using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.IO;

using System.Net;

using System.Security.Cryptography;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using Newtonsoft.Json.Linq;

using iText.Kernel.Geom;

using iText.Kernel.Pdf;

using iText.Signatures;

using Org.BouncyCastle.Asn1;

using Org.BouncyCastle.Cms;

using Org.BouncyCastle.Crypto;

using Org.BouncyCastle.Crypto.Digests;

using Org.BouncyCastle.Math;

using Org.BouncyCastle.Ocsp;

using Org.BouncyCastle.Tsp;

using Org.BouncyCastle.Utilities.Encoders;

using Org.BouncyCastle.X509;

using Sign;

using X509Certificate = Org.BouncyCastle.X509.X509Certificate;

namespace Sign

{

 public class ApiConnect

 {

 public static string baseURL = "https://emea.api.dss.globalsign.com:8443/v2";

 private static string fieldName = "sig1";

 private static string RESOURCE\_FOLDER = "../../resources/gstest/";

 private static string SRC = RESOURCE\_FOLDER + "test.pdf";

 private static string DEST = RESOURCE\_FOLDER + "test\_dss\_production.pdf";

 private static string LTV = RESOURCE\_FOLDER + "LTV\_dss\_production.pdf";

 private static X509Certificate2Collection collection;

 public static JObject Login(String aURL, String aKey, String aSecret)

 {

 Uri loginURL = new Uri(aURL + "/login");

 JObject apiLogin = new JObject();

 apiLogin.Add("api\_key", aKey);

 apiLogin.Add("api\_secret", aSecret);

 var httpWebRequest = (HttpWebRequest) WebRequest.Create(loginURL);

 httpWebRequest.Method = "POST";

 httpWebRequest.ContentType = "application/json; charset=UTF-8";

 httpWebRequest.ContentLength = apiLogin.ToString().Length;

 httpWebRequest.ClientCertificates = collection;

 //Send Request

 using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))

 {

 streamWriter.Write(apiLogin.ToString());

 }

 //Get Response

 var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();

 string result;

 using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))

 {

 result = streamReader.ReadToEnd();

 }

 JObject accessCode = JObject.Parse(result);

 return accessCode;

 }

 public static JObject Identity(String aURL, JObject aObj)

 {

 Uri loginURL = new Uri(aURL + "/identity");

 //info for certificate with individual identities

 JObject apiID = new JObject();

 JObject subj = new JObject();

 //subj.Add("common\_name", "ENTER YOUR N@ME");

 //apiID.Add("subject\_dn", subj);

 String token = (String) aObj.GetValue("access\_token");

 var httpWebRequest = (HttpWebRequest) WebRequest.Create(loginURL);

 httpWebRequest.Method = "POST";

 httpWebRequest.Headers.Add("Authorization", "Bearer " + token);

 httpWebRequest.ContentType = "application/json; charset=UTF-8";

 httpWebRequest.ContentLength = apiID.ToString().Length;

 httpWebRequest.ClientCertificates = collection;

 //Send Request

 using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))

 {

 streamWriter.Write(apiID.ToString());

 }

 //Get Response

 var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();

 string result;

 using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))

 {

 result = streamReader.ReadToEnd();

 }

 JObject identity = JObject.Parse(result);

 return identity;

 }

 public static JObject CertificatePath(String aURL, JObject aObj)

 {

 Uri loginURL = new Uri(aURL + "/certificate\_path");

 String token = (String) aObj.GetValue("access\_token");

 var httpWebRequest = (HttpWebRequest) WebRequest.Create(loginURL);

 httpWebRequest.Method = "GET";

 httpWebRequest.Headers.Add("Authorization", "Bearer " + token);

 httpWebRequest.ClientCertificates = collection;

 //Get Response

 var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();

 string result;

 using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))

 {

 result = streamReader.ReadToEnd();

 }

 JObject path = JObject.Parse(result);

 return path;

 }

 public static JObject Sign(String aURL, String id, String digest, JObject aObj)

 {

 Uri loginURL = new Uri(aURL + "/identity/" + id + "/sign/" + digest);

 String token = (String) aObj.GetValue("access\_token");

 var httpWebRequest = (HttpWebRequest) WebRequest.Create(loginURL);

 httpWebRequest.Method = "GET";

 httpWebRequest.Headers.Add("Authorization", "Bearer " + token);

 httpWebRequest.ClientCertificates = collection;

 //Get Response

 var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();

 string result;

 using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))

 {

 result = streamReader.ReadToEnd();

 }

 JObject signature = JObject.Parse(result);

 return signature;

 }

 public static JObject Timestamp(String aURL, String digest, JObject aObj)

 {

 Uri loginURL = new Uri(aURL + "/timestamp/" + digest);

 String token = (String) aObj.GetValue("access\_token");

 var httpWebRequest = (HttpWebRequest) WebRequest.Create(loginURL);

 httpWebRequest.Method = "GET";

 httpWebRequest.Headers.Add("Authorization", "Bearer " + token);

 httpWebRequest.ClientCertificates = collection;

 //Get Response

 var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();

 string result;

 using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))

 {

 result = streamReader.ReadToEnd();

 }

 JObject time = JObject.Parse(result);

 return time;

 }

 public static X509Certificate[] CreateChain(String cert, String ca)

 {

 X509Certificate[] chainy = new X509Certificate[2];

 X509CertificateParser parser = new X509CertificateParser();

 chainy[0] = new X509Certificate(parser.ReadCertificate(Encoding.UTF8.GetBytes(cert)).CertificateStructure);

 chainy[1] = new X509Certificate(parser.ReadCertificate(Encoding.UTF8.GetBytes(ca)).CertificateStructure);

 return chainy;

 }

 class DSSTSAClient : ITSAClient

 {

 public static int DEFAULTTOKENSIZE = 4096;

 public static String DEFAULTHASHALGORITHM = "SHA-256";

 private JObject accessToken;

 public DSSTSAClient(JObject accessToken)

 {

 this.accessToken = accessToken;

 }

 public IDigest GetMessageDigest()

 {

 return new Sha256Digest();

 }

 public byte[] GetTimeStampToken(byte[] imprint)

 {

 TimeStampRequestGenerator tsqGenerator = new TimeStampRequestGenerator();

 tsqGenerator.SetCertReq(true);

 BigInteger nonce = BigInteger.ValueOf((long) (new TimeSpan(DateTime.Now.Ticks)).TotalMilliseconds);

 TimeStampRequest request = tsqGenerator.Generate(new DerObjectIdentifier(

 DigestAlgorithms.GetAllowedDigest(DEFAULTHASHALGORITHM)),

 imprint, nonce);

 JObject time = Timestamp(baseURL, Hex.ToHexString(request.GetMessageImprintDigest()),

 accessToken);

 String tst = (String) time.GetValue("token");

 byte[] token = Base64.Decode(tst);

 CmsSignedData cms = new CmsSignedData(token);

 TimeStampToken tstToken = new TimeStampToken(cms);

 return tstToken.GetEncoded();

 }

 public int GetTokenSizeEstimate()

 {

 return DEFAULTTOKENSIZE;

 }

 }

 static void addLTV(String src, String dest, IOcspClient ocsp, ICrlClient crl,

 LtvVerification.Level timestampLevel, LtvVerification.Level signatureLevel)

 {

 PdfDocument pdfDoc = new PdfDocument(new PdfReader(new FileStream(src, FileMode.Open)),

 new PdfWriter(new FileStream(dest, FileMode.OpenOrCreate)),

 new StampingProperties().UseAppendMode());

 LtvVerification v = new LtvVerification(pdfDoc);

 SignatureUtil signatureUtil = new SignatureUtil(pdfDoc);

 IList<string> names = signatureUtil.GetSignatureNames();

 String sigName = names[(names.Count - 1)];

 PdfPKCS7 pkcs7 = signatureUtil.ReadSignatureData(sigName);

 if (pkcs7.IsTsp())

 {

 v.AddVerification(sigName, ocsp, crl, LtvVerification.CertificateOption.WHOLE\_CHAIN,

 timestampLevel, LtvVerification.CertificateInclusion.YES);

 }

 else

 {

 foreach (String name in names)

 {

 v.AddVerification(name, ocsp, crl, LtvVerification.CertificateOption.WHOLE\_CHAIN,

 signatureLevel, LtvVerification.CertificateInclusion.YES);

 }

 }

 v.Merge();

 pdfDoc.Close();

 }

 public static void Main(String[] args)

 {

 collection = new X509Certificate2Collection();

 collection.Import(GsConfig.SslCertificatePath(), GsConfig.GetKeyPassword(),

 X509KeyStorageFlags.DefaultKeySet);

 ServicePointManager.Expect100Continue = true;

 ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;

 //get JSON access token

 JObject access = Login(baseURL, GsConfig.GetApiKey(), GsConfig.GetApiSecret());

 //get JSON with id/certificate/ocsp response

 JObject identity = Identity(baseURL, access);

 String cert = (String) identity.GetValue("signing\_cert");

 String id = (String) identity.GetValue("id");

 String oc1 = (String) identity.GetValue("ocsp\_response");

 JObject path = CertificatePath(baseURL, access);

 String ca = (String) path.GetValue("path");

 //Create Certificate chain

 X509Certificate[] chain = CreateChain(cert, ca);

 String temp = RESOURCE\_FOLDER + Guid.NewGuid() + ".pdf";

 //create empty signature

 PdfReader reader = new PdfReader(SRC);

 PdfSigner stamper;

 using (FileStream os = new FileStream(temp, FileMode.OpenOrCreate))

 {

 stamper = new PdfSigner(reader, os, new StampingProperties());

 PdfSignatureAppearance appearance = stamper.GetSignatureAppearance();

 appearance.SetPageRect(new Rectangle(36, 508, 254, 200));

 appearance.SetPageNumber(1);

 appearance.SetLayer2FontSize(14f);

 stamper.SetFieldName(fieldName);

 appearance.SetReason("ENTER YOUR RE@SON");

 IExternalSignatureContainer external = new ExternalBlankSignatureContainer(PdfName.Adobe\_PPKLite,

 PdfName.Adbe\_pkcs7\_detached);

 stamper.SignExternalContainer(external, 8192);

 }

 //OCSP

 byte[] oc2 = Convert.FromBase64String(oc1);

 OcspResp ocspResp = new OcspResp(oc2);

 IExternalSignatureContainer gsContainer = new MyExternalSignatureContainer(id, access, chain, ocspResp);

 using (FileStream os1 = new FileStream(DEST, FileMode.OpenOrCreate))

 {

 PdfSigner signer = new PdfSigner(new PdfReader(temp), os1, new StampingProperties());

 PdfSigner.SignDeferred(signer.GetDocument(), fieldName, os1, gsContainer);

 }

 addLTV(DEST, LTV, new OcspClientBouncyCastle(null),

 new CrlClientOnline(), LtvVerification.Level.OCSP\_CRL,

 LtvVerification.Level.OCSP\_CRL);

 Console.WriteLine("GS Finished");

 }

 class MyExternalSignatureContainer : IExternalSignatureContainer

 {

 private String id;

 private X509Certificate[] chain;

 private JObject access;

 private OcspResp ocspResp;

 public MyExternalSignatureContainer(String id, JObject access, X509Certificate[] chain, OcspResp ocspResp)

 {

 this.id = id;

 this.access = access;

 this.chain = chain;

 this.ocspResp = ocspResp;

 }

 public byte[] Sign(Stream data)

 {

 BasicOcspResp basicResp = (BasicOcspResp) ocspResp.GetResponseObject();

 byte[] oc = basicResp.GetEncoded();

 Collection<byte[]> ocspCollection = new Collection<byte[]>();

 ocspCollection.Add(oc);

 String hashAlgorithm = "SHA256";

 PdfPKCS7 sgn = new PdfPKCS7(null, chain, hashAlgorithm, false);

 byte[] hash = DigestAlgorithms.Digest(data, DigestAlgorithms.GetMessageDigest(hashAlgorithm));

 byte[] sh = sgn.GetAuthenticatedAttributeBytes(hash, PdfSigner.CryptoStandard.CADES, ocspCollection,

 null);

 //create sha256 message digest

 using (SHA256 sha256 = SHA256.Create())

 {

 sh = sha256.ComputeHash(sh);

 }

 //create hex encoded sha256 message digest

 String hexencodedDigest = new BigInteger(1, sh).ToString(16).ToUpper();

 JObject signed = ApiConnect.Sign(baseURL, id, hexencodedDigest, access);

 String sig = (String) signed.GetValue("signature");

 //decode hex signature

 byte[] dsg = Hex.Decode(sig);

 //include signature on PDF

 sgn.SetExternalDigest(dsg, null, "RSA");

 //create TimeStamp Client

 ITSAClient tsc = new DSSTSAClient(access);

 return sgn.GetEncodedPKCS7(hash, PdfSigner.CryptoStandard.CADES, tsc, ocspCollection, null);

 }

 public void ModifySigningDictionary(PdfDictionary signDic)

 {

 }

 }

 public static class GsConfig

 {

 public static String GetApiSecret()

 {

 return "yourAPISecret";

 }

 public static String GetApiKey()

 {

 return "yourAPIKey";

 }

 public static String GetKeyPassword()

 {

 return "youpfxpassword";

 }

 public static String SslCertificatePath()

 {

 return "../../resources/gstest/your.pfx";

 }

 }

 }

}