FHIR authorization

1. Sugeneruojamas privatus raktas (šiuo atveju PEM formato, bet gali bti ir DER ar kitas):
   `openssl genrsa -out private.pem 1024`
   (pavyzdyje naudojamas DER rakto formatas, į pakonvertuoti galima
   `openssl pkcs8 -topk8 -in private.pem -inform PEM -out private.der -outform DER -nocrypt`)
2. Sugeneruojamas public RSA raktas PEM formatu:
   `openssl rsa -in private.pem -out public.pem -outform PEM -pubout`
3. Public raktas perduodamas (pvz., el. paštu) užregistravimui į ESPBI IS
4. Perduodamas įstaigos, kuri atsakinga už teisingų duomenų pateikimą, JAR kodas (pvz., el. paštu)
5. Užklausa į FHIR serverį pasirašoma. Užklausos pasirašymas aprašytas vykdomas pagal "ESPBI IS DUOMENŲ MAINŲ IR INTEGRACIJOS PROJEKTAVIMO DOKUMENTACIJA" skyriuje 2.5.1.1. Duomenų perdavimo sauga. (RFC5849)
6.
   Pavyzdinis Java programavimo kalbos kodas sudaryti Authorization header parametrą:

```
import net.oauth.OAuth;
import net.oauth.OAuthAccessor;
import net.oauth.OAuthConsumer;
import net.oauth.OAuthMessage;
import net.oauth.signature.RSA_SHA1;
import org.apache.cxf.common.util.Base64Utility;

import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.Signature;
import java.security.spec.PKCS8EncodedKeySpec;

/**
* @author Maksim Boiko (Maksim.Boiko@nortal.com)
*/
public class AuthenticationUtil {

public static void main(String[] args) throws Exception {
String authorizationHeader = AuthenticationUtil.getAuthorizationHeader(
"GET",
"http://195.12.185.33:8181/cxf/Patient",
"test",
"",
KeyFactory.getInstance("RSA").generatePrivate(new PKCS8EncodedKeySpec(readFile(getFile("private.der")
))));

System.out.println(authorizationHeader);

String authorizationHeader2 = AuthenticationUtil.getAuthorizationHeader(
"POST",
"http://195.12.185.33:8181/cxf/Patient",
"test",
"<here be xml>",
KeyFactory.getInstance("RSA").generatePrivate(new PKCS8EncodedKeySpec(readFile(getFile("private.der")
))));

System.out.println(authorizationHeader2);
}

public static String getAuthorizationHeader(String httpMethod, String url, String consumerKey, String
body, PrivateKey privateKey) throws Exception {

String signedBody = sign(body, privateKey);

OAuthMessage message = new OAuthMessage(httpMethod, url, null);

OAuthConsumer consumer = new OAuthConsumer(null, consumerKey, null, null);
consumer.setProperty(RSA_SHA1.PRIVATE_KEY, privateKey);
consumer.setProperty(OAuth.OAUTH_SIGNATURE_METHOD, OAuth.RSA_SHA1);
```

```
OAuthAccessor accessor = new OAuthAccessor(consumer);
message.addParameter("oauth_body_hash", signedBody);
message.addRequiredParameters(accessor);

return message.getAuthorizationHeader(null);
}


private static String sign(String message, PrivateKey privateKey) throws Exception {
Signature signer = Signature.getInstance("SHA1withRSA");
signer.initSign(privateKey);
signer.update(message.getBytes("UTF-8"));
byte[] signature = signer.sign();

return base64Encode(signature);
}

public static String base64Encode(byte[] b) {
return Base64Utility.encode(b);
}

private static File getFile(String keysFilePath) {
return new File(keysFilePath);
}

private static byte[] readFile(File file) throws IOException {
FileInputStream fileInputStream = null;
DataInputStream dataInputStream = null;
try {
fileInputStream = new FileInputStream(file);
dataInputStream = new DataInputStream(fileInputStream);
byte[] keyInDerFormat = new byte[(int) file.length()];
dataInputStream.readFully(keyInDerFormat);
return keyInDerFormat;
} finally {
fileInputStream.close();
dataInputStream.close();
}
```

```
}

}
```

7. Atliekama užklausa į FHIR serverį:
    a. Kreipiamasi į servisą, perduodant Authorization header parametrą
    b. Perduodamas requestor_id header parametras (nuoroda į Practitioner resursą, kuris nusako specialistą, kurio kontekste vykdoma užklausa)

       Pateikiamas GET užklausos pavyzdys:

```
GET /cxf/Patient HTTP/1.1
Host: 195.12.185.33:8181
requestor_id: 1000005152
Authorization: OAuth oauth_consumer_key="test", oauth_signature_method="RSA-SHA1",
oauth_timestamp="1432503230", oauth_nonce="390756402947300", oauth_version="1.0",
oauth_signature="ScweLqrW1dkZ71c1ph7gwY1QnlBGblidaCfQMCi2iV1qvFvjywi6y%2FB0%2BAULu%2B6ca3rCn
DRl5I7bFgIJp50JTOw3ftDcrxIvyoiFNZzuemj9tPmvnrjM61LM8dRK5FhxuGdg%2B0z7%2B47GfyOQBO4%2BZ0MBt
foRFn7B0hBKwKCUFm0%3D", oauth_body_hash="Vsl5kjJQE3RjCbzZEDW5XaJIfGlIj44uOVCwIBDdxjKdtB81U3Q
vpO9HFEtgVWQ7SPgofQ3xuc1VB97BcZZZGFuMXEWgPvqupzvhaCCwXB5a2RamJjdNrrMDV6BKUKWlgeKE/ETjDoy
mSN1eT4fc9CAr8APiRlGEff8+TvW74Dw="
```