Request signature Java example

```
package com.riddletag.recognitionserver.example;

import com.riddletag.recognitionserver.keys.SHA256Util; import okhttp3.*;

import javax.crypto.BadPaddingException; import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException; import javax.crypto.NoSuchPaddingException; import
java.io.IOException;

import java.nio.charset.StandardCharsets; import java.nio.file.Files;

import java.nio.file.Paths;

import java.security.*;

import java.security.spec.InvalidKeySpecException; import java.security.spec.X509EncodedKeySpec;
import java.time.Instant;

import java.util.Arrays;

import java.util.Base64;

public class OkHttpExample {

private final OkHttpClient httpClient = new OkHttpClient(); private String accessKey = "";

private static String base64RsaPublicKey = "";

final static private MessageDigest digest; static {

try {

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

} catch (NoSuchAlgorithmException e) { throw new RuntimeException(e);

} }

public static final MediaType JSON = MediaType.parse("application/json; charset=utf-8"); public static void
main(String[] args) throws Exception {

OkHttpExample obj = new OkHttpExample();

System.out.println("Testing 2 - Send Http POST request"); obj.sendPost();

}

private void sendPost() throws IOException, NoSuchPaddingException, NoSuchAlgorithmException,
InvalidKeyException, BadPaddingException, IllegalBlockSizeException {

String host = "i4awvkpjw6.execute-api.eu-west-1.amazonaws.com/OugqX2kL4p"; String uri =
"/profile/recognize";

String base64Photo = new String(Files.readAllBytes(Paths.get(""))); base64Photo = base64Photo.trim();
```

```
String body = "{\"base64Photo\": \"" + base64Photo + "\"}"; System.out.println("body: " + body);

String timeStampMillis = ((Long)Instant.now().toEpochMilli()).toString();

String stringForSigning = getStringForSigning(host, uri, body, timeStampMillis); PublicKey publicKey =
getPublicKey(base64RsaPublicKey);

String sBase64EncryptedHash = getSignature(stringForSigning, publicKey); RequestBody requestBody =
RequestBody.create(JSON, body);

Request request = new Request.Builder()

.url("https://" + host + uri) .addHeader("Access-Key", accessKey) .addHeader("Signature",
sBase64EncryptedHash) .addHeader("Timestamp", timeStampMillis) .post(requestBody)

.build();

Response response = httpClient.newCall(request).execute();

String sBase64EncryptedResponseHash = response.header("Signature");
System.out.println("sBase64EncryptedResponseHash: " + sBase64EncryptedResponseHash);
ResponseBody responseBody = response.body();

String responseString = null; if (responseBody == null){

System.out.println("RESPONSE IS NULL"); } else {

responseString = responseBody.string();

System.out.println("responseString: " + responseString); }

String responseForSignature = getStringForSigning(host, uri, responseString, timeStampMillis); byte[]
bHash = getHash(responseForSignature);

String sBase64Hash = Base64.getEncoder().encodeToString(bHash); System.out.println("sBase64Hash: "
+ sBase64Hash);

byte[] bEncryptedResponseHash = Base64.getDecoder().decode(sBase64EncryptedResponseHash);
byte[] bResponseHash = decrypt(bEncryptedResponseHash, publicKey);

String sBase64ResponseHash = Base64.getEncoder().encodeToString(bResponseHash);
System.out.println("sBase64ResponseHash: " + sBase64ResponseHash);

if(Arrays.equals(bHash, bResponseHash)) { System.out.println("Arrays are equals");

} else {

System.out.println("Arrays aren't equals");

}

}

public static String getStringForSigning(String host, String uri, String body, String timeStampMillis) { return
host + uri + body + timeStampMillis;

}
```

```
public static PublicKey getPublicKey(String base64PublicKey) { PublicKey publicKey = null;

try {

X509EncodedKeySpec keySpec = new
X509EncodedKeySpec(Base64.getDecoder().decode(base64PublicKey.getBytes()));

KeyFactory keyFactory = KeyFactory.getInstance("RSA"); publicKey =
keyFactory.generatePublic(keySpec);

return publicKey;

} catch (NoSuchAlgorithmException e) { e.printStackTrace();

} catch (InvalidKeySpecException e) { e.printStackTrace();

}

return publicKey; }

public static String getSignature(String stringForSigning, PublicKey publicKey) { byte[] bHash =
getHash(stringForSigning);

byte[] bEncryptedHash = encrypt(bHash, publicKey);

return Base64.getEncoder().encodeToString(bEncryptedHash);

}

public static byte[] getHash(String sForHashing) {

byte[] bFoHashing = sForHashing.getBytes(StandardCharsets.UTF_8); return
SHA256Util.digest.digest(bFoHashing);

}

public static byte[] encrypt(byte[] data, Key key) { return doFinal(data, Cipher.ENCRYPT_MODE, key);

}

public static byte[] decrypt(byte[] data, Key key) { return doFinal(data, Cipher.DECRYPT_MODE, key);

}

private static byte[] doFinal(byte[] data, int mode, Key key) { try {

Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding"); cipher.init(mode, key);

return cipher.doFinal(data);

}catch (Exception e) {

throw new RuntimeException(); }

} }
```