceHelper.getAliasForSignature();

    var aliasForEncryption = CybersourceHelper.getAliasForMLEinJKSfile();
    var isMLEEnabled = CybersourceHelper.isMLEEnabled();
    var keystoreTypeforAuthentication =
CybersourceHelper.getKeystoreTypeforAuthentication();

    var secretsMap = new HashMap();
    secretsMap.put(alisForSignature, passwordOfKeystore);

    //request-config
    var requestCfg = new HashMap();

    if (isMLEEnabled) {
        requestCfg.put(WSUtil.WS_ACTION, WSUtil.WS_TIMESTAMP + " " +
WSUtil.WS_SIGNATURE + " " + WSUtil.WS_ENCRYPT);
        // define enrcryption properties
```

```
        requestCfg.put(WSUtil.WS_ENCRYPTION_USER, aliasForEncryption);
        requestCfg.put(WSUtil.WS_ENC_PROP_KEYSTORE_TYPE, "jks");
        requestCfg.put(WSUtil.WS_ENC_PROP_KEYSTORE_PW, passwordOfKeystore);
        requestCfg.put(WSUtil.WS_ENC_PROP_KEYSTORE_ALIAS,
aliasForEncryption);
        requestCfg.put(WSUtil.WS_ENC_KEY_ID,
WSUtil.KEY_ID_TYPE_X509_KEY_IDENTIFIER);


        requestCfg.put(
            WSUtil.WS_ENCRYPTION_PARTS,
            "{Element}{" +
            WSU_NS +
            "}Timestamp;" +
            "{Content}{http://schemas.xmlsoap.org/soap/envelope/}Body",
        );
    }
    else {
        requestCfg.put(WSUtil.WS_ACTION, WSUtil.WS_TIMESTAMP + " " +
WSUtil.WS_SIGNATURE);
    }
    requestCfg.put(WSUtil.WS_SIGNATURE_USER, alisForSignature);
    requestCfg.put(WSUtil.WS_PASSWORD_TYPE, WSUtil.WS_PW_TEXT);
    requestCfg.put(WSUtil.WS_SIG_DIGEST_ALGO,
"http://www.w3.org/2001/04/xmlenc#sha256");

    // define signature properties
    // the keystore file has the basename of the WSDL file and the
    // file extension based on the keystore type (for example,
HelloWorld.pkcs12).
    // The keystore file has to be placed beside the WSDL file.
    requestCfg.put(WSUtil.WS_SIG_PROP_KEYSTORE_TYPE,
keystoreTypeforAuthentication.value.toLowerCase());
    requestCfg.put(WSUtil.WS_SIG_PROP_KEYSTORE_PW, passwordOfKeystore);
    requestCfg.put(WSUtil.WS_SIG_PROP_KEYSTORE_ALIAS, alisForSignature);
    requestCfg.put(WSUtil.WS_SIGNATURE_PARTS,
"{Element}{http://schemas.xmlsoap.org/soap/envelope/}Body");
    requestCfg.put(WSUtil.WS_SIG_KEY_ID,
WSUtil.KEY_ID_TYPE_DIRECT_REFERENCE);


    requestCfg.put(WSUtil.WS_SECRETS_MAP, secretsMap);

    //response-config
    var responseCfg = new HashMap();
```

```
    responseCfg.put(WSUtil.WS_ACTION, WSUtil.WS_TIMESTAMP);

    WSUtil.setWSSecurityConfig(svc.serviceClient, requestCfg, responseCfg);
// Setting WS security
    return svc.serviceClient.runTransaction(parameter.request);
},
```

Step 3: Metadata changes to create configurations

1. Make changes to "**metadata/sfra_meta/meta/Cybersource.xml**" file:

Add these new configurations code in **<custom-attribute-definitions>** element of the xml.

```
<attribute-definition attribute-id="CsAuth_KeystoreType">
    <display-name xml:lang="x-default">Keystore Type for
Authentication</display-name>
    <description xml:lang="x-default">Type of Keystore for Authentication
(PKCS12 or JKS).</description>
    <type>enum-of-string</type>
    <mandatory-flag>false</mandatory-flag>
    <externally-managed-flag>false</externally-managed-flag>
    <value-definitions>
        <value-definition default="true">
            <display xml:lang="x-default">PKCS12</display>
            <value>PKCS12</value>
        </value-definition>
        <value-definition>
            <display xml:lang="x-default">JKS</display>
            <value>JKS</value>
        </value-definition>
    </value-definitions>
</attribute-definition>
<attribute-definition attribute-id="CsMLE_Enabled">
    <display-name xml:lang="x-default">Enable Message-Level
Encryption</display-name>
    <description xml:lang="x-default">Enable or Disable Message-Level
Encryption</description>
    <type>boolean</type>
    <mandatory-flag>false</mandatory-flag>
    <externally-managed-flag>false</externally-managed-flag>
    <default-value>false</default-value>
</attribute-definition>
<attribute-definition attribute-id="CsJKS_MLEAlias">
    <display-name xml:lang="x-default">Alias of the certificate in JKS file
for MLE</display-name>
    <description xml:lang="x-default">Alias of the certificate in JKS file
for MLE</description>
    <type>string</type>
```

```
    <mandatory-flag>false</mandatory-flag>
    <externally-managed-flag>false</externally-managed-flag>
    <min-length>0</min-length>
    <default-value>cybersource_sjc_us</default-value>
</attribute-definition>
```

Replace the below existing configurations with the following config code.

Configs to be replaced (Old)	CsP12_UserName	CsP12_Password	CsP12_Name
Replaced with (New)	CsAuth_Alias	CsKeystore_Password	CsKeystore_Name

If you do not have the old configurations (v24.1.3), you can directly add the new ones with the code below.

```
<attribute-definition attribute-id="CsKeystore_Password">
    <display-name xml:lang="x-default">Keystore Password</display-name>
    <description xml:lang="x-default">Password of the Keystore placed in
webrefrences2 folder</description>
    <type>password</type>
    <mandatory-flag>false</mandatory-flag>
    <externally-managed-flag>false</externally-managed-flag>
</attribute-definition>
<attribute-definition attribute-id="CsAuth_Alias">
    <display-name xml:lang="x-default">Alias of the certificate for
Authentication</display-name>
    <description xml:lang="x-default">Alias of the certificate in Keystore
for Authentication</description>
    <type>string</type>
    <mandatory-flag>false</mandatory-flag>
    <externally-managed-flag>false</externally-managed-flag>
    <min-length>0</min-length>
</attribute-definition>
<attribute-definition attribute-id="CsKeystore_Name">
    <display-name xml:lang="x-default">Keystore Name</display-name>
    <description xml:lang="x-default">Name of Keystore placed in
webreferences 2 folder</description>
    <type>string</type>
    <mandatory-flag>false</mandatory-flag>
    <externally-managed-flag>false</externally-managed-flag>
    <min-length>0</min-length>
</attribute-definition>
```

Replace the following **<group-definitions>** with the existing one.

```
<group-definitions>
    <attribute-group group-id="CyberSource">
        <display-name xml:lang="x-default">CyberSource: Core</display-name>
```

```
        <attribute attribute-id="IsCartridgeEnabled" />
        <attribute attribute-id="CsMerchantId" />
        <attribute attribute-id="CsEndpoint" />
        <attribute attribute-id="CsDeveloperID" />
        <attribute attribute-id="CsDebugCybersource" />
        <attribute attribute-id="csMasterCardAuthIndicator" />
        <attribute attribute-id="CsAuth_KeystoreType" />
        <attribute attribute-id="CsAuth_Alias" />
        <attribute attribute-id="CsKeystore_Name" />
        <attribute attribute-id="CsKeystore_Password" />
        <attribute attribute-id="CsMLE_Enabled" />
        <attribute attribute-id="CsJKS_MLEAlias" />
    </attribute-group>
</group-definitions>
```

2. Add the below code in "**metadata/sfra_meta/sites/Refarch/preferences.xml**" file

```
<preference preference-id="CsAuth_KeystoreType">PKCS12</preference>
<preference preference-id="CsMLE_Enabled">false</preference>
<preference preference-id="CsJKS_MLEAlias">cybersource_sjc_us</preference>
<preference preference-
id="CsKeystore_Name">CyberSourceTransaction</preference>
<preference preference-id="CsAuth_Alias">merchantid</preference>
<preference preference-id="CsKeystore_Password"></preference>
```

2. Merchants using Site Genesis cartridge v21.1.0 and above

Merchants using Site Genesis cartridge v21.1.0 and above can follow the steps mentioned in section 1.

NOTE: When referring to any file, use the path 'cartridges/int_cybersource/' instead of 'cartridges/int_cybersource_sfra/'."s

3. Merchants using cartridge version older than v21.1.0

We strongly recommend merchants using older versions of our cartridge to upgrade to our latest cartridge version as the older version contains deprecated packages and methods which may not be compatible with our latest changes.

However, please follow the steps below to update required files to use p12 authentications and MLE.

Step 1: Update folder name from webreference to webreferences2.

Change all the references of webreference to webreferences2 in our cartridge.

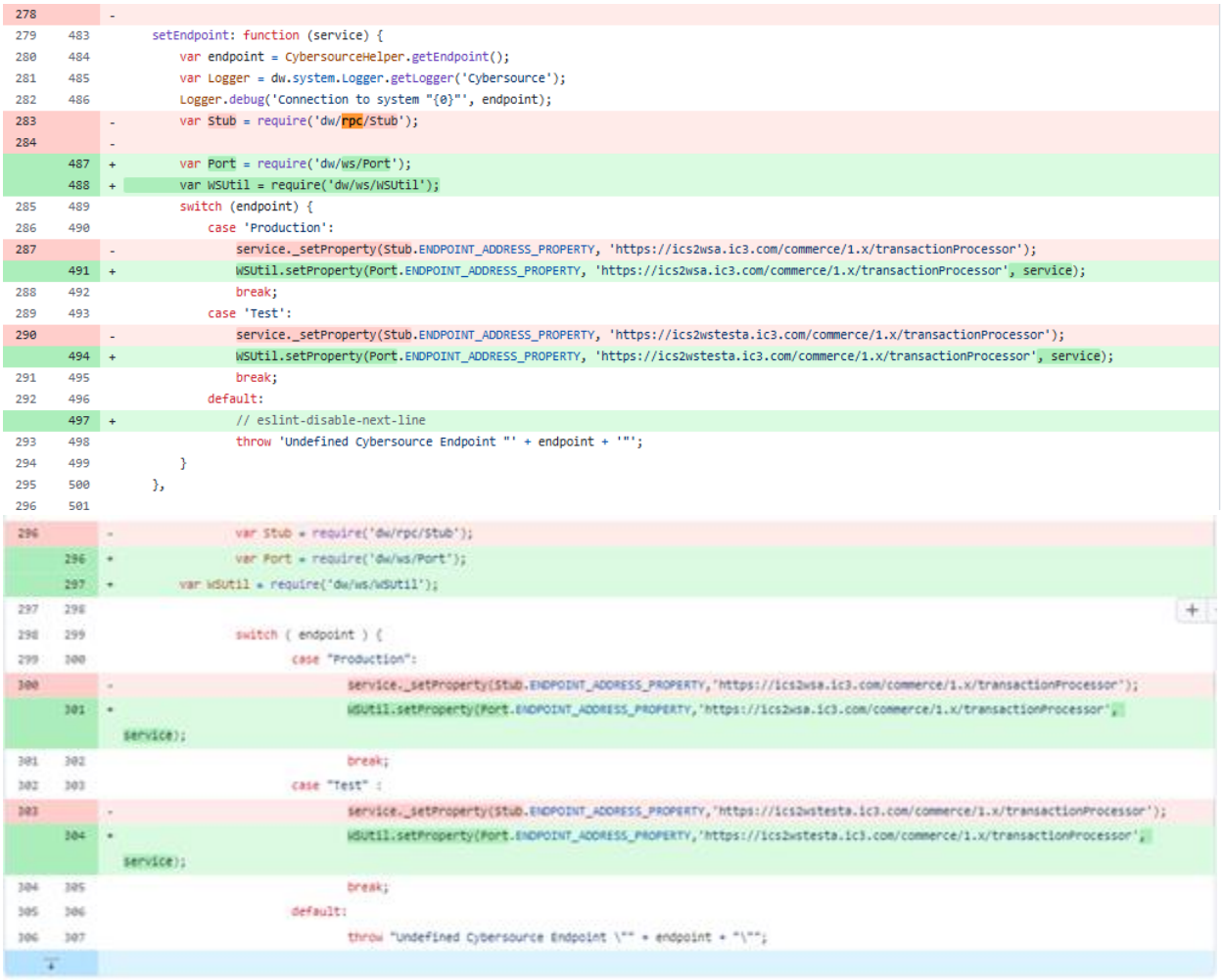
Step 2: Add below changes to SoapServiceInit.js

The SOAPUtil class is deprecated. Replace it with WSUtil. Replace the execute() function of “CyberSourceTransactionService” with below code snippet.

```
execute: function (svc, parameter) {
    var WSU_NS = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd";
    var libCybersource =
require('*/cartridge/scripts/cybersource/libCybersource');
    var CybersourceHelper = libCybersource.getCybersourceHelper();
    var passwordOfKeystore = CybersourceHelper.getKeystorePassword();
    var alisForSignature = CybersourceHelper.getAliasForSignature();

    var aliasForEncryption = CybersourceHelper.getAliasForMLEinJKSfile();
    var isMLEEnabled = CybersourceHelper.isMLEEnabled();
    var keystoreTypeforAuthentication =
CybersourceHelper.getKeystoreTypeforAuthentication();

    var secretsMap = new HashMap();
    secretsMap.put(alisForSignature, passwordOfKeystore);

    //request-config
    var requestCfg = new HashMap();

    if (isMLEEnabled) {
        requestCfg.put(WSUtil.WS_ACTION, WSUtil.WS_TIMESTAMP + " " +
WSUtil.WS_SIGNATURE + " " + WSUtil.WS_ENCRYPT);
        // define enrcryption properties
        requestCfg.put(WSUtil.WS_ENCRYPTION_USER, aliasForEncryption);
        requestCfg.put(WSUtil.WS_ENC_PROP_KEYSTORE_TYPE, "jks");
        requestCfg.put(WSUtil.WS_ENC_PROP_KEYSTORE_PW, passwordOfKeystore);
        requestCfg.put(WSUtil.WS_ENC_PROP_KEYSTORE_ALIAS, aliasForEncryption);
        requestCfg.put(WSUtil.WS_ENC_KEY_ID,
WSUtil.KEY_ID_TYPE_X509_KEY_IDENTIFIER);
```

```
        requestCfg.put(
            WSUtil.WS_ENCRYPTION_PARTS,
            "{Element}{" +
            WSU_NS +
            "}Timestamp;" +
            "{Content}{http://schemas.xmlsoap.org/soap/envelope/}Body",
        );
    }
    else {
        requestCfg.put(WSUtil.WS_ACTION, WSUtil.WS_TIMESTAMP + " " +
WSUtil.WS_SIGNATURE);
    }
    requestCfg.put(WSUtil.WS_SIGNATURE_USER, alisForSignature);
    requestCfg.put(WSUtil.WS_PASSWORD_TYPE, WSUtil.WS_PW_TEXT);
    requestCfg.put(WSUtil.WS_SIG_DIGEST_ALGO,
"http://www.w3.org/2001/04/xmlenc#sha256");

    // define signature properties
    // the keystore file has the basename of the WSDL file and the
    // file extension based on the keystore type (for example,
HelloWorld.pkcs12).
    // The keystore file has to be placed beside the WSDL file.
    requestCfg.put(WSUtil.WS_SIG_PROP_KEYSTORE_TYPE,
keystoreTypeforAuthentication.value.toLowerCase());
    requestCfg.put(WSUtil.WS_SIG_PROP_KEYSTORE_PW, passwordOfKeystore);
    requestCfg.put(WSUtil.WS_SIG_PROP_KEYSTORE_ALIAS, alisForSignature);
    requestCfg.put(WSUtil.WS_SIGNATURE_PARTS,
"{Element}{http://schemas.xmlsoap.org/soap/envelope/}Body");
    requestCfg.put(WSUtil.WS_SIG_KEY_ID, WSUtil.KEY_ID_TYPE_DIRECT_REFERENCE);


    requestCfg.put(WSUtil.WS_SECRETS_MAP, secretsMap);

    //response-config
    var responseCfg = new HashMap();
    responseCfg.put(WSUtil.WS_ACTION, WSUtil.WS_TIMESTAMP);

    WSUtil.setWSSecurityConfig(svc.serviceClient, requestCfg, responseCfg); //
Setting WS security
    return svc.serviceClient.runTransaction(parameter.request);
},
```

Step 3: Please refer to below screenshots and make changes in libCybersource.js



```
-
    setEndpoint: function (service) {
        var endpoint = CybersourceHelper.getEndpoint();
        var Logger = dw.system.Logger.getLogger('Cybersource');
        Logger.debug('Connection to system "{0}"', endpoint);
-       var Stub = require('dw/rpc/Stub');
-
+       var Port = require('dw/ws/Port');
+       var WSUtil = require('dw/ws/WSUtil');
        switch (endpoint) {
            case 'Production':
-               service._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, 'https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor');
+               WSUtil.setProperty(Port.ENDPOINT_ADDRESS_PROPERTY, 'https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor', service);
                break;
            case 'Test':
-               service._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, 'https://ics2wstesta.ic3.com/commerce/1.x/transactionProcessor');
+               WSUtil.setProperty(Port.ENDPOINT_ADDRESS_PROPERTY, 'https://ics2wstesta.ic3.com/commerce/1.x/transactionProcessor', service);
                break;
            default:
+               // eslint-disable-next-line
                throw 'Undefined Cybersource Endpoint "' + endpoint + '"';
        }
    },
```

```
-               var Stub = require('dw/rpc/Stub');
+               var Port = require('dw/ws/Port');
+       var WSUtil = require('dw/ws/WSUtil');

                switch ( endpoint ) {
                        case "Production":
-                               service._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY,'https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor');
+                               WSUtil.setProperty(Port.ENDPOINT_ADDRESS_PROPERTY,'https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor',
service);
                                break;
                        case "Test" :
-                               service._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY,'https://ics2wstesta.ic3.com/commerce/1.x/transactionProcessor');
+                               WSUtil.setProperty(Port.ENDPOINT_ADDRESS_PROPERTY,'https://ics2wstesta.ic3.com/commerce/1.x/transactionProcessor',
service);
                                break;
                        default:
                                throw "Undefined Cybersource Endpoint \"" + endpoint + "\"";
```

Step 4: Post completing the above changes please make the changes by referring to section 1.

NOTE: webreference has been updated to **webreferences2** in later versions of our cartridge. So, changes added to replace **webreferences2** in section 1 to be considered as **webreferences** in older versions.