import com.itextpdf.kernel.geom.Rectangle;

import com.itextpdf.kernel.pdf.PdfDictionary;

import com.itextpdf.kernel.pdf.PdfDocument;

import com.itextpdf.kernel.pdf.PdfName;

import com.itextpdf.kernel.pdf.PdfReader;

import com.itextpdf.kernel.pdf.PdfWriter;

import com.itextpdf.kernel.pdf.StampingProperties;

import com.itextpdf.signatures.\*;

import org.apache.commons.codec.DecoderException;

import org.apache.commons.codec.binary.Hex;

import org.bouncycastle.asn1.ASN1ObjectIdentifier;

import org.bouncycastle.cert.ocsp.BasicOCSPResp;

import org.bouncycastle.cert.ocsp.OCSPException;

import org.bouncycastle.cert.ocsp.OCSPResp;

import org.bouncycastle.cms.CMSSignedData;

import org.bouncycastle.jce.provider.BouncyCastleProvider;

import org.bouncycastle.tsp.\*;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;

import org.json.simple.parser.ParseException;

import javax.net.ssl.\*;

import java.io.\*;

import java.math.BigInteger;

import java.net.URL;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.security.GeneralSecurityException;

import java.security.KeyStore;

import java.security.MessageDigest;

import java.security.Security;

import java.security.cert.Certificate;

import java.security.cert.CertificateException;

import java.security.cert.CertificateFactory;

import java.security.cert.X509Certificate;

import java.util.Base64;

import java.util.Collection;

import java.util.Collections;

import java.util.List;

import java.util.UUID;

public class ApiConnect {

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

 private static final String fieldName = "sig1";

 private static String RESOURCE\_FOLDER = "src/main/resources/";

 private static final String SRC = RESOURCE\_FOLDER + "hello\_world.pdf";

 private static final String DEST = RESOURCE\_FOLDER + "signed\_dss\_production.pdf";

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

 public static JSONObject login(String aURL, Object aKey, Object aSecret)

 throws IOException, ParseException {

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

 HttpsURLConnection conn = (HttpsURLConnection) loginURL.openConnection();

 JSONObject apiLogin = new JSONObject();

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

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

 conn.setRequestMethod("POST");

 conn.setRequestProperty("Content-Type", "application/json; charset=UTF-8");

 conn.setRequestProperty("Content-Length", "" + apiLogin.toString().length());

 //Send Request

 conn.setDoOutput(true);

 DataOutputStream os = new DataOutputStream(conn.getOutputStream());

 os.writeBytes(apiLogin.toString());

 os.flush();

 os.close();

 //Get Response

 BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));

 String aux = "";

 StringBuilder builder = new StringBuilder();

 while ((aux = br.readLine()) != null) {

 builder.append(aux);

 }

 String output = builder.toString();

 JSONParser parser = new JSONParser();

 JSONObject accessCode = (JSONObject) parser.parse(output);

 br.close();

 conn.disconnect();

 return accessCode;

 }

 public static JSONObject identity(String aURL, JSONObject aObj) throws IOException, ParseException {

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

 HttpsURLConnection conn = (HttpsURLConnection) loginURL.openConnection();

 //info for certificate with individual identities

 /\*\*JSONObject apiID = new JSONObject();

 JSONObject subj = new JSONObject();

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

 subj.put("country", "US");

 JSONArray adm = new JSONArray();

 adm.add("ENTER YOUR DEP@RTMENT");

 subj.put("organizational\_unit", adm);

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

 //info for organization certificate has been prepopulated so we send an empty request

 JSONObject apiID = new JSONObject();

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

 conn.setRequestMethod("POST");

 conn.setRequestProperty("Authorization", "Bearer " + token);

 conn.setRequestProperty("Content-Type", "application/json; charset=UTF-8");

 conn.setRequestProperty("Content-Length", "" + apiID.toString().length());

 //Send Request

 conn.setDoOutput(true);

 DataOutputStream os = new DataOutputStream(conn.getOutputStream());

 os.writeBytes(apiID.toString());

 os.flush();

 os.close();

 //Get Response

 BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));

 String aux = "";

 StringBuilder builder = new StringBuilder();

 while ((aux = br.readLine()) != null) {

 builder.append(aux);

 }

 String output = builder.toString();

 JSONParser parser1 = new JSONParser();

 JSONObject identity = (JSONObject) parser1.parse(output);

 br.close();

 conn.disconnect();

 return identity;

 }

 public static JSONObject certificatePath(String aURL, JSONObject aObj) throws IOException, ParseException {

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

 HttpsURLConnection conn = (HttpsURLConnection) loginURL.openConnection();

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

 conn.setRequestMethod("GET");

 conn.setRequestProperty("Authorization", "Bearer " + token);

 //Get Response

 BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));

 String aux = "";

 StringBuilder builder = new StringBuilder();

 while ((aux = br.readLine()) != null) {

 builder.append(aux);

 }

 String output = builder.toString();

 JSONParser parser = new JSONParser();

 JSONObject path = (JSONObject) parser.parse(output);

 br.close();

 conn.disconnect();

 return path;

 }

 public static JSONObject sign(String aURL, String id, String digest, JSONObject aObj)

 throws IOException, ParseException {

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

 HttpsURLConnection conn = (HttpsURLConnection) loginURL.openConnection();

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

 conn.setRequestMethod("GET");

 conn.setRequestProperty("Authorization", "Bearer " + token);

 //Get Response

 BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));

 String aux = "";

 StringBuilder builder = new StringBuilder();

 while ((aux = br.readLine()) != null) {

 builder.append(aux);

 }

 String output = builder.toString();

 JSONParser parser = new JSONParser();

 JSONObject signature = (JSONObject) parser.parse(output);

 br.close();

 conn.disconnect();

 return signature;

 }

 public static JSONObject timestamp(String aURL, String digest, JSONObject aObj) throws IOException, ParseException{

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

 HttpsURLConnection conn = (HttpsURLConnection) loginURL.openConnection();

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

 conn.setRequestMethod("GET");

 conn.setRequestProperty("Authorization", "Bearer "+ token);

 //Get Response

 BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));

 String aux = "";

 StringBuilder builder = new StringBuilder();

 while ((aux = br.readLine()) != null){

 builder.append(aux);

 }

 String output = builder.toString();

 JSONParser parser = new JSONParser();

 JSONObject time = (JSONObject) parser.parse(output);

 br.close();

 conn.disconnect();

 return time;

 }

 public static Certificate[] createChain(String cert, String ca) throws IOException, CertificateException {

 Certificate[] chainy = new Certificate[2];

 CertificateFactory fact = CertificateFactory.getInstance("X.509");

 X509Certificate cer = null;

 InputStream in = new ByteArrayInputStream(cert.getBytes("UTF-8"));

 cer = (X509Certificate) fact.generateCertificate(in);

 chainy[0] = (Certificate) cer;

 X509Certificate caCert = null;

 in = new ByteArrayInputStream(ca.getBytes("UTF-8"));

 caCert = (X509Certificate) fact.generateCertificate(in);

 chainy[1] = (Certificate) caCert;

 return chainy;

 }

static class DSSTSAClient implements ITSAClient{

 public static final int DEFAULTTOKENSIZE = 4096;

 public static final String DEFAULTHASHALGORITHM = "SHA-256";

 private final JSONObject accessToken;

 public DSSTSAClient (JSONObject accessToken){

 this.accessToken = accessToken;

 }

 public MessageDigest getMessageDigest() throws GeneralSecurityException {

 return MessageDigest.getInstance(DEFAULTHASHALGORITHM);

 }

 public byte[] getTimeStampToken(byte[] imprint) throws Exception {

 TimeStampRequestGenerator tsqGenerator = new TimeStampRequestGenerator();

 tsqGenerator.setCertReq(true);

 BigInteger nonce = BigInteger.valueOf(System.currentTimeMillis());

 TimeStampRequest request = tsqGenerator.generate(new ASN1ObjectIdentifier(DigestAlgorithms.getAllowedDigest(DEFAULTHASHALGORITHM)), imprint, nonce);

 JSONObject time = timestamp(baseURL,Hex.encodeHexString(request.getMessageImprintDigest()),accessToken);

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

 byte[] token = Base64.getDecoder().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) throws IOException, GeneralSecurityException

{

 PdfDocument pdfDoc = new PdfDocument(new PdfReader(new FileInputStream(src)), new PdfWriter(new FileOutputStream(dest)), new StampingProperties().useAppendMode());

 LtvVerification v = new LtvVerification(pdfDoc);

 SignatureUtil signatureUtil = new SignatureUtil(pdfDoc);

 List<String> names = signatureUtil.getSignatureNames();

 String sigName = names.get(names.size() - 1);

 PdfPKCS7 pkcs7 = signatureUtil.readSignatureData(sigName);

 if (pkcs7.isTsp())

 {

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

 timestampLevel, LtvVerification.CertificateInclusion.YES);

 }

 else

 {

 for (String name : names)

 {

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

 signatureLevel, LtvVerification.CertificateInclusion.YES);

 }

 }

 v.merge();

 pdfDoc.close();

}

 public static void main(String[] args) throws Exception {

 Security.addProvider(new BouncyCastleProvider());

 GSConfig config = new GSConfig();

 InputStream trustStream = new FileInputStream(config.sslCertificatePath());

 char[] trustPassword = config.getKeyPassword().toCharArray();

 KeyStore trustStore = KeyStore.getInstance(KeyStore.getDefaultType());

 trustStore.load(trustStream, trustPassword);

 KeyManagerFactory kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());

 kmf.init(trustStore, trustPassword);

 KeyManager[] kms = kmf.getKeyManagers();

 TrustManager[] trustAllCerts = new TrustManager[]{

 new X509TrustManager() {

 public void checkClientTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {

 }

 public void checkServerTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {

 }

 public X509Certificate[] getAcceptedIssuers() {

 return null;

 }

 }

 };

 SSLContext sslContext = SSLContext.getInstance("TLS");

 sslContext.init(kms, trustAllCerts, null);

 SSLContext.setDefault(sslContext);

 //get JSON access token

 JSONObject access = login(baseURL, config.getApiKey(), config.getApiSecret());

 //get JSON with id/certificate/ocsp respone

 JSONObject identity = identity(baseURL, access);

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

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

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

 JSONObject path = certificatePath(baseURL, access);

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

 //Create Certificate chain

 Certificate[] chain = createChain(cert, ca);

 String temp = RESOURCE\_FOLDER + UUID.randomUUID().toString() + ".pdf";

 //create empty signature

 PdfReader reader = new PdfReader(SRC);

 FileOutputStream os = new FileOutputStream(temp);

 PdfSigner 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("Test GS Jose");

 appearance.setLocation("GlobalSign Belgium");

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

 PdfName.Adbe\_pkcs7\_detached);

 stamper.signExternalContainer(external, 8192);

 //OCSP

 byte[] oc2 = Base64.getDecoder().decode(oc1);

 OCSPResp ocspResp = new OCSPResp(oc2);

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

 FileOutputStream os1 = new FileOutputStream(DEST);

 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);

 // Files.deleteIfExists(Paths.get(temp)); //by some reason itext does not release lock

 // Files.deleteIfExists(Paths.get(DEST));

 System.out.println("GS Finished");

 }

 static class MyExternalSignatureContainer implements IExternalSignatureContainer {

 protected final String id;

 private final Certificate[] chain;

 private final JSONObject access;

 private OCSPResp ocspResp;

 public MyExternalSignatureContainer(String id, JSONObject access, Certificate[] chain, OCSPResp ocspResp) {

 this.id = id;

 this.access = access;

 this.chain = chain;

 this.ocspResp = ocspResp;

 }

 public void modifySigningDictionary(PdfDictionary arg0) {

 }

 public byte[] sign(InputStream arg0) {

 try {

 BasicOCSPResp basicResp = (BasicOCSPResp) ocspResp.getResponseObject();

 byte[] oc = basicResp.getEncoded();

 Collection<byte[]> ocspCollection = Collections.singletonList(oc);

 String hashAlgorithm = "SHA256";

 BouncyCastleDigest digest = new BouncyCastleDigest();

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

 byte[] hash = DigestAlgorithms.digest(arg0, digest.getMessageDigest(hashAlgorithm));

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

 null);

 //create sha256 message digest

 sh = MessageDigest.getInstance("SHA-256").digest(sh);

 //create hex encoded sha256 message digest

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

 hexencodedDigest = hexencodedDigest.toUpperCase();

 JSONObject signed = ApiConnect.sign(baseURL, id, hexencodedDigest, access);

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

 //decode hex signature

 byte[] dsg = Hex.decodeHex(sig.toCharArray());

 //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);

 } catch (DecoderException | IOException | ParseException | GeneralSecurityException | OCSPException de) {

 throw new RuntimeException(de);

 }

 }

 }

 public static class GSConfig {

 public String getApiSecret() {

 return "00000000000000000000000000";

 }

 public String getApiKey() {

 return "000000000";

 }

 public String getKeyPassword() {

 return "yourJKSpassword";

 }

 public String sslCertificatePath() {

 return "src/main/resources/yourjks.jks";

 }

 }

}