CDSL VENTURES LIMITED
...exploring new horizons

CVL ESign 2.1

ECS eSign API Integration Specification

Version 2.1
January 8, 2022

Version History

Version	Prepared by	Reviewed by	Date	Remarks
1.0	Prabu R	Vinoth Kana	June 04,2017	Initial Release
1.0.20170116	Prabu R			Updated support for Multipage Signature Image
1.1.20170129	Prabu R		Jan 29, 2017	Support for Password protected PDF
1.1.20170309	Prabu R		March 09,2017	Support for signing using existing KycRes
2.0.201709278	Prabu R		Sep 27, 2017	Support for eSign 2.0 API
2.1.20171003	Prabu R		Oct 03, 2017	Added support for Multiple appearance of same signature
2.1.20180216	Prabu R		Feb 16, 2018	Removed Pre-Kyc and added support for Kyc performed by ESP
2.1.20180223	Prabu R		Feb 23, 2018	UID (eg., Aadhaar Number) is optional and not Mandatory. Added signature verification of response data
2.1.20210822	Manoj R	Prabu R	October 22,2021	Updated Platform to eSign 2.1 UID should not be sent as input
2.1.20220108	Muthu Kumar	Manoj R	January 08, 2022	Added Java and .net supporting library details

eSign facilitates digitally signing a document by an Aadhaar holder using an Online Service. eSign is designed for applying Digital Signature using authentication of signer through Aadhaar e-KYC service. This is an integrated service which facilitates issuing a Digital Signature Certificate and performing signing of requested data by authenticating the Aadhaar holder. Aadhaar ID is mandatory for availing the eSign Service.

Benefits of eSign

- **Easy and secure way to digitally sign information anywhere, anytime –** eSign is an online service that offers application service providers the functionality to authenticate signers and perform the digital signing of documents using Aadhaar e-KYC service. Hardware tokens are not required to be used.

- **Facilitates legally valid signatures –** eSign process involves consumer consent, Digital Signature Certificate generation, Digital Signature creation & affixing and Digital Signature Certificate acceptance in accordance with the provisions of the Information Technology (IT) Act, 2000. It enforces compliance, through API specification and licensing model of APIs. Comprehensive digital audit trail - in-built to confirm the validity of transactions is also preserved.
- **Flexible and easy to implement -** eSign provides configurable authentication options in line with Aadhaar e-KYC service and also records the Aadhaar ID used to verify the identity of the signer. The authentication options for eKYC includes biometric (fingerprint or iris scan) or OTP (through the registered mobile in the Aadhaar database). eSign enables millions of Aadhaar holders easy access to legally valid Digital Signature service.
- **Respecting privacy -** eSign ensures the privacy of the signer by requiring that only the thumbprint (hash) of the document be submitted for signature function instead of the whole document.

- **Secure online service -** The eSign Service is governed by e-authentication guidelines. While authentication of the signer is carried out using Aadhaar e-KYC services, the signature on the document is carried out on a backend server of the e-Sign provider. eSign services are offered by trusted third party service provider, currently Certifying Authorities (CA) licensed under the IT Act. To enhance security and prevent misuse, Aadhaar holders private keys are created on Hardware Security Module (HSM) and destroyed

eSign Certificate Types

In the case of eSign Online Electronic Signature Service, the Digital Signature Certificates are issued in the following classes:

- **Aadhaar e-Kyc (OTP) :** This class of certificates shall be issued for individuals use based on of subscriber through Aadhaar e-KYC.

- **Aadhaar e-Kyc – Biometrics (Fingerprint / Iris) :** This class of certificate shall be issued based on biometrics authentication of subscriber through Aadhaar e-KYC service

These certificates will confirm that the information in Digital Signature certificate provided by the subscriber is same as information retained in the Aadhaar databases pertaining to the subscriber as Aadhaar holder

OTP/Biometric based eSign Process Flow

To perform eSign using OTP / Biometric using Aadhaar, the agency captures Aadhaar Number and the consent for eSigning the document from the resident and forward the browser to CVL eSign gateway service. CVL eSign Gateway service have interface with various eSign Service provider and prepares request complying with ESP and forward the client browser to eSign Service Provider.

Document Signing

eSign Service provider will display UI page in the browser window prompting user to accept the consent for performing OTP/Biometric based eKyc and retrieving resident eKyc data for eSigning the document hash.

On resident accepting consent, eSign Service Provider will perform OTP or biometric based eSign in eSign Service provider portal. In the case of OTP based eSign, the OTP will be generated by eSign Service provider via UIDAI and the resident is asked to input the OTP in the eSign Service Provider portal.

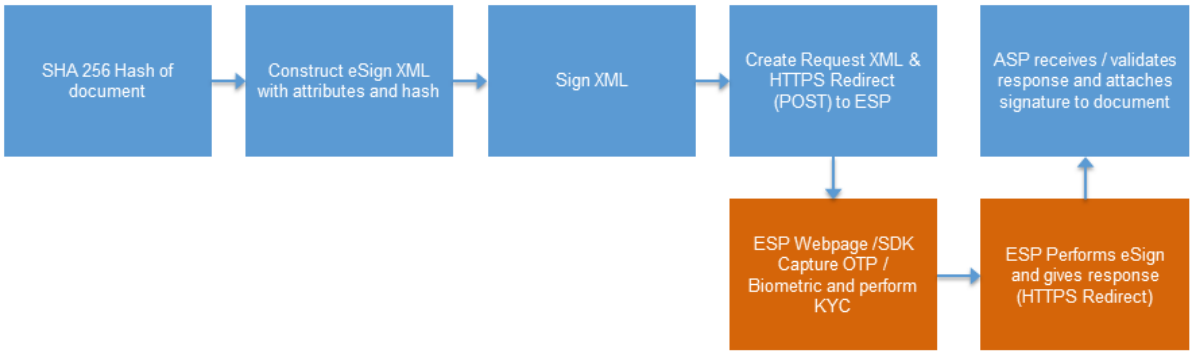
eSign Service provider interface with CIDR for performing eKyc of the resident and using the eKyc data, document hash is signed. The Signed hash along with certificate is sent back to CVL and CVL validates the response form eSign Service Provider and forward the browser back to client response URL.

In the client application, agency on receiving the response from CVL, using the CVL eSign API, embeds the signature in the PDF document to generate signed PDF document.

The signed PDF document should be delivered to the individual/company eSigning the document in the format prescribed by CCA.

The client /agency signed with CVL for eSigning the document should comply with recent CCA guidelines.

Application Flow:



- ASP client application asks eSign user to sign the document
- ASP client application creates the document hash (to be signed) on the client side
- ASP client application asks the eSign user to provide consent for certificate generation and
- signature
- ASP forms the input data for eSign API
- ASP redirect to ESP's URL or uses ESP's SDK application and submit request XML
    - ESP validates the calling application and the input.
    - ESP verifies the Digital signature of ASP for eSign XML received
    - ESP logs the transaction 5
    - ESP redirects eSign user to e- authentication page
    - ESP performs authentication and get e-KYC information from e-KYC provider
    - ESP show the document hash along with document information to eSign user.
    - ESP creates a new key pair and CSR for eSign user.
    - ESP calls the CA service and gets a Digital Signature Certificate for eSign user. The
- certificate will be a e-KYC class Digital Signature Certificate, which has e-KYC
- number, Name of the eSign user, e-KYC response code, Authentication Type, and
- Time Stamp embedded.
    - ESP signs the ‘document hash’ and provides response XML to the ASP by redirecting
- to ASP’s response URL.
- ASP receives the document signature and the eSign user’s Digital Signature Certificate.
- ASP client application attaches the signature to the document.
- eSign user can accept or reject the signature and DSC

Client Onboarding & Integration

This document covers the process involved in availing eSign services from CVL.

1) Onboarding
   The Agency interested in availing CVL eSign service should get enrolled with CVL

2) Agency shall generate RSA 2048 Digital Signature certificate and share the Public Key certificate with CVL. Organisation should keep the private Key certificate safely and if it loaded in the server, then the server access should be restricted to authorised personnel only. (CVL will provide the utility to generate the certificate key pairs, URL to download the Key generation utility and process screenshots mentioned in **KEY Generation Utility** Section)

3) CVL will create new Client ID and associate the Public Key with the generated Client ID

4) CVL will communicate the following details

   a. Client ID
   b. Pre-Production Integration URL
   c. Java / .Net API
   d. Java / .Net Sample Code

5) Agency should take consent from the eSign user for signing the specific document. Only after the consent, the document should be processed for eSigning.

6) Pre-Production integration by Client

7) Integration Testing by Client

8) Production Configuration – CVL will share the production Client ID and URL

9) Production testing

10) Go Live

ESign Service Request Response Format

API UAT URL	https://uatesign.cdslindia.com/ESign21Web/sign.jsp
API PROD URL	https://esign.cdslindia.com/ESign21Web/sign.jsp
Protocol	HTTPS
Method	Post
Content-Type	application/xml
Post data	A well-formed XML, as per the specifications provided, Sample request XML mentioned below
Input Parameters	ESIGN_REQUEST_XML, CONSENT_ID

Request Parameter Name: *ESIGN_REQUEST_XML (Send the Esign Request XML)*

Request Parameter Name: *CONSENT_ID (By default send 1)*

Sample code to post the data:

```
<FORM  name = "frm" ACTION=" https://uatesign.cdslindia.com/ESign21Web/sign.jsp"
METHOD="post">
        <input type="hidden" name="ESIGN_REQUEST_XML" value="<%=b64Request%>">
        <input type="hidden" name="CONSENT_ID" value="1">
 </FORM>
     <script language="JavaScript" type="text/javascript">
   document.frm.submit();
   </script>
```

eSign Service uses XML as the data format for input and output.

eSign Request XML structure

Following is the XML data format for eSign XML.

<Esign ver="" sc="" ts="" txn="" ekycId="" ekycIdType="" aspId=""

AuthMode="" responseSigType="" responseUrl="">

<Docs>

<InputHash id="" hashAlgorithm="" docInfo="">Document

Hash in Hex</InputHash>

</Docs>

<Signature>Digital signature of ASP</Signature>

</Esign>

S.No	Attribute	Mandatory	Value
1	Ver	Mandatory	API Version is “2.1”
2	Sc	Mandatory	Only valid value is “Y”.
3	Ts	Mandatory	Request timestamp in ISO format. The value should be in Indian Standard Time (IST), and should be within the range of maximum 30 minutes deviation to support out of sync server clocks.
4	Txn	Mandatory	Transaction ID of the ASP calling the API, this is logged and returned in the output for correlation
5	e-KYCIdType	Mandatory	This represents the type of e-KYC ID being used. The value can be any one out of below: 1. Aadhaar = A
6	ekycId	Optional	Keep this blank
7	aspId	Mandatory	Organization ID issued by ESP to the ASP
8	AuthMode	Mandatory	Authentication Mode being used for e-KYC Authentication, either to be performed by ESP, or as already made by the ASP. Allowed values are: OTP = 1 Fingerprint = 2 IRIS = 3
9	responseSigType	Mandatory	This value represents the response signature type, where ASP can request for a specific type of signature from one of the following 1. rawrsa 2. PKCS7(with only the signer certificate in the certificate section and no revocation

			information) 3. PKCS7pdf(all issuer certificates up to and including root CA certificate and CRLs/OCSP responses of each issuer certificates should be included in the response. In case, the number CRL entries are more than 5, only OCSP responses are allowed. The signature should also be time stamped using the time stamping services of CA. The revocation information should be included as a signed attribute under pdfRevocationInfoArchival(1.2.840.113583.1.1.8) 4. PKCS7complete(All issuer certificates & its revocation information in unsigned info) 5. Rawecdsa
10	responseUrl	Mandatory	This is mandatory. This should contain a valid HTTPS URL of the ASP, to which ESP has to redirect back to ASP with response XML.
11	Id	Mandatory	
12	hashAlgorithm	Mandatory	Should be fixed to “SHA256”
13	docInfo	Mandatory	Description for the respective document being signed, not more than 50 characters. docInfo should be strictly adhere to the content of document. Multiple documents of same type or different types should not be included in a single file.
14	Signature	Mandatory	Signed value of Input XML, as per the W3C recommendation on XML Signature Syntax and Processing (Second Edition)

eSign Response XML structure

<EsignResp status="" ts="" txn="" resCode="" errCode="" errMsg="">

<UserX509Certificate>base64 value of eSign user certificate

(.cer)</UserX509Certificate>

<Signatures>

<DocSignature id="" sigHashAlgorithm="SHA256" error="">

Signature data in raw (PKCS#1) or raw(ECDSA) or

PKCS7 (CMS) signature as requested

</DocSignature>

</DocSignature>

</Signatures>

<Signature>Signature of ESP</Signature>

</EsignResp>

S.No	Attribute	Mandatory	Value
1	Status	Mandatory	In case of success, it will be “1” In case of failure, it will be “0”
2	Ts	Mandatory	Will contain the response timestamp in ISO format.
3	Txn	Mandatory	The Transaction ID provided by ASP in the request.
4	resCode	Mandatory	A unique response code provided by ESP. ASP is expected to store this code in their audit log
5	errCode	Optional	In case of failure, this will contain the failure error code. In case of success, it will be “NA”
6	errMsg	Optional	In case of failure, this will contain a descriptive message against the error code. In case of success, it will be “NA”
7	Id	Mandatory	Contains the corresponding ID to the Input Hash received

8	sigHashAlgorithm	Mandatory	Should be fixed to “SHA256”
9	Error	Optional	In case of failure, this will contain the corresponding error
10	UserX509Certificate	Mandatory, if Success	Base 64 value of eSign user certificate (public)
11	Signatures	Mandatory, if Success	Sub-elements
12	DocSignature	Mandatory, if Success	Signed value in raw (PKCS#1) or raw(ECDSA) or PKCS7 (CMS) signature format as per the request XML

Error Codes:

Sl No	Error Code	Error message	Originator
1.	ESP-901	Invalid Authentication Mode	ESP
2.	ESP-902	Invalid ASP ID. It cannot be Empty	ESP
3.	ESP-903	Invalid ASP ID. It may not exist or may be inactive.	ESP
4.	ESP-905	Document Hash not received	ESP
5.	ESP-906	UID Token does not match	ESP
6.	ESP-907	Request Time Stamp cannot be Empty	ESP
7.	ESP-908	Request Time Stamp is not valid. Please check the server time.	ESP
8.	ESP-909	Transaction ID cannot be Empty	ESP
9.	ESP-910	Duplicate Transaction ID for the given ASP.	ESP
10.	ESP-911	Input XML Signature verification failed.	ESP
11.	ESP-922	Invalid Signature on Input XML. Please use the corresponding certificate mapped with ESP.	ESP
12.	ESP-944	User terminated eKYC process	ESP
13.	ESP-945	User terminated eKYC process after OTP Generation	ESP
14.	ESP-946	User interface page expired	ESP
15.	ESP-991	ESP Database Connectivity Error	ESP
16.	ESP-992	Input XML Parsing Error.	ESP
17.	ESP-993	Error Parsing CA Response XML	ESP
18.	ESP-999	Unknown Error	ESP

CVL KEY Generation Utility:

URL: https://www.dropbox.com/s/m9jpecqykxlbzy5/ecskeygen_setup_1.0.0.3.exe?dl=0

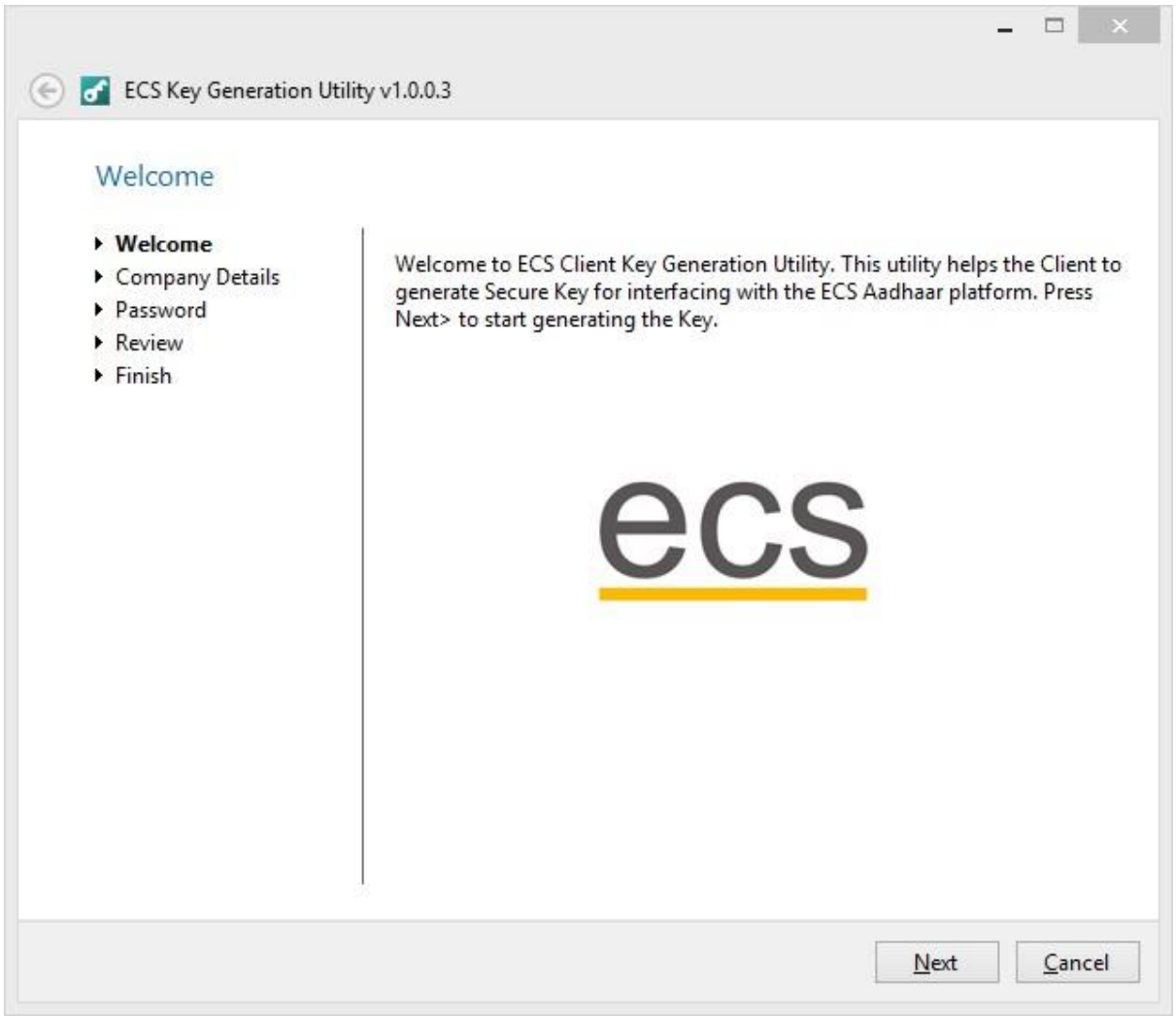
Pre-Requisite:

- Windows XP SP2 or above
- Microsoft .NET runtime 4.0

STEP 1:

Run the application ECSClientKeyGenerationUtility.exe from the installation location

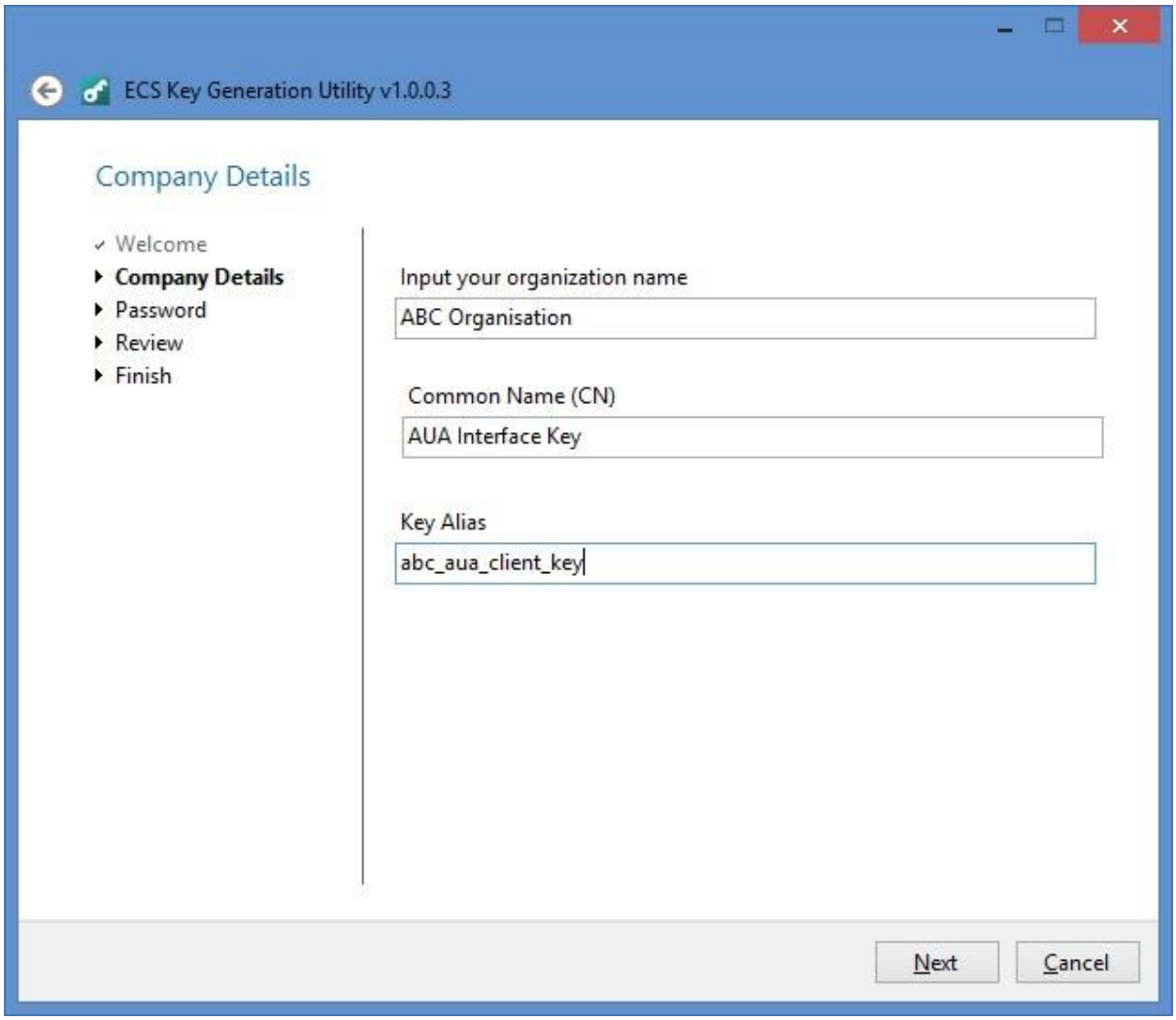
STEP 2:



Press '**Next'** to proceed

STEP 3:

Enter the organisation name in the Text Box. Avoid entering special characters such as, !,@, etc

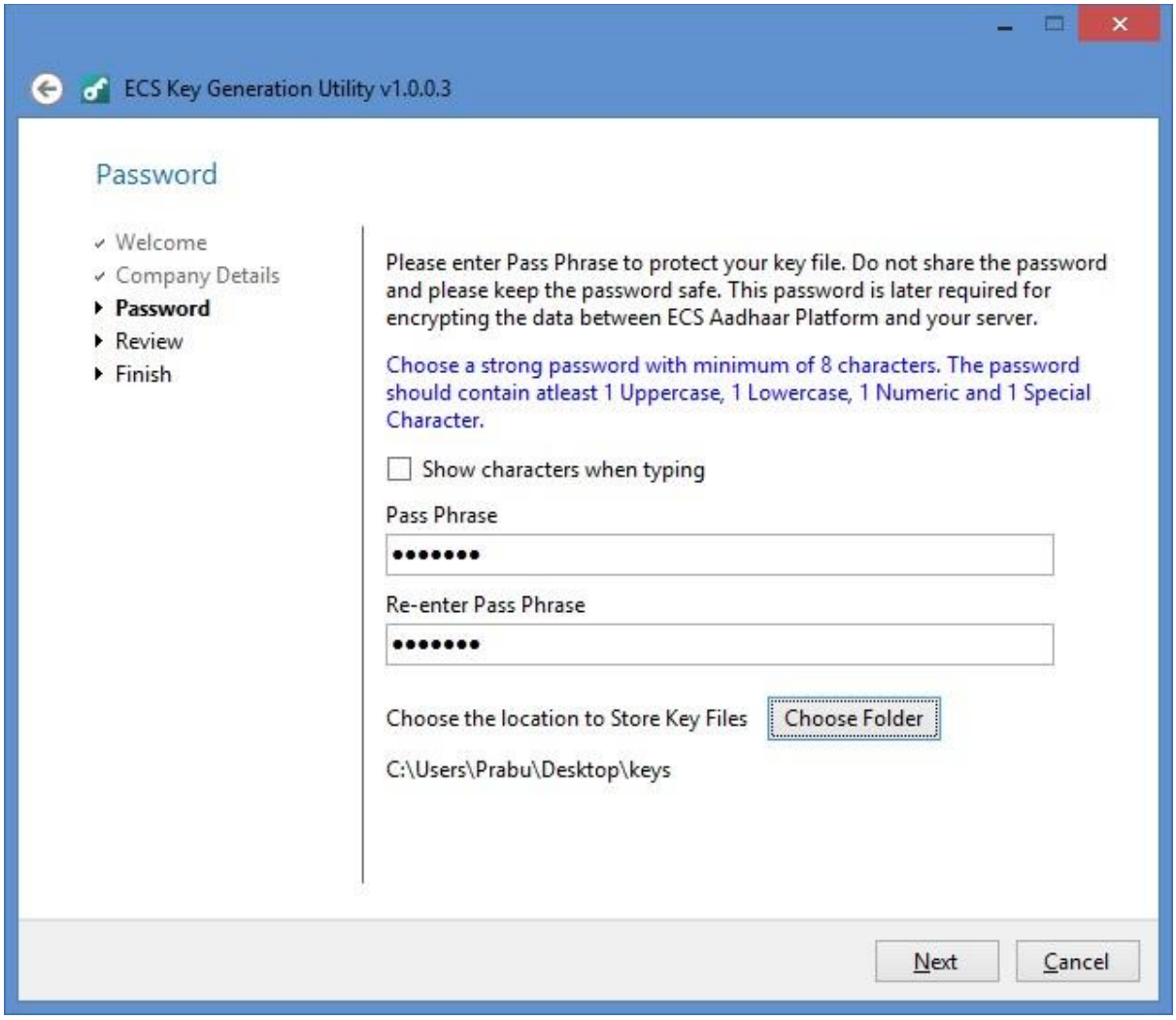


Press '**Next**' to proceed

STEP 4:

Choose the Password for the Key file. This password will be used for protecting the Private Key file. Follow the onscreen instruction for acceptable password strength.

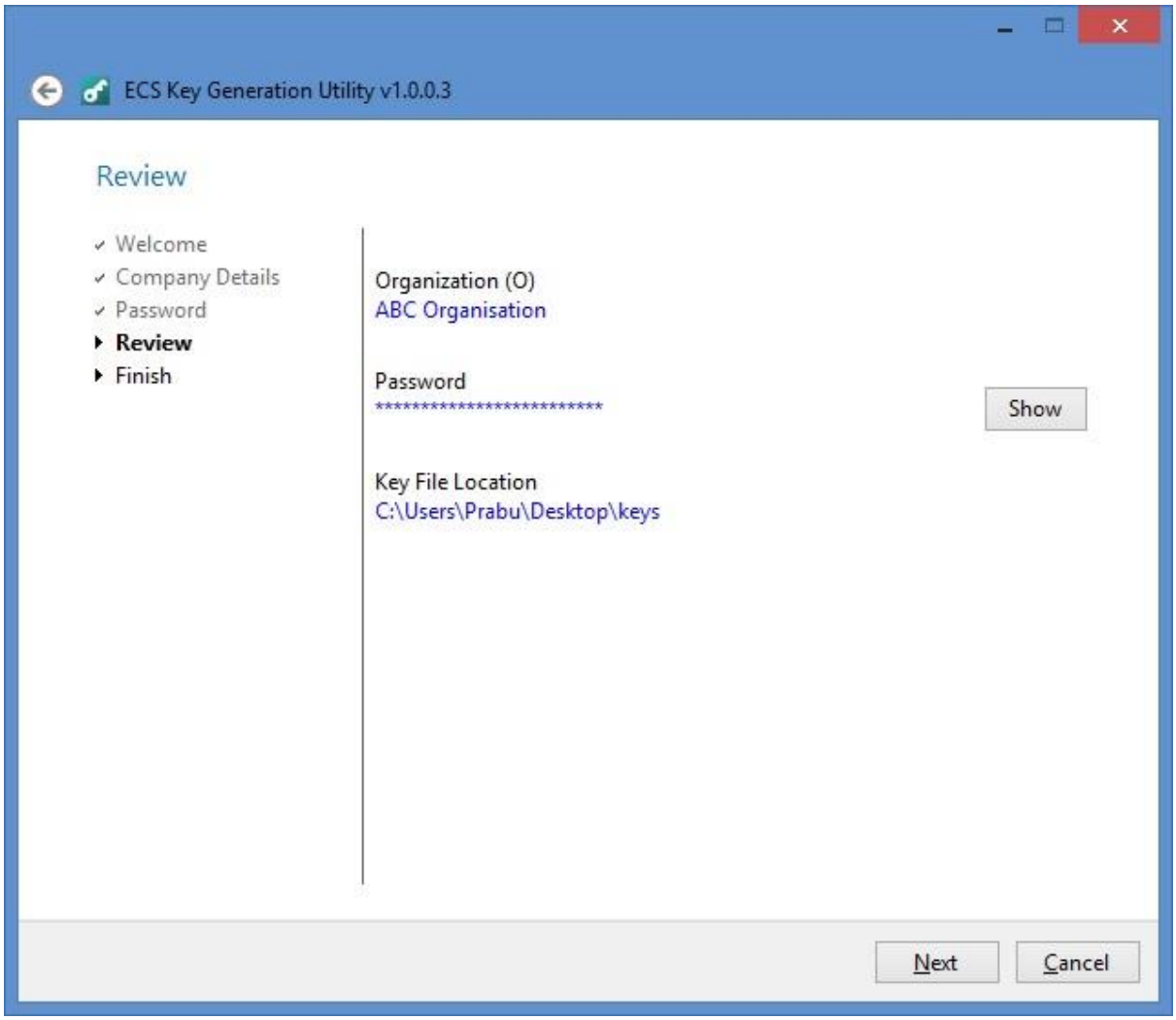
Also choose the directory to store the generated Certificates



Press '**Next**' to proceed

STEP 5:

Review the input details for Key Generation



Press '**Next**' to proceed

STEP 6:

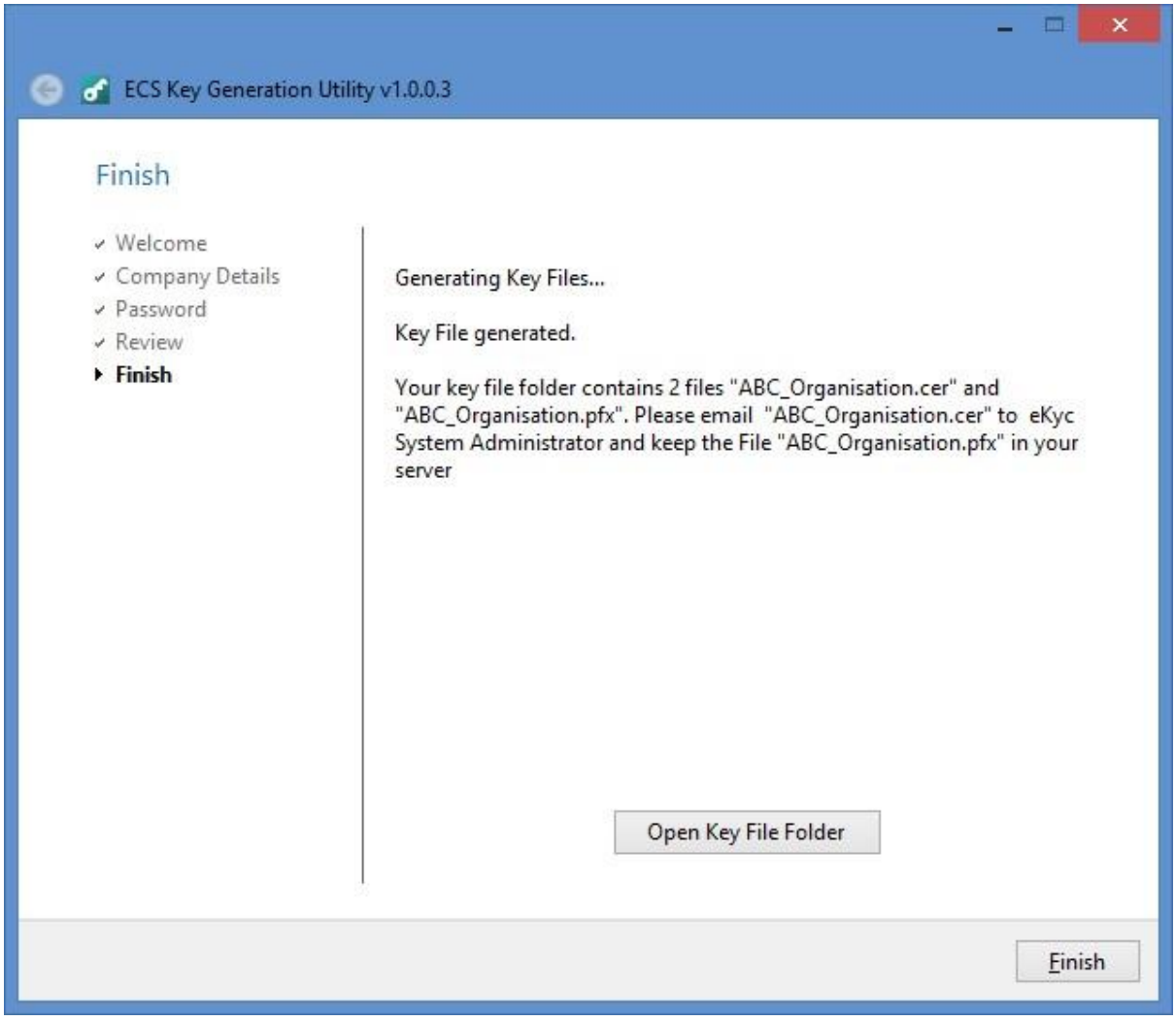
This step will generate Key Files and the generated key files are stored inside the chosen folder in STEP 4.

The Key Files will be generated in the name of the Organisation name entered in STEP 2.

The Private Key will have the extension .PFX and the Public Key will have the extension .CER

The Private Key file and the Key Password should be kept safe and will be used by the client at the time of sending request to CVL Esign Service.

The Public Key file will be shared with CVL as part of client enrolment process.



Press '**Finish'** to complete the Key Generation process.

ECS eSign Supporting Utility (Java / .Net) Details:

ECS eSign API allows the application developers to easily integration eSign capabilities into their business application.

Classes

- **ECSSigner**
  Core class for performing eSign.

- **SignCoordinate**
  Specifies the Signature coordinates in the PDF document

- **PdfSignCoordinates**
  Specifies the Signature coordinates of Signature in multiple pages

- **SignResponse**
  Response object containing the signed document. This object is returned on calling Sign Request.

ECSSigner

Constructor:

```
public ECSSigner(String clientId, String txnId, String signerName, String consent, byte[]pfxFile,String pfxPassword,String uid, String authMode, String responseForwardUrl);
```

Parameters	Description
clientId	Client ID assigned by CVL to the organisation for availing CVL eSign Services
txnId	Unique transaction ID generated by client application. This transaction ID shall be used by client application to map the request and response for completing the eSign process.
signerName	Name of the Signer (Appear above the signature in the PDF)
Consent	Consent provided by eSign user for eSigning the document.
pfxFile	Private key file for signing the request XML sent to CVL. CVL will validate the XML signature before processing the request.
pfxPassword	Password for the Pfx File for signing the XML request sent to CVL
Uid	This field is reserved for future use. This field should be null
authMode	eSign class used for signing the document. Allowed modes are: OTP – OTP based eSign FMR – Fingerprint biometric based eSign IIR – Iris biometric based eSign
ResponseForwardUrl	URL receiving eSign response from CVL

```
public void addPDFSigner(String identifier,  InputStream srcPdfStream, String  docInfo, String
signTextTemplate,String reason, String location, String  signatureBackgroundText, PdfSignCoordinates
multipageSignature) throws  Exception;
```

Add PDF document for signing.

Parameters	Description
Identifier	Document Identifier. This Identifier will be sent back in SignResponse. Identifier can be any string value that will uniquely identify the document. eg., Document ID
srcPdfStream	PDF document to be signed as InputStream
docInfo	Description for the respective document being signed, not more than 50 characters
signTextTemplate	Template of the Text displayed on the Visual signature. Template Parameters: - {#SIGNER#} represents Signer name - {#DATE#} represents eSign timestamp - {#REASON#} represents reason - {#LOCATION#} represents location (eg, Bangalore) Eg. Digitally Signed by {#SIGNER#} on {#DATE#} for {#REASON#}. Sign Location: {#LOCATION#}
Reason	Signing Reason. This will be embedded in the PDF document as Reason Text
Location	Signing Location (eg., Bengaluru). This will be embedded in PDF document
signatureBackgroundText	Background watermark text inside signature field. If the value is null then the signature will have transparent background.
multipageSignature	Coordinates of signatures to be embedded in single/multiple pages

```
public void addPDFSigner(String identifier,  InputStream srcPdfStream, String  pdfPassword, String
docInfo, String signTextTemplate, String reason, String  location, String  signatureBackgroundText,
PdfSignCoordinates  multipageSignature) throws  Exception;
```

Add PDF document for signing. Call this function to add more documents for signing

Parameters	Description
Identifier	Document Identifier. This Identifier will be sent back in SignResponse. Identifier can be any string value that will uniquely identify the document. eg., output filename
srcPdfStream	PDF document to be signed as InputStream
pdfPassword	Password of the source PDF document
docInfo	Description for the respective document being signed, not more than 50 characters
signTextTemplate	Template of the Text displayed on the Visual signature.

	Template Parameters: - {#SIGNER#} represents Signer name - {#DATE#} represents eSign timestamp - {#REASON#} represents reason - {#LOCATION#} represents location (eg, Bangalore) Eg. Digitally Signed by {#SIGNER#} on {#DATE#} for {#REASON#}. Sign Location: {#LOCATION#} If this parameter is null, default Signature Text rendering will be used for rendering the text in the signature
Reason	Signing Reason / Document info (Maximum 50 characters) (eg., Loan Document. This will be embedded in signed PDF document
Location	Signing Location (eg., Bengaluru). This will be embedded in signed PDF document
signatureBackgroundText	Background watermark text inside signature field. If the value is null then the signature will have transparent background.
multipageSignature	Coordinates of signatures to be embedded in single/multiple pages

public void clear();

Removes all the added document from the signing list

public String prepareSignRequest() **throws** ECSESignException

Prepare request data containing document hash. The response data will be sent to CVL for processing the eSign request.

public SignResponse sign(String resXml, byte[] verifyCert)

Sign the document and send the SignResponse object.

Parameters	Description
resXml	Contains response XML containing signed HASH and certificates.
verifyCert	Public Key certificate of CVL for validating response data from CVL

public static void writeFile(String filename, **byte**[] data) **throws** Exception;

Write byte[] data to file. Support function to save signed document to file

Parameters	Description
Filename	File name with path to save the file
Data	Data to be written into the file

public static byte[] readFile(String filename) **throws** Exception;

Read the whole file and return the content as byte[]

Parameters	Description
Filename	File name with path to read the file

SignCoordinate

public SignCoordinate(**int** page, **float** x, **float** y, **float** width, **float** height);

Specifies Coordinates of the Visible Signature element

Parameters	Description
Page	Page number to sign
X	X coordinate from page left
Y	Y coordinate from page top
Width	Width of the signature
Height	Height of the signature

PdfSignCoordinates

public void addSignatureInAllPages(**float** x, **float** y, **float** width, **float** height)
throws Exception

Add signature in all Pages in specific coordinates. The Signature will be image representation in all the pages. If any of the page contains original Signature in the same location as image signature, then for that specific page Image signature is not added.

Parameters	Description
X	X coordinate from page left
Y	Y coordinate from page top
Width	Width of the signature
Height	Height of the signature

public void addSignature(**int** page, **float** x, **float** y, **float** width, **float** height) **throws** Exception

Add signature in specific Pages in specific coordinates. The Signature will be image representation. This function shall be called multiple times to add signature in multiple pages

Parameters	Description
Page	Page number to sign
X	X coordinate from page left
Y	Y coordinate from page top
Width	Width of the signature
Height	Height of the signature

SignResponse

public boolean isError();

returns status of the eSign request

Return value	Description
True	eSign successful
False	eSign failed. Call getErrorCode() and getErrorMessage() to get error detail

public String getErrorCode();

returns error code in case of error. Will return null if isError() returns false

public String getErrorMessage();

returns error message in case of error. Will return null if isError() returns false

public String getAspTimestamp()

Will contain the response timestamp in ISO format
This method will return null if isError() returns true

public String getAspTxnId()

returns the Transaction ID of ASP (CVL Gateway) for processing the request
This method will return null if isError() returns true

public List<SignResponse.SignedData> getSignedData()

returns Array List of SignedData object.

SignedData object contains the following methods

Method	Return Value
public boolean isError()	Return true incase of error during signing Return false on successful signing
public String getErrorCode()	Return error code when isError() return true
public String getIdentifier()	Return the document Identifier which was sent while adding document for signing
public byte[] getData()	Return signed document. In case of PDF, the byte[] contains the signed PDF document. This method will return null if isError() return true

ECSESignException

The exception thrown by prepareSignRequest() method ECSESign class in the case of error preparing request data.

Method	Return Value
public String getErrorCode()	Return error code
public String getMessage()	Return the error message description

eSign Application Development Process flow

The application to perform eSign should be developed as Web Application. For eSigning the document, request data should be prepared and the browser will be forwarded to CVL eSign URL and the response from CVL containing document Hash and certificate will be sent to response URL which is sent in the request.

The client application should use ECSSigner class for preparing the request and processing the response for signing the PDF document. The client application should maintain the ECSSigner object reference in session and should use the same ECSSigner object reference in response page for embedding the signature (ie., Same ECSSigner instance should be used during sign request and sign response)

If the application server is behind load balancer, the load balancer shall be configured for sticky session.

JAVA Sample Code (Request Page)

```
<%@page import="com.ecs.esign.webapi.tester.Base64"%>
<%@page import="java.util.HashMap"%>
<%@page import="java.io.ByteArrayInputStream"%>
<%@page import="com.ecs.esign.api.data.PdfSignCoordinates"%>
<%@page import="com.ecs.esign.webapi.tester.Utils"%>
<%@page import="com.ecs.esign.api.ECSSigner"%>
<%@page import="java.io.InputStream"%>
<%@page import="java.util.UUID"%>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>ESign Request</title>
</head>
<body>

<%
```

```
//private String SIGNATURE_TEXT_TEMPLATE = "Digitally Signed by {#SIGNER#} on {#DATE#} for {#REASON#}.
Sign Location: {#LOCATION#}";

String CLIENT_ID = "**** REPLACE WITH CLIENT ID ***";

String SIGNATURE_TEXT_TEMPLATE = null;
String SIGNER_NAME = "**** REPLACE WITH SIGNER NAME ***";
String UID = null;
byte[] PDF = Utils.getBytes("**** REPLACE WITH PDF FILE***");
byte[] DS_FILE = Utils.getBytes("**** REPLACE WITH PFX FILE ***");
String DS_PWD = "**** REPLACE WITH PFX PASSWORD ***";
String RESPONSE_URL = "**** REPLACE WITH RESPONSE URL ***";
String ESIGN_URL = "**** REPLACE WITH ESIGN URL ***";

String AUTH_MODE = "OTP";
String CONSENT = "**** REPLACE WITH SIGNER CONSENT FOR SIGN THE DOCUMENT ***";
String DOC_INFO="Trading Account opening form";
String SIGN_REASON="Account Opening"; // Displayed in Signature of PDF
String SIGN_LOCATION="Bengaluru"; // Displayed in Signature of PDF
String SIGNATURE_BG_TEXT = "Sign"; // Background text in Signature of PDF


ServletContext context = getServletContext();

//Session
HashMap<String, ECSSigner> hmGlobal = null;
if(context.getAttribute("SIGN_SESSION") == null)
{
    hmGlobal = new HashMap<String,ECSSigner>();
    context.setAttribute("SIGN_SESSION", hmGlobal);
}
else
{
    hmGlobal = (HashMap<String,ECSSigner>) context.getAttribute("SIGN_SESSION");
}


String txnId = null;

// Search for Duplicate TxnId in Session
while(true)
{
    txnId = UUID.randomUUID().toString();

    if(hmGlobal.containsKey(txnId) == false)
        break;
}


// Create ECSSigner object
ECSSigner signer = new ECSSigner(aspId,  txnId, SIGNER_NAME, DS_FILE, DS_PWD, UID,
AUTH_MODE, RESPONSE_URL);

PdfSignCoordinates coordinates = new PdfSignCoordinates();
coordinates.addSignatureInAllPages(436, 713, 158, 81);
// Sign in all Pages
```

```
// Following commented lines are used to specify coordinates for signing in
// specific pages

//coordinates.addSignature(2,431,685,167,48);
//coordinates.addSignature(4,400,775,172,46);
//coordinates.addSignature(5,120,850,175,55);
//coordinates.addSignature(6,373,716,182,51);

signer.addPDFSigner("/ESIGN_SAMPLE_OUTPUT.pdf",new ByteArrayInputStream(pdfBytes),
DOC_INFO, SIGNATURE_TEXT_TEMPLATE, SIGN_REASON, SIGN_LOCATION, SIGNATURE_BG_TEXT,
coordinates);


// Prepare request XML
String requestXml = signer.prepareSignRequest();

// encode request XML to base64
String b64Request = Base64.encode(requestXml.getBytes());

// Store ECSSigner object in Global session

hmGlobal.put(txnId, signer);


// Forward the browser to CVL

%>
<!-- getServletConfig().getServletContext().getAttribute(arg0) // Global Attribute
 -->

<form id="frm " name="frm "
      ACTION="<%=ESIGN_URL%>" METHOD="post">
      <input type="hidden" name="ESIGN_REQUEST_XML" id="ESIGN_REQUEST_XML"
         ="<%=b64Request%>"> <input type="hidden" name="CONSENT_ID" value="1">
   </form>
    <script language="JavaScript" type="text/javascript">
  document.frm.submit();
  </script>
</body>
</html>
```

JAVA Sample Code (Response Page)

```
<%@page import="java.io.FileOutputStream"%>
<%@page import="java.io.File"%>
<%@page import="com.ecs.esign.response.SignResponse"%>
<%@page import="com.ecs.esign.webapi.tester.Base64"%>
<%@page import="com.ecs.esign.api.ECSSigner"%>
<%@page import="java.util.HashMap"%>
<%@page import="com.ecs.asp.esign.xsd.AspSignResponse"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html lang="en">
<head>
     <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <title>eSign Response </title>
</head>
<body>
<%
boolean isError = false;
String message = "";

// Retrieve Transaction ID and response data

try {

    String signedData = request.getParameter("ESIGN_RES");

    ServletContext context = getServletContext();
    InputStream is = context.getResourceAsStream("/WEB-INF/settings.properties");
    SettingsReader settings = new SettingsReader(is);

    byte[] DS_FILE =
Utils.getBytes(context.getResourceAsStream(settings.getProperty("ESIGN_PUBLIC_CERT")));

    if(signedData == null) {
        isError = true;
        message = "Unknown Error - SignedData is NULL";
        throw new Exception("");
    }

    //Retrieve ECSSigner object from global session
    HashMap<String, ECSSigner> hmGlobal = (HashMap<String,ECSSigner>)
    context.getAttribute("SIGN_SESSION");

    if(hmGlobal == null) {
        isError = true;
        message = "Unable to read session";
        throw new Exception("");
    }
    txnId = ECSSigner.getTxnId(signedData);
```

```
    if(!hmGlobal.containsKey(txnId)) {
        isError = true;
        message = "Unable to find transaction id in session";
        throw new Exception("");
    }

    ECSSigner signer = hmGlobal.get(txnId);


    //VERIFY RESPONSE SIGNATURE and Sign the PDF by passing the response data to ECSSigner object
        SignResponse signResponse = signer.sign(new String(Base64.decode(signedData),    "UTF-8"), DS_FILE);

    if(signResponse.isError()) {
        isError = true;
        message = "Error Occurred. Code: " + signResponse.getErrorCode() + ", Message: " +
signResponse.getErrorMessage();
        throw new Exception("");
    }

    //ALERT!!! - Since ESP supports only one document, the sample code reads only the first response and
saves it
    File file = new File("/temp/SignedPdf.pdf");        //NOTE: Please ensure that the file is not open in external
application
    FileOutputStream fos = new FileOutputStream(file);
    fos.write(signResponse.getSignedData().get(0).getData());
    fos.close();

    isError = false;
    message = "Document eSigned Successfully";

} catch(Exception ex) {
    if(ex.getMessage() != "") {
        ex.printStackTrace();

        isError = true;
        message = "Internal Server Error";
    }
}

</body>
</html>
```

.Net Sample Code (Request Page)

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Request.aspx.cs"
Inherits="eSignV20Demo.Request" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>ESign Request</title>
</head>
<body>

<%
//private String SIGNATURE_TEXT_TEMPLATE = "Digitally Signed by {#SIGNER#} on {#DATE#} for {#REASON#}.
Sign Location: {#LOCATION#}";

String CLIENT_ID = "**** REPLACE WITH CLIENT ID ***";

String SIGNATURE_TEXT_TEMPLATE = null;
String SIGNER_NAME = "**** REPLACE WITH SIGNER NAME ***";
String UID = "**** REPLACE WITH SIGNER AADHAAR NUMBER ***";
byte[] PDF = System.IO.File.ReadAllBytes("**** REPLACE WITH PDF FILE***");
byte[] DS_FILE = System.IO.File.ReadAllBytes("**** REPLACE WITH PFX FILE ***");
String DS_PWD = "**** REPLACE WITH PFX PASSWORD ***";
String RESPONSE_URL = "**** REPLACE WITH RESPONSE URL ***";
String ESIGN_API_URL = "**** REPLACE WITH ESIGN API URL ***";

String AUTH_MODE = "OTP";
String CONSENT = "**** REPLACE WITH SIGNER CONSENT FOR SIGN THE DOCUMENT ***";
String DOC_INFO="Trading Account opening form";
String SIGN_REASON="Account Opening"; // Displayed in Signature of PDF
String SIGN_LOCATION="Bengaluru"; // Displayed in Signature of PDF
String SIGNATURE_BG_TEXT = "Sign"; // Background text in Signature of PDF


String txnId = null;

// Search for Duplicate TxnId in Session
while(true)
{
    txnId = Guid.NewGuid().ToString();

    if(Application.AllKeys.FirstOrDefault(k=> k == txnId) == null)
        break;
}


// Create ECSSigner object
com.ecs.esign.api.ECSSigner signer = new com.ecs.esign.api.ECSSigner(CLIENT_ID,  txnId, SIGNER_NAME,
DOC_INFO, DS_FILE, DS_PWD, UID, AUTH_MODE, RESPONSE_URL);

com.ecs.esign.api.data.PdfSignCoordinates coordinates = new com.ecs.esign.api.data.PdfSignCoordinates();
coordinates.AddSignatureInAllPages(436, 713, 149, 69); // Sign in all Pages
```

```
// Following commented lines are used to specify coordinates for signing in
// specific pages

//coordinates.AddSignature(2,431,685,167,48);
//coordinates.AddSignature(4,400,775,172,46);
//coordinates.AddSignature(5,120,850,175,55);
//coordinates.AddSignature(6,373,716,182,51);

signer.AddPDFSigner("AC_OPENING_DOC",new System.IO.MemoryStream(PDF), CONSENT,
SIGNATURE_TEXT_TEMPLATE, SIGN_REASON, SIGN_LOCATION, SIGNATURE_BG_TEXT, coordinates);


// Prepare request XML
String requestXml = signer.PrepareSignRequest();

// encode request XML to base64
String b64Request = Convert.ToBase64String(System.Text.Encoding.UTF8.GetBytes(requestXml));

// Store ECSSigner object in Global session

Application[txnId] = signer;


// Forward the browser to CDSL

%>
<!--  getServletConfig().getServletContext().getAttribute(arg0) // Global Attribute
 -->

<FORM  name = "frm" ACTION="<%=ESIGN_API_URL%>" METHOD="post">
        <input type="hidden" name="ESIGN_REQUEST_XML" value="<%=b64Request%>">
        <input type="hidden" name="CONSENT_ID" value="1">
 </FORM>
      <script language="JavaScript" type="text/javascript">
   document.frm.submit();
   </script>
</body>
</html>
```

.Net Sample Code (Response Page)

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Response.aspx.cs"
Inherits="eSignV20Demo.Response" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <title>eSign Response </title>
</head>
<body>
<%
bool isError = false;
String message = "";

// Retrieve Transaction ID and response data

try {

    String signedData = Request["ESIGN_RES"];

    byte[] DS_FILE = System.IO.File.ReadAllBytes("**** REPLACE WITH CDSL POUBLIC KEY FILE***");

    if(signedData == null) {
        isError = true;
        message = "Unknown Error - SignedData is NULL";
        throw new Exception("");
    }

    //Retrieve ECSSigner object from global session
    txnId = ECSSigner.getTxnId(signedData);

    if(Application.AllKeys.FirstOrDefault(k=> k == txnId) == null) {
        isError = true;
        message = "Unable to find transaction id in session";
        throw new Exception("");
    }

    com.ecs.esign.api.ECSSigner signer = (com.ecs.esign.api.ECSSigner) Application[txnId];


//Verify response data and Sign the PDF by passing the response data to ECSSigner object
    com.ecs.esign.response.SignResponse signResponse =
signer.Sign(System.Text.Encoding.UTF8.GetString(Convert.FromBase64String(signedData)), DS_FILE);

    if(signResponse.Error) {
        isError = true;
```

```
        message = "Error Occurred. Code: " + signResponse.ErrorCode + ", Message: " +
signResponse.ErrorMessage;
        throw new Exception("");
    }

    //ALERT!!! - Since ESP supports only one document, the sample code reads only the first response
and saves it
    System.IO.File.WriteAllBytes("/temp/SignedPdf.pdf", signResponse.signedData[0].Data); //NOTE:
Please ensure that the file is not open in external application

    isError = false;
    message = "Document eSigned Successfully";

} catch(Exception ex) {
    if(ex.Message != "") {

        isError = true;
        message = "Internal Server Error";
    }
}
%>
</body>
</html>
```