POSTIDENT

SCR-Ident API Guide 3 - Encryption

Standard Connect & Result (SCR) API

Version 2.0
21.06.2024

SCR-Ident API Guide 3 - Encryption

Contents

Changelog

Date	Change
21.06.2024	Added information on incoming encryption and structured document accordingly
06.03.2023	adaptation of the description to the current implementation, revision of the implementation guidelines
30.07.2021	Several Encryption settings marked as deprecated (Q3/2022)
05.10.2020	Clarification on the format of the format of the public key in x-scr-key
22.09.2020	Updated recommendations from RSA 2048 bit keys to 3072 bits and from RSA1_5 to RSA-OAEP-256
16.10.2017	Document renamed to "SCR-Ident API Guide 3 Encryption"
27.03.2017	Added ScrClientTool
27.01.2017	Updated sample data
06.01.2017	Improved Overview section
07.12.2016	Minor textual improvements
17.11.2016	Update on "Http-header" and "Sample requests"

1 Preface

This document is split in two parts explaining incoming and outgoing encryption (respectively named from the POSTIDENT system's point of view):

- Part I explains the symmetrical encryption method that can optionally be used to encrypt data sent from the business customer's system to the POSTIDENT system (e.g. to protect user data that is sent along with the order for an identification case).
- Part II explains the asymmetrical encryption method that is mandatorily used to encrypt data provided by the POSTIDENT system to the business customer's system.

2 Part I - Incoming Encryption

Symmetrical encryption is used for incoming data in the request body. The cipher is transmitted in JWE format. If you want to encrypt the incoming payload, you have to use the PBES2 encryption method.

2.1 Encrypt incoming payload

- During setup you should have received a data password for decrypting provided POSTIDENT results; this data password is used as password in PBES2 encryption.
- The following example uses the nimbus library for password-based encryption. You have to use the following parameters for incoming encryption
    - **JWEAlgorithm:** PBES2_HS512_A256KW
    - **EncryptionMethod:** A256GCM
    - **saltLength:** 8
    - **pbes2 iterations:** 1000 (to prevent DOS attacks this value is limited in the POSTIDENT system to the range from 1000 to 2000)

PBES2 encryption

```
/** Snippet used in scr encryption incoming documentation
* @param strPayload json payload string to be encrypted
* @param dataPassword password used for pbes2 encryption
* @return jwe Cipher
* @throws NoSuchAlgorithmException
* @throws JOSEException
*/
public String encryptPayloadScrIncomingSnippet(String strPayload, String
dataPassword)
    throws NoSuchAlgorithmException, JOSEException {
    // PBES2 is currently the only supported option
    JWEAlgorithm jweAlg = JWEAlgorithm.PBES2_HS512_A256KW;
    EncryptionMethod encMethod = EncryptionMethod.A256GCM;
    int saltLength = 8;
    int iterationen = 1000;
    JWEHeader header = new JWEHeader.Builder(jweAlg, encMethod).build();
    Payload payload = new Payload(strPayload);
    JWEObject jweObject = new JWEObject(header, payload);
    JWEEncrypter encrypter = null;
    encrypter = new PasswordBasedEncrypter(dataPassword, saltLength, iterationen);
    jweObject.encrypt(encrypter);
    String ret = jweObject.serialize();
    return ret;
}
```

2.2 Send encrypted payload to the POSTIDENT system

- To announce the encrypted payload to the POSTIDENT system, you have to set the HTTP content type application/jose
- If you dont want to use encrypted payload, simply use unencrypted json with content type application/json

```
curl --location --request PATCH 'https://<HOSTNAME>/api/scr/v1/<CLIENTID>/cases' \
--header 'Content-Type: application/jose' \
--header 'Authorization: Basic
bmljX3Jlc3Rfc2NyX2lkZW50OjJSeTYjY3ZQY3JHOXEjI1Q0ZVJTN0dHMw==' \
--data-raw
'eyJwMnMiOiJ0TFFjX0hIWWtjMCIsInAyYyI6MTAwMCwiZW5jIjoiQTI1NkdDTSIsImFsZyI6IlBCRVMyLU
hTNTEyK0EyNTZLVyJ9.smhH-zNsqI6HZ_89GBas4MWKEaiSTvcEe0-
hdGUqKpkfiWLJq_1OIg.8g6s0_ivZN8K9YLw.SoZv-
_Sj3sNCCD68aMez8_3dGqDarcal.46E1OFU3vD4TYd9PW5U7Ug'
```

HTTP level request data with incoming encrypted payload

```
HTTP:

PATCH /api/scr/v1/<CLIENTID>/cases HTTP/1.1
Host: <HOSTNAME>
Content-Type: application/jose
Authorization: Basic bmljX3Jlc3Rfc2NyX2lkZW50OjJSeTYjY3ZQY3JHOXEjI1Q0ZVJTN0dHMw==
Content-Length: 228

eyJwMnMiOiJ0TFFjX0hIWWtjMCIsInAyYyI6MTAwMCwiZW5jIjoiQTI1NkdDTSIsImFsZyI6IlBCRVMyLUh
TNTEyK0EyNTZLVyJ9.smhH-zNsqI6HZ_89GBas4MWKEaiSTvcEe0-
hdGUqKpkfiWLJq_1OIg.8g6s0_ivZN8K9YLw.SoZv-
_Sj3sNCCD68aMez8_3dGqDarcal.46E1OFU3vD4TYd9PW5U7Ug
```

2.3 Encryption Details

The encryption of the incoming SCR payload is optional and is based on JWE (JSON Web Encryption https://tools.ietf.org/html/rfc7516).
JWE is a standard based on JSON and Base64 for the exchange of encrypted data.
The JWE Compact serialization is used as the transmission format, which has the following form:

2.3.1 Components of the JWE Compact Serialization

```
BASE64URL(UTF8(JWE Protected Header)) || '.' ||
BASE64URL(JWE Encrypted Key) || '.' ||
BASE64URL(JWE Initialization Vector) || '.' ||
BASE64URL(JWE Ciphertext) || '.' ||'
BASE64URL(JWE Authentication Tag)
```

2.3.2 Password-based encryption PBES2 for incoming SCR encryption

⚠ The PBES2-HS512+A256KW algorithm is supported for incoming SCR encryption

- PBES stands for Password Based key derivation with underlying Encryption Scheme (https://tools.ietf.org/html/rfc2898).
- The content encryption key is encrypted using a password (pre-shared key) and embedded (wrapped) in the encrypted message.
- With BPES, the JWE header must contain salt and iteration counters. Example: p2s = jDqle9w_IjM, p2c = 1000
- When using the JOSE JWE implementation, p2s and p2c are automatically added to the header.
- A salt length of 8 and an iteration counter of 1000 are specified for SCR.
- Iteration counters above 2000 are rejected by SCR.

Password Based Encryption		
ALG Param	**Key Management Algorithm**	**support in JDK7 / JDK 8 / BouncyCastle**
PBES2-HS512+A256KW	PBES2 with HMAC SHA-512 and "A256KW" wrapping	✅ / ✅ / ✅

2.3.3 Payload Encryption with AES GCM using 256-bit key

The payload encryption (content encryption) method to be used for incoming SCR encryption is AES GCM using 256-bit key (enc=A256GCM).

Erforderliche Schlüssellängen für Direct Encryption			
ENC Param	**Content Encryption Algorithm**	**Key Length Bit / Byte**	**support in JDK7 / JDK 8 / BouncyCastle**
A256GCM	AES GCM using 256-bit key	256 / 32	❌ / ✅ / ✅

2.3.4 Example Payload

Encryption and serialization of the payload {"identityData": {"firstName": "string","lastName": "string"}

	PBES2-HS512+A256KW 1000 iterations, 8 byte salt
Header	{p2s=ujdHYse6D-4, *p2c=1000, enc=A256GCM, alg=PBES2-HS512+A256KW*}
JWE	eyJwMnMiOiJ1amRIWXNlNkQtNCIsInAyYyI6MTAwMCwia2lkIjoia2V5I2RhdGFwYXNzd2QjMDAwMDEiLCJlbmMiOiJBMjU2R0NNIiwiYWxnIjoiUEJFUzItSFM1MTIrQTI1NktXIn0. TO_9a8K4V22W8_Vy-WbntnMOegigitHbYDZ47FAC3jWbVyg3YGo65g. jSfjXTrsVtdYR6Kh. _x0Xsq94leq9WKOhSUy0mBoJx8i9koTvQS- obFmHDM394tUB_zujCXFv0EwHSkmw80_t7BEmTU2gq0Trlw. OclzcreWD3WdFevMUs0bxA

3 Part II - Outgoing Encryption

3.1 Overview

The SCR result data can be accessed through GET operations of the resource *cases* (cf. SCR-Ident API Guide 2 Result).

Asymmetrical encryption is used for the result data in the response body. The result data will be encrypted with a public key provided by you. The key is an additional parameter in the HTTP header of the GET requests. The cipher is transmitted in JWE format. You can decrypt the received data with your private key.

The payload of your requests is secured by the HTTPS connection. There is no further encryption supported by the POSTIDENT system.

> **Unencrypted Result Data in Test Environment**
>
> During the integration of the SCR-Ident API the encryption can be configured as optional. So the http header fields "x-scr-key" and "x-scr-keyhash" can be omitted in your request. The response will not be encrypted.
> If the headers are sent, the result will be encrypted.
> In the productive environment the encryption is mandatory. It will be activated after a successful encryption test.

3.1.1 Preconditions

- During setup you should have received data password (required for keyHash) as the pre-shared-secret for the encryption.
- You have to create a RSA key pair, consisting of a public and a private key
  - a key size of 3 kbit is recommended (minimum 3 kbit)
  - you don't need a full X.509 digital certificate that is issued by a trusted CA. A simple key pair or a self signed certificate is sufficient.

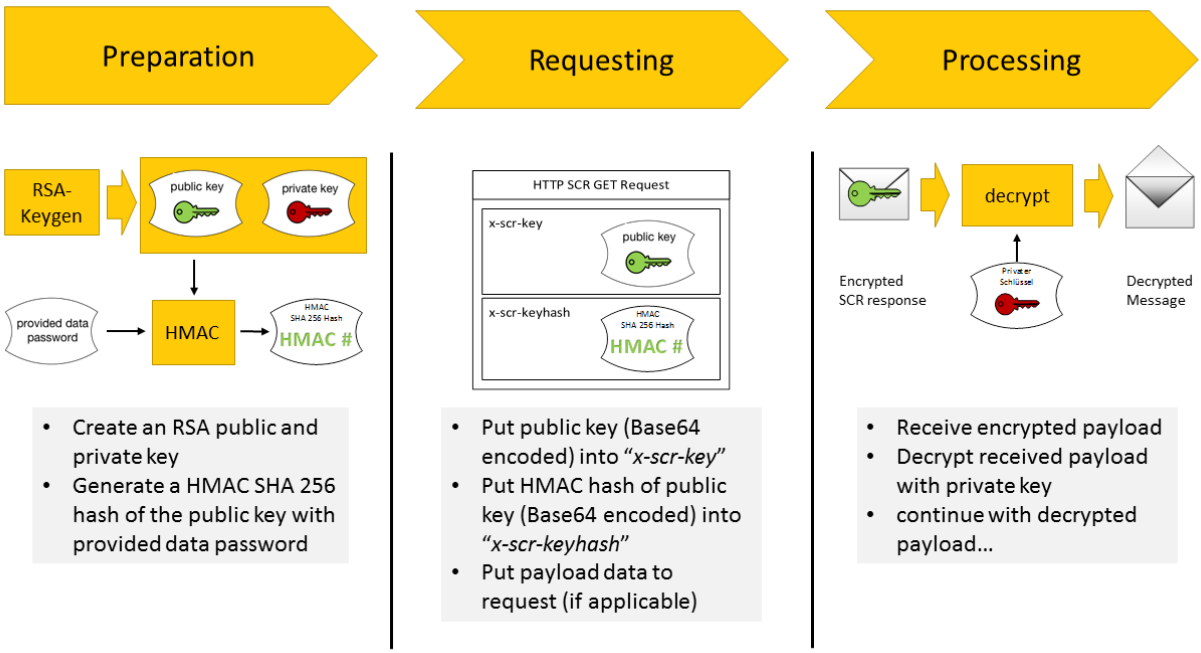
3.1.2 General Flow

The asymmetrical encryption works as follows:

- The public key must be passed in HTTP header field "x-scr-key"
- Postident system encrypts the response with given public key
- The encrypted response can only be decrypted with customer's hidden private key

In addition, the public key shall be encoded via HMAC-Hash in combination with provided data password and passed in http header field "x-scr-keyhash", in order to suspend Man-In-The-Middle attacks.

The following figure shows the public key encryption process:



3.2 Encryption Settings using header

3.2.1 HTTP-Header

The public key in http field "x-scr-key" and its HMAC-Hash in field "x-scr-keyhash" are mandatory.
Furthermore, the encryption algorithm and encryption can be chosen by using the optional http header fields "x-scr-alg" and "x-scr-enc". If you are using this option, it is up to you to ensure all requirements of security in accordance with RFC 7516.

Element	Mandatory	Description	Example
x-scr-key	yes	Contains the public key for content encryption with a size of 3072 or 4096 Bits. The value must be a base64 encoded string of the key encoded according to the ASN.1 type SubjectPublicKeyInfo which is defined in the X.509 standard (see RFC 5280).	`MIIBIjANBgkqhkiG9(...)FopeO2Z6TrwIDAQAB` For full length see "Sample Curl Request"

x-scr-keyhash	yes	Contains the Base64 encoded HMAC-Hash (HmacSHA256) of the public key. Use your POSTIDENT DataPassword to calculate the x-scr-hash.	`YAcCWwCyEyE6Fg0wuCgip3AjOk2mU/rU/UGuTW5O6p0=`
			
			`Used DataPassword: EAHqr_9NvCw2BuI23$a.0vRsS`

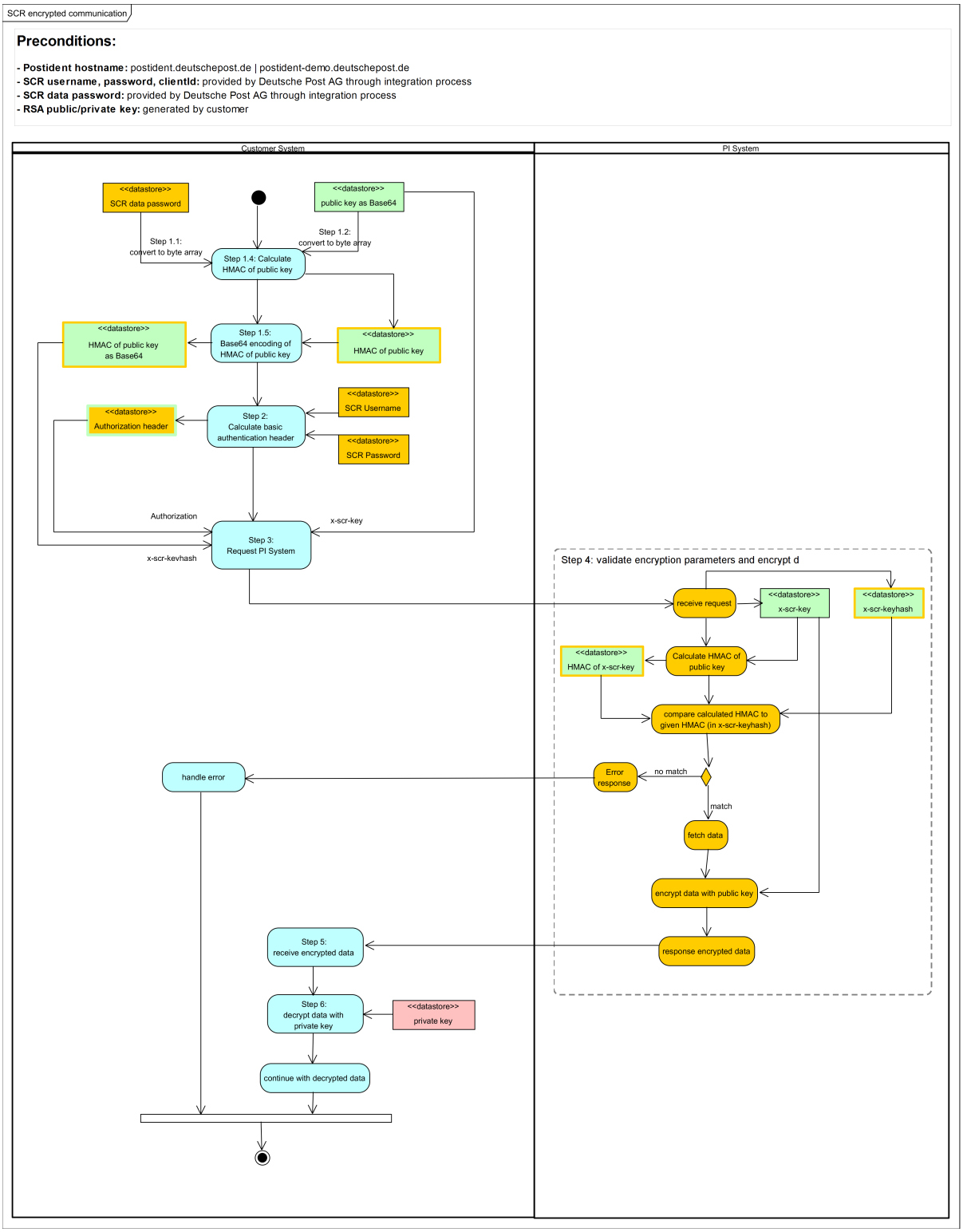
Element	Mandatory	Description	Example
x-scr-alg	no	Algorithm for result encryption To use an asymmetric RSA based public key encryption choose: • RSA-OAEP-256 (RSAES using Optimal Asymmetric Encryption Padding (OAEP) - RFC 3447 with the SHA-256 hash function and the MGF1 with SHA-256 mask generation function.	RSA-OAEP-256
x-scr-enc	no	Specify an AES encryption method for symmetric payload encryption. Available methods: • **A256CBC-HS512 (AES_256_CBC_HMAC_SHA_512 authenticated encryption using a 512 bit key (default value, recommended).** • A256GCM (AES in Galois/Counter Mode (GCM) (NIST.800-38D) using a 256 bit key. Not supported by PHP SecLib.)	A256CBC-HS512

3.2.2 Standard Encryption

By default, the following parameters are used:

Property	Value	Description
JWE algorithm of the response	**RSA-OAEP-256**	RSAES using Optimal Asymmetric Encryption Padding (OAEP) - RFC 3447 with the SHA-256 hash function and the MGF1 with SHA-256 mask generation function
JWE encryption of the response	**A256CBC-HS512**	AES_256_CBC_HMAC_SHA_512 authenticated encryption using a 512 bit key

3.3 Detailed Flow with Samples



3.3.1 Preconditions

Compile all the information required to execute the SCR GET request.

⚠️ Always take care for the specified way of conversion between string and byte. Doing the conversions in a different way will cause the postident system to reject the request.

#	Property	Value	Description
P1	Postident hostname	production: postident.deutschepost.de itu test environment: postident-itu.deutschepost.de	
P2	username	*Sample: SCRDEMO*	The clientid an the credentials for basic authentication will provided by the Deutsche Post technical sales or service team.
	password	*Sample: 3r#4Mu#GBRmP*	
	clientid	*Sample: 865E6E37*	
P3	data password	*Sample: xR7ea2_53S(m*	The data password will provided by the Deutsche Post technical sales or service team. This is the pre-shared-secret for HMAC calculation.
P4	auth	"Basic " + Base64.getEncoder().encodeToString((username + ":" + password).getBytes("UTF-8"));	build the basic auth header
P5	x-scr-key RSA public Key	public key in base64 form	Code Sample in section RSA key pair generation
	RSA private Key	private key in base64 form	
P6	x-scr-keyhash	base64(HmacSHA256 (base64decode(x-scr-key), data password.getByted(utf-8)))	Code Sample in section Calculate HMAC of public key calculate the hmac hash of the byte-value of RSA pubkey secured with the utf-8 bytes of data password string 🛑 Caution! Urgently take care, • to use the real bytes of RSA pubkey (base64decode(x-scr-key)) and • use the bytes of password string in UTF-8 encoding

3.3.2 RSA key pair generation

The keys can be generated before every call or stored in your system.
A key length of 3 kbit is recommended and is also the smallest accepted length.

Sample RSA 3072 key pair

```
======================================== RSA Public Key
========================================
pubKeyBase64:
MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAyF8XXjW+iZaYYH6l6i9wj5Of73Xom4CcdYAh
MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEA27wHDk8QQ2Jf0pWVRXCuB4WoWysx5unlNPHp
HLBRzEetu7/cZHURRPuVLf7ahF94H7P9smzVrkPqEwiciUvt+UwdTHqolWYZn0UB+FJ9HLLGFRSGOT0a
tIJTX8AfW3qFlJqTKPk2urr59nl5f9FayQjtu9YIU/Rpf/8Bxxnvxw/QUuOwJNYEnBoktur+PMc4DWrN
LapJ6f86luE96pBw4jEl+aSID0K+o6lezgVMuTJEpD0z56HQVs1O8IhwdF5P07WxtCy92V3WNz4B611v
UHy30JujpZQ4rvHsIYPAQD67unS4kga2gGVkmyyMQ7deTUsddOO1/xU6Czd1Fv6BXMA98wDTFtkfQm5P
oUciEtFOmjbY2AeF4kQQ9m9JHP7ToqfKD4nUBcqk7vWBcoAmRXX2E69VqkfjqSnY/2c4bX+M7Yy743xY
EFpNtWF/PnPXbdjoXmSmr0+fzaV+iH0+iWsegqEVVtJCOOa8XrvLwEUD6NN1uXu81ElZixs+5Co7AgMB
AAE=


  ======================================== RSA Private Key
========================================
privKeyBase64:
MIIG/QIBADANBgkqhkiG9w0BAQEFAASCBucwggbjAgEAAoIBgQDbvAcOTxBDYl/SlZVFcK4HhahbKzHm
6eU08ekcsFHMR627v9xkdRFE+5Ut/tqEX3gfs/2ybNWuQ+oTCJyJS+35TB1MeqiVZhmfRQH4Un0cssYV
FIY5PRq0glNfwB9beoWUmpMo+Ta6uvn2eXl/0VrJCO271ghT9Gl//wHHGe/HD9BS47Ak1gScGiS26v48
xzgNas0tqknp/zqW4T3qkHDiMSX5pIgPQr6jqV7OBUy5MkSkPTPnodBWzU7wiHB0Xk/TtbG0LL3ZXdY3
PgHrXW9QfLfQm6OllDiu8ewhg8BAPru6dLiSBraAZWSbLIxDt15NSx1047X/FToLN3UW/oFcwD3zANMW
2R9Cbk+hRyIS0U6aNtjYB4XiRBD2b0kc/tOip8oPidQFyqTu9YFygCZFdfYTr1WqR+OpKdj/Zzhtf4zt
jLvjfFgQWk21YX8+c9dt2OheZKavT5/NpX6IfT6Jax6CoRVW0kI45rxeu8vARQPo03W5e7zUSVmLGz7k
KjsCAwEAAQKCAYBSs93p/ksOOaf8MA2rQaJWtXEsw0C4ex8oQsas5BYdb3sN18QhUUiLMK+wzKO9P9uL
/yhE8B6qxOgEevDD4j6y3nmGkAIEnRWWsxfvOUjatNTGGV/9iiiTeRvdRYfn2+EMEfPCb64wMPCk2oqm
14q2cLRdxsyCoNLWeInYQMRVJXgQwp7I9n5srJBWAnNnN8o6jpCkF9xTBJuuLx+pOe0qzTnQUKNfLw1m
tvg2TZtXdvqL4Hc5PteV3BdSg9/jhJ0O9+LpPc6XPeyeX5W2wjcMlUjHqleZr25JQUGTpz2Xc/PyMF9a
ajjrFvI2JQaapCbbogX5xT1WBxgUIj0gSXattZE2YA+LYN0p3kWHALfjflJ/hEgXQlvThcc/i7NPuwyZ
9LmBUl0nnsUh+ZrBg+qb20d6hR801uUD9rOiIQCPqM+4MszPZEuPLGgw7XyDOrw8ridyQOuIOs/MkwW8
WUwGkBzTgfJhxNOcAjbFnEVvdqpV9brxcUai7HZpCyuOB5ECgcEA/3gQGwVJebtBW+ZBWFyP2XeJMAVm
3ouKmDcixAaPPkGO3WqjW2PuOcOjc3reL2qGNBdp15RuXhoQgzrriBhbD7+fLxFoEUh8r3r4Ee59fC4V
R+duZcAgzn7mGKL06Dmxn1pWRoIMi/oiFf4qASqKHixRG16Qre/qx/Aq148wwpVHW8aj3qi/O8WhMpyL
C0B9a67x9iZuZHpv4oECM/8m2VwKUQDTlTfgIfPT8x4uZm79FWoYNu4Sov0oqZelrKCLAoHBANww8zU2
bAtbUxHmcdL+Ep4A4GJ7IEslYxpCA6rE032h1JPjhsgkFCpCMm9ULuPB7GVO8FCi/G8np04k4tQCGAyk
owVo91F3/bHDzKg53eQNrlHHK+5v0Wig5FdrJYjzesTzXNE5vHGXIZlStLJvxydbNi6f+GKolUYrRiUD
q1rXblH/fzcH/OeoHTZMkditXGl04MeLZsVI1XyyabUO+Mbn1tCAbA/QGRwvxkrsUbMxhRwHZh9OpA/g
w+2Ktj+jEQKBwQCt1NKQsE5IyMQ2djzslT9EqHPu+x3+PFE4NfwwCo6ZvB4sR42UeDD3mUm1IFvjQ8bB
11Sr3lIjMfUedo8h8gC0fJ0id+OqLvq4pSH7gZ5dSzYr0SxrjApGuE51YRzoj6lvVxbxrzg+Xxumf2Wx
mXKWCW2/A9/Kmz/UgHWGa+av4MfHiSikL4rAbz+6oo5yP5WYGliGN6xSLoXHSCVpoCqdKGXUrI1MMdMP
B0ynV1gXV1paofCMH/8KbNyoWYngJvUCgcAbg25QXt1/SckGepIMeDzhS2C+TX5KhYBtnoQQyYlXsGn8
lLiZt5Bhe2YL7Svyv5+HRSbYHJWIWh8Yr5k8Poq8wbGxEl2c625i0QF4n8pvHZqDk2yU3ZoABeKiXbwR
8cKMa5BUgG9gji7flqsHhNoQu9/4UXra8Q+99dhM9blqn3KkglVmDfR3CdWknEq5FWhvsPAJXwzXZ/Nt
Imn8+sLxL7Ty1qkqDKmmkP2pDQBkcLKePvpuvQfZCA/TVbrSFtECgcAUga7lVcG4j2z6UXl5DG/68YYG
REN+8ZfD3SmZn8rrAP8QuhgFHPe3u2ROCMjwWxYAgQ9Yf8uUKapZ/5O/fKcrf1ET/G26gsCxxj6i4PRq
ch6akyXuVlyR7WbGvqd89N3sdTS7jO/K/p8JB0GKwS2cOUdRsv7GdXBIcWl3ZL8ls7WWNLmucc4kk34G
LOSGzJbr4YY8d0DDbol+eUuIhbXZqqkKyb08x8N/86ytYQNVeOMO2qTpN4lVAF03i0qDAdA=
```

Code sample RSA keypair generation

This code is kept simple in the interest of easily comprehensible tests. Error handling and in memory storage of the private key should be improved in production use.

Keygen Step	Description	java Snippet	Data
1	instantiate and initialize keypair-generator	java.security.KeyPairGenerator keyGen = java.security.KeyPairGenerator.*getInstance*("RSA"); keyGen.initialize(3072);	
2	generate keypair	java.security.KeyPair keypair = keyGen.genKeyPair();	
4	get public key in base64 form	String pubKeyBase64 = Base64.encodeBase64String(keypair.getPublic().getEncoded());	see codeblock above
6	get private key in base64 form	String privKeyBase64 = Base64.encodeBase64String(keypair.getPrivate().getEncoded());	see codeblock above

initializeKeypair

```
    /**
     * generate the RSA key pair. the key pair and the base64 representations of the
     * public and private key are stored in static class variables of JUnit testclass
     */
    public static void inititalizeKeypair() throws NoSuchAlgorithmException {
        // use KeyPairGenerater to generate RSA keypair
        java.security.KeyPairGenerator keyGen = java.security.KeyPairGenerator.getInstance("RSA");
        keyGen.initialize(key_length);
        // generate keypair
        keypair = keyGen.genKeyPair();
        // store keys in base64 format
        pubkey_base64 = Base64.encodeBase64String(keypair.getPublic().getEncoded());
        privkey_base64 = Base64.encodeBase64String(keypair.getPrivate().getEncoded());
        out("pubkey:  " + pubkey_base64);
        out("privkey: " + privkey_base64);
    }
```

3.3.3 Calculate HMAC of public key

Calcuclate an HMAC of your private key as bytearray with the SCR data passwort as secret.

Code Sample for HMAC calculation

IN: dataPassword(precondition #3), publicKey(precondition #4)

OUT: base64 encoded HMAC hash

HMAC Step	Description	java Snippet	sample Data
1	convert datapassword to byte[]	byte[] dataPasswordBytes = dataPassword.getBytes("UTF-8");	dataPasswordBytes = 78 52 37 65 61 32 5F 35 33 53 28 6D
2	convert RSA public key to byte[]	byte[] publicKeyBytes = Base64.getDecoder().decode(publicKey Base64);	publicKeyBytes = 30 82 01 A2 30 ... 02 03 01 00 01
3	create and initialize javax.crypto.Mac	SecretKeySpec signingKey = new SecretKeySpec(dataPasswordBytes, "HmacSHA256"); Mac mac = Mac.getInstance("HmacSHA256"); mac.init(signingKey);	
4	calculate HMAC hash bytes	mac.update(publicKeyBytes); byte[] hmacHashBytes = mac.doFinal();	hmacHashBytes = 09 F6 1B E1 8F 4D 1F DD 6E 19 31 80 3D EF 9A 24 B4 8F 7F CD C2 99 F9 1E 5C 8F 14 D0 E8 4B 1E 02
5	convert hmac bytes to base64 form	String hmacHashBase64 = Base64.getEncoder().encodeToString(hashBytes);	hmacHashBase64 = CfYb4Y9NH91uGTGAPe+a JLSPf83CmfkeXI8U0OhL HgI=

hmacHashOverKey

```
1         /**
2         * Calculates sha256 hmac over an base64 encoded payload.
3         * SCR flow step 1: Calculate HMAC of public key
4         *
5         * @param dataPassword
6         *           HMAC secret - will be converted in the utf8 byte
representation.
7         * @param publicKeyBase64
8         *           Base64 encoded payload - will be decoded to bytearray befor
hashing
9         * @return der Base64 encoded HMAC hashes
10        * @throws UnsupportedEncodingException
11        * @throws NoSuchAlgorithmException
12        * @throws InvalidKeyException
13        */
14        public static String hmacHashOverKey(String dataPassword, String
publicKeyBase64)
15                throws UnsupportedEncodingException, NoSuchAlgorithmException,
InvalidKeyException {
16          String hmacHashBase64 = "";
17          // HMAC Step 1:  convert datapassword to byte[]
```

```
            byte[] dataPasswordBytes = dataPassword.getBytes("UTF-8");
            // HMAC Step 2:  convert RSA public key to byte[]
            byte[] publicKeyBytes = Base64.getDecoder().decode(publicKeyBase64);
             // HMAC Step 3:  create and initialize javax.crypto.Mac
             // i). create HmacSha secretKey from Datapassword
            SecretKeySpec hmacKey = new SecretKeySpec(dataPasswordBytes,
HMAC_SHA256_ALGORITHM);
            // ii) instantiate and initialize mac
            Mac mac = Mac.getInstance(HMAC_SHA256_ALGORITHM);
            mac.init(hmacKey);
            // HMAC Step 4: calculate HMAC hash bytes
            mac.update(publicKeyBytes);
            byte[] hmacHashBytes = mac.doFinal();
            // HMAC Step 5: convert hmac bytes to base64 form
            hmacHashBase64 = Base64.getEncoder().encodeToString(hmacHashBytes);
            return hmacHashBase64;
        }
```

3.3.4 Handle the get Request in 3 JUnit Tests

For testing the SCR client connection, 3 test cases with increasing complexity are used below. This makes it easier to localize potential problems.
Test case 1 *testGetCasesUnencrypted* Success means:
- HTTPS connection possible
- one of the required TLS is supported by the client
- Username, password, authorisation HTTP header and clientid are correct

Test case 2 *testGetCasesEncrypted*
Success means:
- all of Test case 1
- succesful key pair generation in required strength
- the Encryption HTTP headers were set successfully
- the Postident server has delivered an encrypted response

Test case 3 *testGetCasesEncryptedWithDecrypt*
Success means:
- all of Test case 1 & 2
- successful decryption of Response with the generated private key

3.3.5 Test case 1 Processing of the HTTP request without encryption

Meaning: check access to the ITU SCR api with unencrypted Result. (In production environment this call is not supported - you will get an Error 90101: *Encryption is obligatory*)

Entry point: JUnit Test ScrCallTests.testGetCasesUnencrypted

Required Date for the Request

```
    /**
     * Processing of an SCR get request (unencrypted answer)
     *
     * The result is unencrypted because the headers x-scr-key and x-scr-keyhash
are
     * not be set. Note: The production environment suppresses unencrypted result
     * querys
     */
```

```
    @Test
    void testGetCasesUnencrypted() {
        out("JUNIT Test testGetCasesUnencrypted");
        String ret = ScrHttpRequestHandler.callScr(scr_url_full_all, scr_user,
scr_password);
        out(ret);
        assertThatNoException();
        assertTrue(ret.startsWith("["));
    }
```

The data required for the SCR getCase request are stored as constants in the JUnit test class ScrCallTests.

Required Data for the Request

```
    /* keyLength of the RSA key pair used in the test ( 3072 or 4096 Bit) */
    static int key_length = 3072;
    /* Username for Basic Auth */
    static String scr_user = "<your username>";
    /* Password for Basic Auth */
    static String scr_password = "<your password>";
    /* Data password for HMAC calculation */
    static String scr_datapassword = "<your datapassword>";
    /* clientid, used as request parameter */
    static String scr_clientid = "<your clientid>";
    /*
     * Host of the SCR endpoint postident-itu.deutschepost.de (test environment) or
     * postident.deutschepost.de (productive system)
     */
    static String scr_host = "postident-itu.deutschepost.de";
    /* URL for the SCR GET request getting all cases for clientid */
    static String scr_url_full_all = "https://" + scr_host + "/api/scr/v1/" +
scr_clientid
            + "/cases/full";//?inProgress=true&new=true&closed=true";
```

The Processing of the HTTP Request is done by calling ScrRequestHandler

Calculate the required Parameter

```
String ret = ScrHttpRequestHandler.callScr(scr_url_full_all, scr_user,
scr_password);
```

For details in processing the request please see implementation of ScrRequestHandler.

3.3.6 Test case 2 Processing of the HTTP request with encryption Parameters (no decryption on client side)

Required Date for the Request

```
    /**
     * Processing of an SCR get request (encrypted response without decryption)
     *
     * The result is encrypted because the headers x-scr-key and x-scr-keyhash are
     * set. no decryption takes place in this test, the first ones Characters of
the
     * encrypted response are output on the console
     */
    @Test
    void testGetCasesEncrypted() throws ParseException, JOSEException, ScrException
{
        out("JUNIT Test testGetCasesEncrypted");
        String ret = ScrHttpRequestHandler.callScr(scr_url_full_all, scr_user,
scr_password, pubkey_base64, hmac_hash);
        out(ret.substring(0, 80) + " ...");
        assertThatNoException();
        assertTrue(ret.startsWith("eyJlbmMi"));// base64 representation of
'{"enc":"'
    }

```

The parameters required for the get request with encrypted answer are prepared in the @BeforeAll initializeAll() method.

Calculate the required Parameter

```
hmac_hash = ScrEncryptionHandler.hmacHashOverKey(scr_datapassword, pubkey_base64);
// use KeyPairGenerater to generate RSA keypair
java.security.KeyPairGenerator keyGen =
java.security.KeyPairGenerator.getInstance("RSA");
keyGen.initialize(key_length);
// generate keypair
keypair = keyGen.genKeyPair();
// store keys in base64 format
pubkey_base64 = Base64.encodeBase64String(keypair.getPublic().getEncoded());
privkey_base64 = Base64.encodeBase64String(keypair.getPrivate().getEncoded());
```

3.3.7 Test case 3 Processing of the HTTP request with encryption Parameters (with decryption on client side)

Required Date for the Request

```
    /**
     * Processing of an SCR get request (encrypted response with decryption)
     *
     * The result is encrypted because the headers x-scr-key and x-scr-keyhash are
     * set. The decrypted payload of the Encrypted Response is sent to Console
     * output
     */
    @Test
    void testGetCasesEncryptedWithDecrypt() throws ParseException, JOSEException,
ScrException {
        out("JUNIT Test testGetCasesEncryptedWithDecrypt");
        String ret = ScrHttpRequestHandler.callScr(scr_url_full_all, scr_user,
scr_password, pubkey_base64, hmac_hash);
        out(ret);//out(ret.substring(0, 80) + " ...");
        String decrypted = ScrEncryptionHandler.decryptPayload(ret,
privkey_base64);
        assertThatNoException();
        assertTrue(ret.startsWith("eyJlbmMi"));// base64 representation of
'{"enc":"'
        assertTrue(decrypted.startsWith("["));// begin of json array
        out(decrypted);
    }


```

3.3.8 Sample Response of SCR getCases full:

This json data corresponds to the scr specification April 2023


```
[
    {
        "caseId": "K6JNXGBG2XVU",
        "caseStatus": {
            "status": "closed",
            "archived": false,
            "validUntil": "2023-03-23T14:02:04+01:00",
            "created": "2023-03-09T14:02:04+01:00",
            "modified": "2023-03-09T15:11:39+01:00"
        },
        "orderData": {
            "customData": {},
            "processData": {
                "targetCountry": "DEU",
                "preferredLanguage": "DE_DE",
                "referenceId": "K6JNXGBG2XVU",
                "callbackUrlCouponRequested": {},
                "callbackUrlReviewPending": {},
                "callbackUrlIncomplete": {},
```

```
                "callbackUrlSuccess": {},
                "callbackUrlDeclined": {}
            },
            "contactDataProvided": {},
            "identificationDocumentProvided": {},
            "drivingLicenceProvided": {
                "drivingLicenceClasses": []
            }
        },
        "contactData": {
            "title": {},
            "firstName": {
                "status": "new",
                "value": "Matthias"
            },
            "lastName": {
                "status": "new",
                "value": "Schulz"
            },
            "mobilePhone": {},
            "email": {
                "status": "new",
                "value": "terter.alpha@email.de"
            },
            "epost": {},
            "address": {
                "streetAddress": {},
                "appendix": {},
                "postalCode": {},
                "city": {},
                "country": {}
            }
        },
        "identifications": [
            {
                "identificationMethod": "basic",
                "identificationStatus": {
                    "status": "success",
                    "identificationTime": "2023-03-09T15:11:38+01:00",
                    "created": "2023-03-09T14:02:19+01:00",
                    "modified": "2023-03-09T15:11:39+01:00"
                },
                "identificationDocument": {
                    "type": {
                        "status": "new",
                        "value": "1"
                    },
                    "number": {
                        "status": "new",
                        "value": "sdadasdsa"
                    },
                    "firstName": {
                        "status": "new",
                        "value": "Matthias"
                    },
                    "lastName": {
                        "status": "new",
                        "value": "Mustermann"
                    },
```

```
                    "birthName": {},
                    "birthDate": {
                        "status": "new",
                        "value": "1977-01-11"
                    },
                    "birthPlace": {
                        "status": "new",
                        "value": "Paddington"
                    },
                    "nationality": {
                        "status": "new",
                        "value": "DEU"
                    },
                    "address": {
                        "streetAddress": {
                            "status": "new",
                            "value": "Sackgasse 1"
                        },
                        "appendix": {},
                        "postalCode": {
                            "status": "new",
                            "value": "12345"
                        },
                        "city": {
                            "status": "new",
                            "value": "Rostock"
                        },
                        "country": {
                            "status": "new",
                            "value": "DEU"
                        }
                    },
                    "dateIssued": {
                        "status": "new",
                        "value": "2020-11-11"
                    },
                    "dateOfExpiry": {
                        "status": "new",
                        "value": "2027-11-12"
                    },
                    "authority": {
                        "status": "new",
                        "value": "Sas"
                    },
                    "placeOfIssue": {},
                    "countryOfDocument": {
                        "status": "new",
                        "value": "DEU"
                    },
                    "records": [
                        {
                            "recordId": "935254843",
                            "fileName": "K6JNXGBG2XVU_idimage_1.jpg",
                            "belongsTo": "identificationdocument",
                            "type": "idimage",
                            "mimeType": "image/jpeg"
                        }
                    ]
                },
```

```
                "records": [
                    {
                        "recordId": "935254842",
                        "fileName": "K6JNXGBG2XVU_usersignature.jpg",
                        "belongsTo": "method",
                        "type": "usersignature",
                        "mimeType": "image/jpeg"
                    }
                ],
                "additionalDataBasic": {
                    "couponDownloadCount": 1,
                    "couponDownloadLastTimestamp": "2023-03-09T14:02:20+01:00",
                    "postOfficeStreetAddress": "Platz der Deutschen Post",
                    "postOfficeCity": "Bonn"
                }
            }
        ],
        "accountingData": {
            "accountingNumber": "506606638437A2",
            "accountingProduct": "Postident Basic Zusatzprodukt 2"
        }
    }
]
```

3.4 Errors

In error situations the SCR API will return HTTP 4xx status codes and a detailed error description in the body.

3.4.1 Sample Error Messages

```
// 400: Base64 format error in keyhash
{"apiversion":"v1","errors":[{"errorcode":"90104","reason":"base64 error","key":"x-scr-keyhash","message":"Base64 format error."}]}

// 400: Base64 format error in key Error:
400 {"apiversion":"v1","errors":[{"errorcode":"90104","reason":"base64 error","key":"x-scr-key","message":"Base64 format error."}]}

// 400: No keyhash provided
{"apiversion":"v1","errors":[{"errorcode":"90106","reason":"missing keyhash","key":"x-scr-keyhash","message":"No keyhash value provided in header x-scr-keyhash."}]}

// 400: wrong keyhash
{"apiversion":"v1","errors":[{"errorcode":"90107","reason":"hash failure","key":"","message":"Provided encryption key does not match keyhash. Possible reasons: wrong data password or x-scr-key has been manipulated."}]}

// 400: invalid RSA public Key
{"apiversion":"v1","errors":[{"errorcode":"90109","reason":"invalid key","key":"x-scr-key","message":"Invalid RSAPublicKey format for encryption key."}]}

// 401: invalid credentials
{"apiversion":"v1","errors":[{"errorcode":"90114","reason":"unauthorized","key":"Authorization","message":"Authorization failed."}]}

```

```
// 403: wrong clientid
{"apiversion":"v1","errors":[{"errorcode":"90127","reason":"forbidden","key":"Autho
rization","message":"User has insufficient rights."}]}
```

3.4.2 Encryption Errors

Here is an overview of the possible errors within encryption:

HTTP Status	Errorcode	Reason	Key	Message
400	90101	encryption is obligatory		Unencrypted responses are not allowed. Provide encryption key and keyhash in the header fields x-scr-key and x-scr-keyhash to receive encrypted responses.
400	90102	algorithm not supported	x-scr-alg	SCR does not support ALG {0}
400	90103	encryption not supported	x-scr-enc	SCR does not support ENC {0}
400	90104	base64 error	result encryption	Base64 format error.
400	90105	wrong key size	x-scr-key	The provided encryption key does not match the requirements of ALG:{0}, ENC:{1}, Bits provided:{2}, Bits required:{3}
400	90106	missing keyhash	x-scr-keyhash	No keyhash value provided in header x-scr-keyhash.
400	90107	hash failure		Provided encryption key does not match keyhash. Possible reasons: wrong data password or x-scr-key has been manipulated
400	90108	encryption error		Unexpected encryption error .
400	90109	invalid key	x-scr-key	Invalid RSAPublicKey format for encryption key.
400	90110	missing key	x-scr-key	No encryption key provided in header x-scr-key.
401	90114	unauthorized	Authorization	Authorization failed.

3.4.3 Typical error situations and error messages

The following error situations are typical during the integration process when implementing encryption.

Encryption is mandatory	header x-scr-key is used	header x-scr-keyhash is used	keyhash matches key	Response from postident system
Yes	No	No	-	http status: 400 errorcode: 90101
Yes	Yes	No	-	http status: 400 errorcode: 90106
Yes	No	Yes	-	http status: 400 errorcode: 90110

Encryption is mandatory	header x-scr-key is used	header x-scr-keyhash is used	keyhash matches key	Response from postident system
Yes	Yes	Yes	No	http status: 400 errorcode: 90107
Yes	Yes	Yes	Yes	Response body is encrypted
No (only in test environment)	No	No	-	Response body is clear text
No (only in test environment)	Yes	No	-	http status: 400 errorcode: 90106
No (only in test environment)	No	Yes	-	http status: 400 errorcode: 90110
No (only in test environment)	Yes	Yes	No	http status: 400 errorcode: 90107
No (only in test environment)	Yes	Yes	Yes	Response body is encrypted

3.5 Code Samples

3.5.1 Disclaimer

Disclaimer

The following Code Samples are not intended for productive usage, but as a coding reference to support the implementation process of the connection to scr service.
Therefore the command line output of the ScrClientTool refers to the steps in paragraph 3 detailed flow with samples.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.

IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

3.5.2 Import the provided Eclipse Project

- Download and expand the provided scr-demo-client.zip
- Choose in Eclipse: File / open Projects from File System ...
- Navigate to the Folder with the expanded scr-demo-client
- By clieck on finish scr-demo-client will be imported as maven Project

3.5.3 Java SCR Sample JUnit Tests

The following code provides an Set of JUnit Tests to show service consumery in connection to scr service. It contains

- generate RSA Keypairs
- calculate HMAC Hash on RSA Public Keys

- process scr requests
- decrypt SCR results

java source

This Source intends to give a simple example of calling the scr service.

`The code follows the structured approach, in order to simplify the procedural view of the call processing and, on the other hand, to make the porting into other languages easy`

provided classes:

- ScrCaller, the HTTP request processor
- ScrCryptoHelper, which collects cryptographic functions to prepare scr requests and to decrypt scr responses.
- junit Tests explained before

public class ScrCaller

pure HTTP request handling

ScrCaller

```
1  package de.deutschepost.postident.scrClient;
2  import java.io.BufferedReader;
3  import java.io.IOException;
4  import java.io.InputStream;
5  import java.io.InputStreamReader;
6  import java.net.ConnectException;
7  import java.net.HttpURLConnection;
8  import java.net.SocketTimeoutException;
9  import java.net.URL;
10 import java.net.UnknownHostException;
11 import java.text.MessageFormat;
12 /**
13  * SCR Request Handler
14  *
15  * methods to handle SCR HTTP requests
16  *
17  * @author Deutsche Post AG
18  * @version 1.0
19  */
20 public class ScrCaller {
21    /**
22     * calls scr service and returns the service response as string
23     *
24     * @param scrUrl
25     *           url to SCR Service e.g. https://postident.deutschepost.de/api/scr/
   v1/865E6E37/cases
26     * @param authString for BASIC authentication
27     * @param rsaPubkey
28     *           rsa public key in base64 form
29     * @param rsaKeyhash
30     *           hmac hash over rsa pubkey in base64 form
31     * @return the SCR Response
32     */
33    public String callScr(String scrUrl, String authString,  String rsaPubkey,
   String rsaKeyhash) {
```

```
34          int responseCode = -1;
35          StringBuilder sbRet = new StringBuilder();
36          if (scrUrl == null || scrUrl.isEmpty()) {
37             out("Error: URL not scecified.");
38             sbRet.append("Error: URL not scecified.");
39             return sbRet.toString();
40          }
41          try {
42             // SCR flow step 3.2 instantiate anf configure URL connection
43             URL url = new URL(scrUrl);
44             HttpURLConnection huc = (HttpURLConnection) url.openConnection();
45             huc.setRequestMethod("GET");
46             huc.setConnectTimeout(30000);
47             huc.setReadTimeout(30000);
48             huc.setRequestProperty("User-agent", "SCR-CLIENT");
49             huc.setRequestProperty("Content-Type", "application/json");
50             out("Info: " + "HEADER User-agent: SCR-CLIENT ");
51             // SCR flow step 3.3 set Authorization Header
52             huc.setRequestProperty("Authorization", authString);
53             out("SCR flow 3.3: HEADER Authorization: " + authString);
54             huc.setDoOutput(false);
55             huc.setDoInput(true);
56             if (rsaPubkey != null) {
57                huc.setRequestProperty("x-scr-key", rsaPubkey);
58                out("SCR flow 3.4: HEADER x-scr-key: " + rsaPubkey);
59             }
60             if (rsaKeyhash != null) {
61                huc.setRequestProperty("x-scr-keyhash", rsaKeyhash);
62                out("SCR flow 3.5:  x-scr-keyhash: " + rsaKeyhash);
63             }
64             out("Info: " + "HEADER Content-Type: application/json");
65             // SCR flow step #3 send the http GET Request to Postident System
66             responseCode = huc.getResponseCode();
67             String encryptedPayload = "";
68             if (responseCode == 200) {
69                encryptedPayload = readInputStream(huc.getInputStream(), true);
70                sbRet.append(encryptedPayload);
71             } else {// Fehlerfall
72                encryptedPayload = readInputStream(huc.getErrorStream(), true);
73                sbRet.append("Error:  " + responseCode + " " + encryptedPayload);
74             }
75             out("SCR flow 5: " + "HTTP Response: " + responseCode + " Payload: " +
   encryptedPayload);
76             return sbRet.toString();
77          } catch (SocketTimeoutException e) { // NOSONAR squid:S1166 SocketTimeout
78             String msg = "Socket Timeout Exception. " + e.getMessage();
79             out("Error:  " + msg);
80             responseCode = -4;
81             sbRet.append(msg);
82          } catch (UnknownHostException e) { // NOSONAR squid:S1166
   UnknownHostException
83             String msg = "UnknownHostException. " + e.getMessage();
84             out("Error:  " + msg);
85             responseCode = -5;
86             sbRet.append(msg);
87          } catch (ConnectException e) { // NOSONAR squid:S1166 ConnectException ist
88                                         // eindeutig - Stacktrace sinnlos
89             String msg = "ConnectException. " + e.getMessage();
90             out("Error:  " + msg, e);
```

```
91          responseCode = -6;
92          sbRet.append(msg);
93       } catch (javax.net.ssl.SSLHandshakeException e) { // NOSONAR squid:S1166
94                                                         // fuer
95                                                         // SSLHandshakeException
96                                                         // ist der
97                                                         // Stacktrace-Inhalt
98                                                         // ohne Belang
99          String msg = "SSLHandshakeException beim Senden. " + e.getMessage();
100         out("Error:  " + msg, e);
101         responseCode = -7;
102         sbRet.append(msg);
103      } catch (Throwable e) { // NOSONAR
104                              //
    checkstyle:com.puppycrawl.tools.checkstyle.checks.coding.IllegalCatchCheck
105                             // 3rdParty (HTTP) Calls mit diversen
106                             // Exceptionfaellen
107         String msg = "Error during scr request. " + e.toString();
108         out("Error:  " + msg);
109         responseCode = -8;
110         sbRet.append(msg + " " + e.getMessage());
111      }
112      out("Info: " + "scr response: " + sbRet.toString());
113      return sbRet.toString();
114   }
115
116   /**
117    * reads out http response stream.
118    *
119    * @param istr
120    *           stream to read
121    * @param supressException
122    *           in case of error an empty string will returned
123    * @return stream content as sting
124    * @throws IOException
125    * @throws IllegalArgumentException
126    */
127   public static String readInputStream(InputStream istr, boolean supressException)
    throws IOException {
128      StringBuilder sb = new StringBuilder();
129      if (istr == null && supressException) {
130         return "";
131      }
132      if (istr == null) {
133         throw new IllegalArgumentException("istr must not be null");
134      }
135      BufferedReader in = null;
136      try {
137         in = new BufferedReader(new InputStreamReader(istr, "UTF-8"));
138         String row = "";
139         while ((row = in.readLine()) != null) {
140            sb.append(row);
141         }
142         in.close();
143      } catch (IOException e) {
144         out("Error: readInputStream() Fehler beim Lesen eines HTTP Streams. {0}",
    e);
145         if (!supressException) {
146            throw e;
```

```
147            }
148         } finally {
149            if (in != null) {
150               in.close();
151            }
152         }
153         return sb.toString();
154      }
155      /**
156       * Call scr without encryption.
157       * calls {@link #callScr(String, String, String, String, String)} with emphy key
    and empty keyhash.
158       *
159       * @param scrUrl
160       * @param authString
161       * @return
162       */
163      public String callScr(String scrUrl, String authString) {
164         return callScr(scrUrl, authString, "", "");
165      }
166      /**
167       * console output with parameter substitution
168       *
169       * @param message
170       * @param args
171       */
172      public static void out(String message, Object... args) {
173         if (args.length == 0) {
174            System.out.println(message);
175         } else {
176            System.out.println(MessageFormat.format(message, args));
177         }
178      }
179   }
```

public class ScrCryptoHelper

cryptographic methods required for scr call handling

ScrCryptoHelper

```
1   package de.deutschepost.postident.scrClient;
2   import java.io.UnsupportedEncodingException;
3   import java.security.InvalidKeyException;
4   import java.security.KeyFactory;
5   import java.security.NoSuchAlgorithmException;
6   import java.security.interfaces.RSAPrivateKey;
7   import java.security.spec.InvalidKeySpecException;
8   import java.security.spec.PKCS8EncodedKeySpec;
9   import java.text.MessageFormat;
10  import java.text.ParseException;
11  import java.util.Base64;
12  import javax.crypto.Mac;
13  import javax.crypto.spec.SecretKeySpec;
14  import javax.xml.bind.DatatypeConverter;
15  import com.nimbusds.jose.JOSEException;
```

```java
import com.nimbusds.jose.JWEObject;
import com.nimbusds.jose.Payload;
import com.nimbusds.jose.crypto.RSADecrypter;
/**
 * ScrEncryptionHelper cryptographic methods required for scr call handling
 *
 * @author Deutsche Post AG
 * @version 1.0
 */
public class ScrCryptoHelper {
   /** HMAC Hash Algorithm to use. */
   private static final String HMAC_SHA256_ALGORITHM = "HmacSHA256";
   /**
    * SCR flow step 6: decrypt data with private key.
    *
    * @param encryptedPayload
    *           data to decrypt.
    * @param rsaPrivateKeyBase64
    *           base64 encoded rsy private key, used to decrypt payload
    * @return decrypted payload as string
    * @throws ParseException
    * @throws JOSEException
    * @throws ScrException
    */
   public static String decryptPayload(String encryptedPayload, String
rsaPrivateKeyBase64)
         throws ParseException, JOSEException, ScrException {
      // SCR flow step 6.1: parse encrypted payload into JWEObjec
      JWEObject jweObject = JWEObject.parse(encryptedPayload);
      out("SCR flow 6.1: parsed JWE header" +
jweObject.getHeader().toJSONObject().toJSONString());
      try {
         // SCR flow step 6.2 and 6.3: {@link #createRSADecrypter(String) }
         // SCR flow step 6.4: decrypt JWEObject
         jweObject.decrypt(createRSADecrypter(rsaPrivateKeyBase64));
      } catch (JOSEException e) {
         if (e.getMessage().contains("Illegal key size")) {
            out("Error: "
                  + "Illegal key size. Maybe Java Cryptography Extension (JCE) is
not installed. See http://www.oracle.com/technetwork/java/javase/downloads/jce-7-
download-432124.html {0}",
                  e);
         } else {
            out("Error during decryprPayload. {0}", e);
         }
      }
      // SCR flow step 6.5: get decrypted responsestring
      Payload payload = jweObject.getPayload();
      return payload.toString();
   }
   /**
    * Instatiates an RSA decrypter from rsaprivate key in bas64 form.
    * SCR flow 6.3: instantiate RSADecrypter
    * @param rsaPrivateKey
    * @return
    * @throws ScrException
    */
   public static RSADecrypter createRSADecrypter(String rsaPrivateKey) throws
 ScrException {
```

```
70        RSADecrypter ret;
71        ret = new RSADecrypter(base64ToPrivateKey(rsaPrivateKey));
72        return ret;
73     }
74     /**
75      * Converts a Base64 String to RSAPrivateKey.
76      * SCR flow 6.2: instantiate RSAPrivateKey
77      *
78      * @param base64Bytes
79      * @return the RsaPrivateKey
80      * @throws ScrException
81      */
82     public static RSAPrivateKey base64ToPrivateKey(String base64Bytes) throws
 ScrException {
83        byte[] keybytes;
84        try {
85           keybytes = Base64.getDecoder().decode(base64Bytes);
86        } catch (Throwable e) { // NOSONAR
87           //
checkstyle:com.puppycrawl.tools.checkstyle.checks.coding.IllegalCatchCheck
88           // 3rdParty Calls
89           out("Warning: " + "PrivateKey creation error. Base64 conversion error.
{0}", e);
90           throw new ScrException("PrivateKey creation error. Base64 conversion
error.", e);
91        }
92        RSAPrivateKey retKey;
93        try {
94           retKey = byteToPrivateKey(keybytes);
95        } catch (InvalidKeySpecException | NoSuchAlgorithmException e) {
96           out("Warning: " + "PrivateKey creation error {0}", e);
97           throw new ScrException("PrivateKey creation error.", e);
98        }
99        return retKey;
100    }
101    /**
102     * Convert a ByteArray to RSAPrivateKey.
103     * Part of SCR flow 6.2: instantiate RSAPrivateKey
104     *
105     * @param keybytes
106     *           Bytes of Key.
107     * @return the RsaPublicKey
108     * @throws InvalidKeySpecException
109     * @throws NoSuchAlgorithmException
110     */
111    public static RSAPrivateKey byteToPrivateKey(byte[] keybytes)
112          throws InvalidKeySpecException, NoSuchAlgorithmException {
113       return (RSAPrivateKey) KeyFactory.getInstance("RSA").generatePrivate(new
 PKCS8EncodedKeySpec(keybytes));
114    }
115    /**
116     * Calculates sha256 hmac over an base64 encoded payload.
117     * SCR flow step 1: Calculate HMAC of public key
118     *
119     * @param dataPassword
120     *           HMAC secret - will be converted in teh utf8 byte representation.
121     * @param publicKeyBase64
122     *           Base64 encoded payload - will be decoded to bytearray befor hashing
123     * @return der Base64 encoded HMAC hashes
```

```
124         * @throws UnsupportedEncodingException
125         * @throws NoSuchAlgorithmException
126         * @throws InvalidKeyException
127         */
128        public static String hmacHashOverKey(String dataPassword, String
    publicKeyBase64)
129                throws UnsupportedEncodingException, NoSuchAlgorithmException,
    InvalidKeyException {
130              String hmacHashBase64 = "";
131              // SCR flow step 1.1: convert datapassword to byte[]
132              byte[] dataPasswordBytes = dataPassword.getBytes("UTF-8");
133              out("SCR flow 1.1: dataPassword Bytes (.getBytes(\"UTF-8\")): " +
    toHexString(dataPasswordBytes));
134              // SCR flow step 1.2: convert RSA public key to byte[]
135              byte[] publicKeyBytes = Base64.getDecoder().decode(publicKeyBase64);
136              System.out.println("SCR flow 1.2: public Key Bytes (.getBytes(\"UTF-8\")): "
     + toHexString(publicKeyBytes));
137               // SCR flow step 1.3:  create and initialize javax.crypto.Mac
138               // i). create HmacSha secretKey from Datapassword
139              SecretKeySpec signingKey = new SecretKeySpec(dataPasswordBytes,
    HMAC_SHA256_ALGORITHM);
140              // ii) instantiate and initialize mac
141              Mac mac = Mac.getInstance(HMAC_SHA256_ALGORITHM);
142              mac.init(signingKey);
143              // SCR flow step 1.4: calculate HMAC hash bytes
144              mac.update(publicKeyBytes);
145              byte[] hmacHashBytes = mac.doFinal();
146              out("SCR flow 1.4: (hmac hash bytes): " + toHexString(hmacHashBytes));
147              // SCR flow step 1.5: convert hmac bytes to base64 form
148              hmacHashBase64 = Base64.getEncoder().encodeToString(hmacHashBytes);
149              out("SCR flow 1.5: (hmac hash base 64): " + hmacHashBase64);
150              return hmacHashBase64;
151           }
152           /**
153            * calcutates HTTP Basic Authorization String (SCR flow step #2)
154            *
155            * @param username
156            * @param password
157            * @return the Authorization string
158            * @throws UnsupportedEncodingException
159            */
160           public static String calcBasicAuthString(String username, String password)
    throws UnsupportedEncodingException {
161              String authString = username + ":" + password;
162              out("SCR flow 2.1: Basic Auth usernamepassword string:  {0} ", authString);
163              out("SCR flow 2.1a: Basic Auth usernamepassword bytes[]:  {0} ",
164                    ScrCryptoHelper.toHexString(authString.getBytes("UTF-8")));
165              authString = Base64.getEncoder().encodeToString(authString.getBytes("UTF-8"))
    ;
166              out("SCR flow 2.2: Basic Auth usernamepassword base64:  {0} ", authString);
167              authString = "Basic " + authString;
168              out("SCR flow 2.3: complete Basic Auth string:  {0} ", authString);
169              return authString;
170           }
171           /**
172            * wandelt eine Bytefolge in einen HexString um.
173            *
174            * @param pArray
175            *           bytearray.
```

```
176      * @return Strin mit hexadezimalziffern.
177      */
178     public static String toHexString(byte[] pArray) {
179        return DatatypeConverter.printHexBinary(pArray);
180     }
181     /**
182      * wandelt einen HexString in ein Bytearray um.
183      *
184      * @param pHexStr
185      *           string mit Hexadezimalziffern.
186      * @return das dem String entsprechende Bytearray.
187      */
188     public static byte[] toByteArray(String pHexStr) {
189        return DatatypeConverter.parseHexBinary(pHexStr);
190     }
191     /**
192      * Konsolenausgabe mit Paramterersetzung
193      *
194      * @param message
195      * @param args
196      */
197     public static void out(String message, Object... args) {
198        if (args.length == 0) {
199           System.out.println(message);
200        } else {
201           System.out.println(MessageFormat.format(message, args));
202        }
203     }
204  }
```

Class ScrCallTests

Contains 3 Junit Tests

1. testGetCasesUnencrypted - processing of an SCR get request without encryption
   (Response is unencrypted - this will only work on ITU environment)
2. testGetCasesEncrypted - processing of an SCR get request with encrypted response
   (Response is encrypted but will not be decrypted)
3. testGetCasesEncryptedWithDecryption - processing of an SCR get request with encrypted response
   (Response is encrypted and will be decrypted)

ScrCallTests

```
1   package de.deutschepost.postident.demo.scr;
2
3   import static org.assertj.core.api.Assertions.assertThatNoException;
4   import static org.junit.Assert.assertEquals;
5   import static org.junit.Assert.assertTrue;
6
7   import java.io.UnsupportedEncodingException;
8   import java.security.InvalidKeyException;
9   import java.security.KeyPair;
10  import java.security.NoSuchAlgorithmException;
11  import java.text.MessageFormat;
12  import java.text.ParseException;
13
14  import org.apache.tomcat.util.codec.binary.Base64;
```

```java
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

import com.nimbusds.jose.JOSEException;

/**
 * Scr Call JUnit Tests
 *
 * JUNIT tests to demonstrate SCR result data retrieval with decryption
 *
 * @author Deutsche Post AG
 * @version 1.0
 */
public class ScrCallTests {
    /* keyLength of the RSA key pair used in the test ( 3072 or 4096 Bit) */
    static int key_length = 3072;
    /* Username for Basic Auth */
    static String scr_user = "<your username>";
    /* Password for Basic Auth */
    static String scr_password = "<your password>";
    /* Data password for HMAC calculation */
    static String scr_datapassword = "<your datapassword>";
    /* clientid, used as request parameter */
    static String scr_clientid = "<your clientid>";
    /*
     * Host of the SCR endpoint postident-itu.deutschepost.de (test environment) or
     * postident.deutschepost.de (productive system)
     */
    static String scr_host = "postident-itu.deutschepost.de";
    /* URL for the SCR GET request getting all cases for clientid */
    static String scr_url_full_all = "https://" + scr_host + "/api/scr/v1/" +
scr_clientid
            + "/cases/full";//?inProgress=true&new=true&closed=true";
    /* The SCR key pair used in the tests */
    static KeyPair keypair;
    /* Base64 representation of the public key of the above key pair */
    static String pubkey_base64;
    /* Base64 representation of the private key of the key pair above */
    static String privkey_base64;
    /*
     * the HMAC hash of the public key (calculated with the scr_datapassword as
hash
     * secret)
     */
    static String hmac_hash;

    /**
     * Generation of the RSA key pair used in the tests and calculation of the HMAC
     * hash
     *
     */
    @BeforeAll
    public static void inititalizeAll()
            throws NoSuchAlgorithmException, InvalidKeyException,
UnsupportedEncodingException {
        inititalizeKeypair();
        hmac_hash = ScrEncryptionHandler.hmacHashOverKey(scr_datapassword,
pubkey_base64);
    }
```

```
70
71      /**
72       * generate the RSA key pair. the key pair and the base64 representations of
    the
73       * public and private key are stored in static class variables
74       */
75      public static void inititalizeKeypair() throws NoSuchAlgorithmException {
76          // use KeyPairGenerater to generate RSA keypair
77          java.security.KeyPairGenerator keyGen =
    java.security.KeyPairGenerator.getInstance("RSA");
78          keyGen.initialize(key_length);
79          // generate keypair
80          keypair = keyGen.genKeyPair();
81          // store keys in base64 format
82          pubkey_base64 =
    Base64.encodeBase64String(keypair.getPublic().getEncoded());
83          privkey_base64 =
    Base64.encodeBase64String(keypair.getPrivate().getEncoded());
84          out("pubkey:  " + pubkey_base64);
85          out("privkey: " + privkey_base64);
86      }
87
88      /**
89       * Processing of an SCR get request (unencrypted answer)
90       *
91       * The result is unencrypted because the headers x-scr-key and x-scr-keyhash
    are
92       * not be set. Note: The production environment suppresses unencrypted result
93       * querys
94       */
95      @Test
96      void testGetCasesUnencrypted() {
97          out("JUNIT Test testGetCasesUnencrypted");
98          String ret = ScrHttpRequestHandler.callScr(scr_url_full_all, scr_user,
    scr_password);
99          out(ret);
100         assertThatNoException();
101         assertTrue(ret.startsWith("["));
102     }
103
104     /**
105      * Processing of an SCR get request (encrypted response without decryption)
106      *
107      * The result is encrypted because the headers x-scr-key and x-scr-keyhash are
108      * set. no decryption takes place in this test, the first ones Characters of
    the
109      * encrypted response are output on the console
110      */
111     @Test
112     void testGetCasesEncrypted() throws ParseException, JOSEException, ScrException
    {
113         out("JUNIT Test testGetCasesEncrypted");
114         String ret = ScrHttpRequestHandler.callScr(scr_url_full_all, scr_user,
    scr_password, pubkey_base64, hmac_hash);
115         out(ret.substring(0, 80) + " ...");
116         assertThatNoException();
117         assertTrue(ret.startsWith("eyJlbmMi"));// base64 representation of
    '{"enc":"'
118     }
```

```
119
120     /**
121      * Processing of an SCR get request (encrypted response with decryption)
122      *
123      * The result is encrypted because the headers x-scr-key and x-scr-keyhash are
124      * set. The decrypted payload of the Encrypted Response is sent to Console
125      * output
126      */
127     @Test
128     void testGetCasesEncryptedWithDecrypt() throws ParseException, JOSEException,
    ScrException {
129         out("JUNIT Test testGetCasesEncryptedWithDecrypt");
130         String ret = ScrHttpRequestHandler.callScr(scr_url_full_all, scr_user,
    scr_password, pubkey_base64, hmac_hash);
131         out(ret.substring(0, 80) + " ...");
132         String decrypted = ScrEncryptionHandler.decryptPayload(ret,
    privkey_base64);
133         assertThatNoException();
134         assertTrue(ret.startsWith("eyJlbmMi"));// base64 representation of
    '{"enc":"'
135         assertTrue(decrypted.startsWith("["));// begin of json array
136         out(decrypted);
137     }
138
139     /**
140      * console output with parameter substitution
141      *
142      * @param message
143      * @param args
144      */
145     public static void out(String message, Object... args) {
146         if (args.length == 0) {
147             System.out.println(message);
148         } else {
149             System.out.println(MessageFormat.format(message, args));
150         }
151     }
152
153 }
154
```

3.5.4 Java Snippets

RSA Java Snippet to Decrypt the JWE Response

rsaDecrypt

```
1  public static String decryptRSA(String jweString, String rsaPrivKey) throws
    InvalidKeySpecException, NoSuchAlgorithmException, ParseException, JOSEException{
2         String ret = "";
3         RSAPrivateKey rpk = base64ToPrivateKey(rsaPrivKey) ;
4         JWEObject jweObject = JWEObject.parse(jweString);
5         jweObject.decrypt(new RSADecrypter(rpk));
```

```
        ret = jweObject.getPayload().toString();
        return ret;
}

public static RSAPrivateKey base64ToPrivateKey(String base64Bytes) throws
 InvalidKeySpecException, NoSuchAlgorithmException{
    byte[] keybytes = Base64.getDecoder().decode(base64Bytes);
    return (RSAPrivateKey) KeyFactory.getInstance("RSA").generatePrivate(new
 PKCS8EncodedKeySpec(keybytes));
}
```

3.5.5 PHP Client Sample

The following snippet generates an RSA key pair before the SCR service call. Alternatively, it is possible to use a static file to deposit the RSA key pair.

PHP Client Sample

```
/*
SCR PHP client sample code

this sample uses php seclib 3
see https://phpseclib.com/docs/install for installation instructions
*/
<?php

include 'c:\php\vendor\autoload.php';
use phpseclib3\Crypt\PublicKeyLoader;
use phpseclib3\Crypt\AES;
use phpseclib3\Crypt\RSA;

// scr credentials
$user_name ="<your user>";
$password = "<your password>";
$clientid ="<your clientid>";
$data_password = "<your datapassword>";
// possible hosts: postident-itu.deutschepost.de (test) or
postident.deutschepost.de (production)
$url_API = 'https://postident-itu.deutschepost.de/api/scr/v1/' . $clientid . '/
cases/full';

// Generate key pair with 3k size
$private_key = RSA::createKey(3072);
$public_key = $private_key->getPublicKey();

$ch = curl_init();
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
curl_setopt($ch, CURLOPT_SSL_VERIFYSTATUS, false);
// for error analysis enable curl verbose
// curl_setopt($ch, CURLOPT_VERBOSE, true);


$headers = [
```

```
    'Content-type: application/JSON; CHARSET=UTF-8',
    'Authorization: Basic '. base64_encode("$user_name:$password"),
    'x-scr-enc: A256GCM', // PHP Seclib3 only supports GCM
    'x-scr-alg: RSA-OAEP-256',
    'x-scr-key: '.exctractKeyFromCert($public_key),
    'x-scr-keyhash: '.base64_encode(hash_hmac('sha256', base64_decode(exctractKeyFr
omCert($public_key)), $data_password, TRUE))
];
var_dump('headers ', $headers );

curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt( $ch, CURLOPT_URL, $url_API );

$result = curl_exec( $ch );
if ($result === FALSE) {
    printf("cUrl error (#%d): %s<br>\n",
           curl_errno($ch),
           htmlspecialchars(curl_error($ch)))
           ;
}
echo " R E S U L T :\r\n";
var_dump($result);
$decrypted = jweRsaDecryptFromBase64Token($private_key, $result);
var_dump($decrypted);

// cert is surrounded by BEGIN/END text - this has to be removed
function exctractKeyFromCert($cert) {
    $key = preg_replace('/\\-\\-\\-\\-.*\\-\\-\\-\\-/', '', $cert);      // remove
the -----BEGIN CERTIFICATE----- and -----END CERTIFICATE----- stuff
    $key = preg_replace("/[\n\r]/", '', $key);         // remove new lines
    return $key;
 }

function jweRsaDecryptFromBase64Token($rsaPrivateKey, $jweTokenBase64)
{
    list($jweHeaderBase64, $jweEncryptedKeyBase64, $nonceBase64,
$encryptedDataBase64, $gcmTagBase64) = explode('.', $jweTokenBase64, 5);

    $keyAlg = getJweHeaderKeyAlg($jweHeaderBase64);
    $encryptionAlg = getJweHeaderEncryptionAlg($jweHeaderBase64);
    $aesDecryptionKey = "";
    switch ($keyAlg) {
        case 'RSA-OAEP-256':
        {
            $rsa = PublicKeyLoader::load($rsaPrivateKey)
                ->withHash('sha256')
                ->withMGFHash('sha256');
            $aesDecryptionKey = $rsa->decrypt(decodeBase64Url($jweEncryptedKeyBase6
4));
            break;
        }
        default:
        {
            // Alg nicht unterstützt
            return '*** Alg nicht unterstützt ***';
        }
    }
    switch ($encryptionAlg) {
```

```
89          case 'A128GCM' || 'A192GCM' || 'A256GCM':
90          {
91              $nonce = decodeBase64Url($nonceBase64);
92              $ciphertext = decodeBase64Url($encryptedDataBase64);
93              $gcmTag = decodeBase64Url($gcmTagBase64);
94              $cipher = new AES('gcm');
95              $cipher->setKey($aesDecryptionKey);
96              $cipher->setNonce($nonce);
97              $cipher->setAAD($jweHeaderBase64);
98              $cipher->setTag($gcmTag);
99              return $cipher->decrypt($ciphertext);
100         }
101         default:
102             {
103                 // Alg nicht unterstützt
104                 return '*** Alg nicht unterstützt ***';
105             }
106     }
107 }
108
109
110 function getJweHeaderKeyAlg($jweHeaderBase64)
111 {
112     $array = json_decode(decodeBase64Url($jweHeaderBase64), true);
113     return $array['alg'];
114 }
115
116 function getJweHeaderEncryptionAlg($jweHeaderBase64)
117 {
118     $array = json_decode(decodeBase64Url($jweHeaderBase64), true);
119     return $array['enc'];
120 }
121
122
123 function encodeBase64Url($data)
124 {
125     return rtrim(strtr(base64_encode($data), '+/', '-_'), '=');
126 }
127
128
129 function decodeBase64Url($data)
130 {
131     return base64_decode(str_pad(strtr($data, '-_', '+/'), strlen($data) % 4, '=',
    STR_PAD_RIGHT));
132 }
133
134
135 ?>
```

3.5.6 Python Client Sample

See https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/

The python modules cryptography and jwcrypto are required.

The following snippet uses a before calculated rsa keypair, prepaires and fires a request and decrypts the response .

SCR Python Client

```
'''

Postident SCR Python Reference Client

Created on 26.06.2022

@author: @author Deutsche Post AG

# RSA
# https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/


run pip install cryptography
run pip install jwcrypto

'''

import base64
import http.client

from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes, hmac
from cryptography.hazmat.primitives import serialization
from jwcrypto import jwk, jwe
from jwcrypto.common import json_decode


#HTTP CONFIGURATION
HTTP_USER_AGENT="SCR-CLIENT"
HTTP_CONTENT_TYPE="application/json"
# ! please take care to use the right hostname
POSTIDENT_HOSTNAME_STR="postident-itu.deutschepost.de"


#ENCRYPTION CONFIGURATION
SCR_ENC_ALG="RSA-OAEP-256"
SCR_ENC="A256CBC-HS512"


# CLIENT CONFIGURATION
USERNAME_STR = "<your username here>"
PASSWORD_STR = "<your password here>"
CLIENTID_STR = "<your clientid here>"
DATAPASSWORD_STR = "<your datapassword here>"

rsa_private_key_loaded3k_base64=\
```

```
47
"MIIG4gIBAAKCAYEApmc13IbTFltQJyH7CRipWfK4ibMikIxeDVO1XjpndbBNHDfQk0C+jZLnzNGpy3BvaR
r1TqWbV"\
48      "06paOazw6v2iocaNEGYA+6bxwf7rKhrmAYzce91E8bzIgbmyLAHU/
d0yPI3lcxqL01QxaF6gi+uSA88Q5loSK7VCp"\
49      "SObb9sXG02iJL4Op0An9s1n6/8NXxyu41tA/GxMTmbSWjH8VlRcO9xr7/ja88aIHIMFjgG9+yr/
IPyr6i0kWtOciF"\
50      "vT+scEseVTxVU7adSOrQPW0OD/yjspIqwHr9WWqv0DGY8/
ha+mY2A0zaYUWYoWS7MnfFaBsDHl01z4vwdl0BwE3Y2"\
51      "lCkrIK6IohkV2RqB5pB7nUrUAzEiDy23Cv31DdF1fJ0R2mW/
GeTQW6oqL7n5pRUsdmsiwVy7pGG1STUJ6qmZgsQsN"\
52
"vYkxWrkbn+C6qSK+xjAs3eqJsSrmIvb0PLYVMWmXXbVvRvonm4KgumCpJGZ4f+1KeI92MSMKekxdy4ACLh
BAgMBAA"\
53      "ECggGAJYW8dX/JdY8dodVdwA1ZkrjSt+o27xVX1/
V6Pkc5MSaSHLzRQYMl9NKdhmf4u7EczjyP+C4Lu0I3nTn4azU"\
54
"JoO8aD5KC7tANPHImZCaOLHepfhWa9tyz0oZv3+0hmeOA6BcGRbcUcpoijo39DAIocINO+YdMJqeVzo71F
vZFPwo5"\
55      "c86APudMTr+ldd6/JmhrjrMzU+JaLFlrFMCV5WxYmvjDT/
gbFiJbJ9nPs+rqV1l85rHyNX37SZdC1yWG7iR7kO6wg"\
56
"lMii9oI1KM2o40QsTMI+zyxDushE3w6lZPPxxNaawqOJdpJfzsEhRGJz0IS2UZZ50t02x1zImh0o5uVwXy
HeMMnNC"\
57      "vowFo+BvTeAbFlDn5cXpEWx+mX7i2Q8qPLT5/rvYxjSjRvAUhhErvuF15/TFzhXDe/
J0gFuXVzktAJeCOVfgrkk9Q"\
58      "vJ9HkzF/
q7WltAKE0EHy5bB1J1e0d0fnDg5hJvsQCsT+89LbNFQ35MeyPX54mvyY3oWw8I7XpAoHBANczIViDxLmB"\
59
"ej2OGmVTG+1Cl7h7xyeQDetWwARYBx4G+e6jxAXAzvRRXqSPNhWv7b3lVLUszWU5W4cAMEPlhhRceyhkXf
kEDBDiG"\
60      "CVEORjUCoJ7E+l6AAB+gSbUbpUqjRBlATinnwmnuY/
2X+0OFUEl30nsXkaFhr2WfQLLehuTa6GUyAK0ieYmEMDcl0"\
61      "C+C5E5swMuSpmuszleuGxgwaqY29JpDSBvc3FxnMYyaMVMi/
VxeEsHvCJ6+wHyB9pvTwKBwQDF87RsBmjUIijOeq3"\
62      "aigXptwljjuwXKOh9POpF9hLDbKDn8U3tkcq5IICri4wf506AD/OrQlu/7fUZgjKBbzKchCgQxg/
JbB9cskHAsYOA"\
63      "+aHne0fRI+kNeI+Wpx6q8UFGHlcp0gSMHaabjEO1PRbxbF4PUwY2csO/kW/
P30hihRxSxbQt3bNsB8Se75xzJOoMy"\
64      "j2BF/pYM9g7uDEs1psyBoVg4UKLybHlqxEQgkq306eYd9EoMrT7h7u/
afPQ+28CgcAGJMiL7V8daKvju/3W7LN8Z4"\
65      "1LUAVUhNFQ6a4bsaOqYMqb3IMJIwMFiJkqG4iQv/AKntR3Q6ste6C4TvIRziiwxh8h/
RONu2bYyIul7LewlMUkCn1"\
66      "8CeacHQB06lWp3ogecrPBOU/aB7bNfFwquV8z9+/S/XOHkeJR4Uz6WnLG/
MNy3nuDUEIrluSdjO6og4RzBfGtYpHw"\
67      "p0MonSKovW5p/
2kvLZ6ZUXU7ROIT6naBQ3kvbajgg30EgvxNhupgexMCgcBEHITaHqJ3S4Gq+j9T0OYT8u0CGq1Vk"\
68      "rfcT9d0VyVBpOEAreDYe6+7X1mmrQ978sTp/
5MiPGUBd0k8i7YBH1fUzwxSKTRJZDCuXo+NVAchm8N/uMWPSsHn5r"\
69
"HRbpN40iZZkBsBwsfZxmILZ4nslKaOT3FV6IVcusRr6AkHB5cKfy6ttGU42u3foBShc2Troan+2J+tCakU
LFKcydX"\
70      "xza382o2ONjQFkVLq6Z+nhI1dDzC9n4ySPlBTs/J//F7ua3UCgcA/ipe/
YMbP8+N79+w0tWDJWMnwJ+tJ0tG9F3Bj"\
71      "Mh2Se4KDQZbhgacReEo3jb6WQ2NxkELsl3ciozZ996VwHTQ6LJHoRJZeN/
3ugUQkxItsMetHt23EbU1BFF6CPtel4"\
72      "MoVi85FxF/mhGvvy/Bc7ri1BaaKVHMriRpxqVJ+diXdfwK/
zLe+dwLhBd70vP38YhtTdbJ2dPsm2gW2h+0+g04NE4"\
73      "prWDKhjNr6EBmjw/3kQEhXF4MJ3FT5j13W0KKx3xQ="
```

```
74
75  rsa_private_key_loaded_bytes = base64.b64decode(rsa_private_key_loaded_base64)
76  rsa_private_key_loaded = serialization.load_der_private_key(
77      data=rsa_private_key_loaded_bytes ,
78      password=None
79      )
80
81  ''' OPTION:
82  to calculate rsa keypair use this
83  rsa_private_key_generated = rsa.generate_private_key(
84      public_exponent=65537,
85      key_size=3096,
86  )
87  '''
88  # work with loaded private key - it will be more secure often to change the private
    key
89  rsa_private_key = rsa_private_key_loaded  # alternatively:
    rsa_private_key_generated
90  rsa_public_key = rsa_private_key.public_key()
91
92
93  # bring the private key in pem form
94  rsa_private_key_pem = rsa_private_key.private_bytes(
95     encoding=serialization.Encoding.PEM,
96     format=serialization.PrivateFormat.TraditionalOpenSSL,
97     encryption_algorithm=serialization.NoEncryption()
98  )
99
100 publicKeyDerBytes = rsa_public_key.public_bytes( encoding=serialization.Encoding.DE
    R, format = serialization.PublicFormat.SubjectPublicKeyInfo)
101 public_key_der_base64 = base64.b64encode( publicKeyDerBytes ).decode("ascii")
102
103 print( "p4.3 get public  key in byte[] form " , publicKeyDerBytes.hex())
104 print( "p4.4 get public  key in Base64 String Form " , public_key_der_base64)
105 print( "p4.5 get private key in byte[] form " , rsa_private_key_pem.hex())
106 print( "p4.6 get private key  " , rsa_private_key_pem)
107
108
109
110 # ALL STEPS
111 '''
112 1.1 convert datapassword to byte[]
113 '''
114 dataPasswordAsBytes = str.encode(DATAPASSWORD_STR, "utf-8")
115 datapassword_as_hexstring = dataPasswordAsBytes.hex()
116 print( "1.1 dataPasswordBytes                " , DATAPASSWORD_STR, " ",
    datapassword_as_hexstring)
117
118
119 '''
120 1.2 convert RSA public key to byte[] and base64
121 '''
122 print( "1.2 public key in DER encoding in byte[] form" , publicKeyDerBytes.hex())
123 print( "1.2 public key in DER encoding in base64" , public_key_der_base64  )
124 '''
125 1.3 create and initialize javax.crypto.Mac
126
127 SecretKeySpec signingKey = new SecretKeySpec(dataPasswordBytes, "HmacSHA256");
128 Mac mac = Mac.getInstance("HmacSHA256");
```

```
mac.init(signingKey);
'''
hmacHashser = hmac.HMAC(dataPasswordAsBytes, hashes.SHA256(),
backend=default_backend())

'''
1.4 calculate HMAC hash bytes'''
hmacHashser.update(publicKeyDerBytes)
hmacHashBytes = hmacHashser.finalize()

print( "1.4 hmacHashBytes" , hmacHashBytes.hex())
'''
1.5 convert hmac bytes to base64 form

String HMAC_HASH_BASE64 = Base64.getEncoder().encodeToString(hashBytes);
'''
hmac_hash_base64 = base64.b64encode( hmacHashBytes ).decode("ascii")
print( "1.5 HMAC_HASH_BASE64" , hmac_hash_base64  )


"""Step 2: Calculate the basic authorization header
Code Sample
IN: username (String - precondition #2) password (String - precondition #2)
OUT: HTTP authorization header string
"""


"""2.1    concatenate username + ":" + password    String userAndPass = username +
":" + password;    userAndPass =
            SCRDEMO:3r#4Mu#GBRmP
"""
userAndPass = USERNAME_STR + ":" + PASSWORD_STR
print( "2.1 userAndPass step 1" , userAndPass  )

"""2.2    convert the result of step 2.1 into base64 form    userAndPass =
Base64.getEncoder().encodeToString(userAndPass.getBytes("UTF-8"));    userAndPass =

            U0NSREVNTzozciM0TXUjR0JSbVA=
"""
userAndPass = base64.b64encode(str.encode(userAndPass, "utf-8")).decode("ascii")
#decode transforms bytearray to string
print( "2.2 userAndPass step 2 base64" , userAndPass  )

"""2.3    prepend "Basic " to authorization header value    userAndPass = "Basic "
+ userAndPass;    userAndPass =
            Basic U0NSREVNTzozciM0TXUjR0JSbVA=
   2.4    when generating HTTP-request    httpUrlConnection.setRequestProperty("Aut
horization", userAndPass);    Authorization: Basic U0NSREVNTzozciM0TXUjR0JSbVA=
    Add authorization header to the request
"""
HTTP_REQUEST_HEADERS = { 'Authorization' : 'Basic %s' %  userAndPass }
print( "2.3 + 2.4  3 prepend \"Basic \" assign to Authorization" ,
HTTP_REQUEST_HEADERS  )


'''3.1

build SCR URL
```

```
String scrUrl = MessageFormat.format("https://{0}/api/scr/v1/{1}/cases",host,
clientid);

https://postident.deutschepost.de/api/scr/v1/865E6E37/cases
'''
SCR_PATH = "/api/scr/v1/%s/cases" % CLIENTID_STR
print( "3.1 build scr url" , SCR_PATH  )


'''3.2 instantiate and configure URL connection'''

http_connection = http.client.HTTPSConnection(POSTIDENT_HOSTNAME_STR)

HTTP_REQUEST_HEADERS["User-agent"]=HTTP_USER_AGENT
HTTP_REQUEST_HEADERS["Content-Type"]=HTTP_CONTENT_TYPE



'''
3.3 set Authorization Header (alredy done in 2.4)
3.4 set x-scr-key Header
3.5 set x-scr-keyhash Header
'''

HTTP_REQUEST_HEADERS["x-scr-alg"]=SCR_ENC_ALG
HTTP_REQUEST_HEADERS["x-scr-enc"]=SCR_ENC
HTTP_REQUEST_HEADERS["x-scr-key"]=public_key_der_base64
HTTP_REQUEST_HEADERS["x-scr-keyhash"]=hmac_hash_base64

print( "3.3, 3.4, 3.5  add x-scr header" , str(HTTP_REQUEST_HEADERS).replace(",",",
\n")  )

payload = ""

'''
3.6 fire the Request
'''
http_connection.request("GET", "/api/scr/v1/%s/cases" % CLIENTID_STR , payload,
HTTP_REQUEST_HEADERS)
print("send Request to : /api/scr/v1/%s/cases" % CLIENTID_STR)

''' 4. Postident System is processing the Request '''

''' 5. receive encrypted data '''
http_response = http_connection.getresponse()
encrypted_response = http_response.read()
print("returncode: %s" % http_response.getcode() )
print(encrypted_response)


''' Step 6: decrypt data with private key
    use JWE to decrypt

    6.1 parse encrypted payload into JWEObject (implicit in 6.4)
    6.2 instantiate RSAPrivateKey '''
jwk_private_key = jwk.JWK.from_pem(rsa_private_key_pem)
''' 6.3 instantiate Decrypter '''
jwe_decrypt_token = jwe.JWE()
''' 6.4. decrypt JWEObject'''
```

```
jwe_decrypt_token.deserialize(encrypted_response.decode("utf-8"), jwk_private_key)
''' 6.5. get decrypted Response'''
payload = jwe_decrypt_token.payload
print(payload.decode('utf-8'))
```