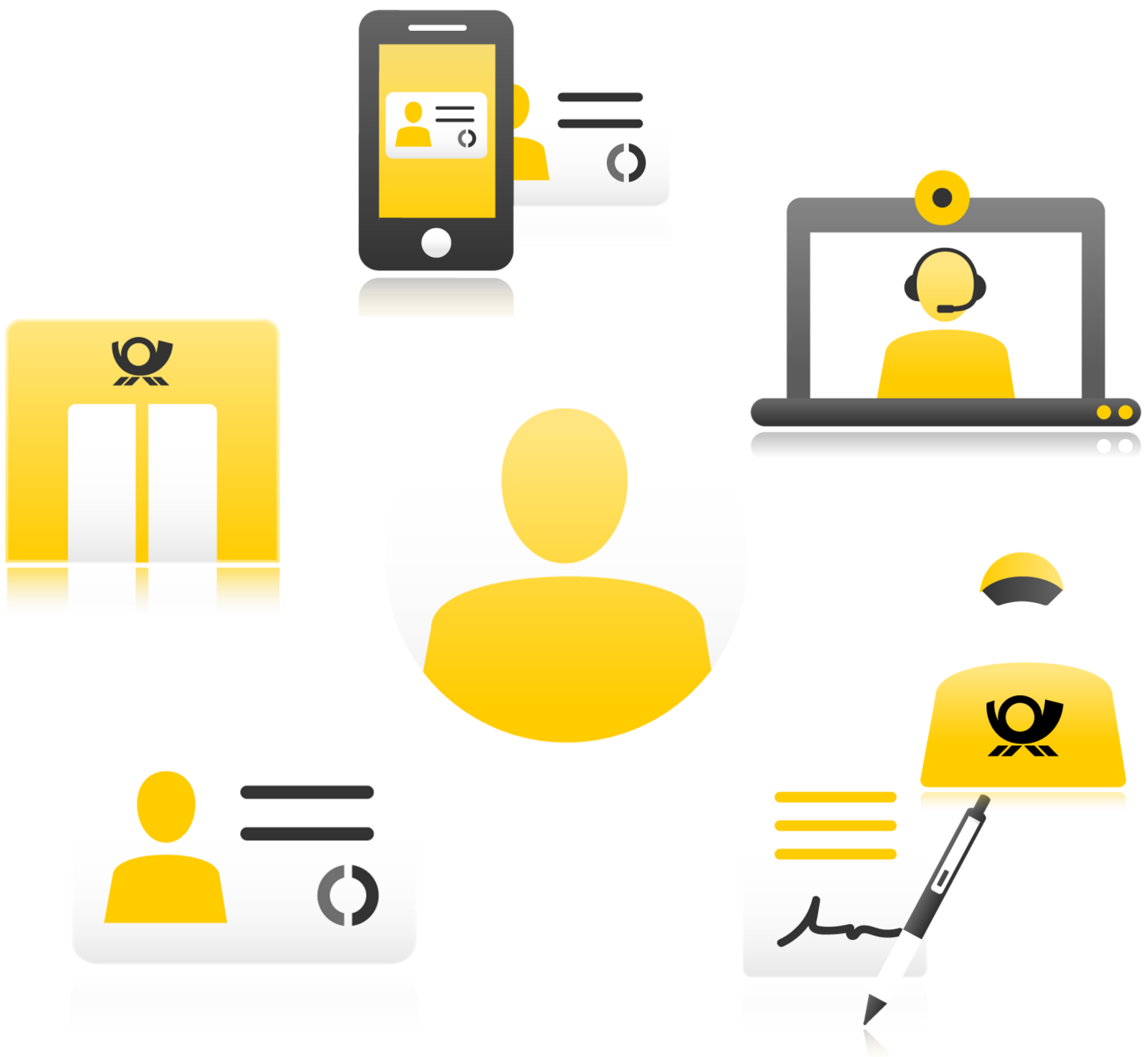


Standard Connect & Result (SCR) API



SCR-Ident API Guide 3 - Encryption

Contents

1	Preface.....	5
2	Part I - Incoming Encryption	6
2.1	Encrypt incoming payload	6
2.2	Send encrypted payload to the POSTIDENT system.....	6
2.3	Encryption Details	7
2.3.1	Components of the JWE Compact Serialization.....	7
2.3.2	Password-based encryption PBES2 for incoming SCR encryption.....	7
2.3.3	Payload Encryption with AES GCM using 256-bit key.....	8
2.3.4	Example Payload	8
3	Part II - Outgoing Encryption	9
3.1	Overview.....	9
3.1.1	Preconditions	9
3.1.2	General Flow.....	9
3.2	Encryption Settings using header.....	10
3.2.1	HTTP-Header.....	10
3.3	Detailed Flow with Samples.....	12
3.3.1	Preconditions	13
3.3.2	RSA key pair generation	13
	Sample RSA 3072 key pair.....	13
	Code sample RSA keypair generation.....	14
3.3.3	Calculate HMAC of public key	15
	Code Sample for HMAC calculation.....	15
3.3.4	Handle the get Request in 3 JUnit Tests	17
3.3.5	Test case 1 Processing of the HTTP request without encryption.....	17
3.3.6	Test case 2 Processing of the HTTP request with encryption Parameters (no decryption on client side)	18
3.3.7	Test case 3 Processing of the HTTP request with encryption Parameters (with decryption on client side)	19
3.3.8	Sample Response of SCR getCases full:	20
3.4	Errors.....	23
3.4.1	Sample Error Messages.....	23
3.4.2	Encryption Errors	23
3.4.3	Typical error situations and error messages.....	24
3.5	Code Samples.....	25
3.5.1	Disclaimer.....	25
3.5.2	Import the provided Eclipse Project	25
3.5.3	Java SCR Sample JUnit Tests	25

java source..... 25

3.5.4 Java Snippets..... 38

 RSA Java Snippet to Decrypt the JWE Response 38

3.5.5 PHP Client Sample 38

3.5.6 Python Client Sample 41

Changelog

Date	Change
 13.08.2026	Added example SCR Caller with method "JWT Bearer".
 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

```

1  /** Snippet used in scr encryption incoming documentation
2  * @param strPayload json payload string to be encrypted
3  * @param dataPassword password used for pbes2 encryption
4  * @return jwe Cipher
5  * @throws NoSuchAlgorithmException
6  * @throws JOSEException
7  */
8  public String encryptPayloadScrIncomingSnippet(String strPayload, String
dataPassword)
9      throws NoSuchAlgorithmException, JOSEException {
10     // PBES2 is currently the only supported option
11     JWEAlgorithm jweAlg = JWEAlgorithm.PBES2_HS512_A256KW;
12     EncryptionMethod encMethod = EncryptionMethod.A256GCM;
13     int saltLength = 8;
14     int iterationen = 1000;
15     JWEHeader header = new JWEHeader.Builder(jweAlg, encMethod).build();
16     Payload payload = new Payload(strPayload);
17     JWEObject jweObject = new JWEObject(header, payload);
18     JWEEncrypter encrypter = null;
19     encrypter = new PasswordBasedEncrypter(dataPassword, saltLength, iterationen);
20     jweObject.encrypt(encrypter);
21     String ret = jweObject.serialize();
22     return ret;
23 }
```

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

```
1  curl --location --request PATCH 'https://<HOSTNAME>/api/scr/v1/<CLIENTID>/cases' \
```

```

2  --header 'Content-Type: application/jose' \
3  --header 'Authorization: Basic
bmljX3Jlc3Rfc2NyX2lkZW50OjJSeTYjY3ZQY3JH0XEjI1Q0ZVJTN0dHMcw==' \
4  --data-raw
'eyJwMnMiOiJ0TFFjX0hIWwtjMCIsInAyYyI6MTAwMCwiZW5jIjoiTl1NkdDTSIImFsZyI6IiBCRVMyLU
hTNTeyK0EYNTZLVyJ9.smhH-zNsQI6HZ_89GBas4MWKEaiSTvcEe0-
hdGUqKpkfiWLJq_10Ig.8g6s0_ivZN8K9YLw.SoZv-
_Sj3sNCCD68aMez8_3dGqDarcAl.46E10FU3vD4TYd9PW5U7Ug'
```

HTTP level request data with incoming encrypted payload

```

1  HTTP:
2
3  PATCH /api/scr/v1/<CLIENTID>/cases HTTP/1.1
4  Host: <HOSTNAME>
5  Content-Type: application/jose
6  Authorization: Basic bmljX3Jlc3Rfc2NyX2lkZW50OjJSeTYjY3ZQY3JH0XEjI1Q0ZVJTN0dHMcw==
7  Content-Length: 228
8
9  eyJwMnMiOiJ0TFFjX0hIWwtjMCIsInAyYyI6MTAwMCwiZW5jIjoiTl1NkdDTSIImFsZyI6IiBCRVMyLUh
TNTeyK0EYNTZLVyJ9.smhH-zNsQI6HZ_89GBas4MWKEaiSTvcEe0-
hdGUqKpkfiWLJq_10Ig.8g6s0_ivZN8K9YLw.SoZv-
_Sj3sNCCD68aMez8_3dGqDarcAl.46E10FU3vD4TYd9PW5U7Ug
```

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-HS12+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_ljM, 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	eyJwMnMiOiJ1amRIWXNlNkQtNCIsInAyYyI6MTAwMCwia2lkIjoia2V5I2RhdGFwYXNzd2QjMDAwMDEiLCJlbmMiOiJBMjU2R0NNIiwiaWYwXnIjoiUEJFUzltSFM1MTIrQTl1NktXIn0. 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.

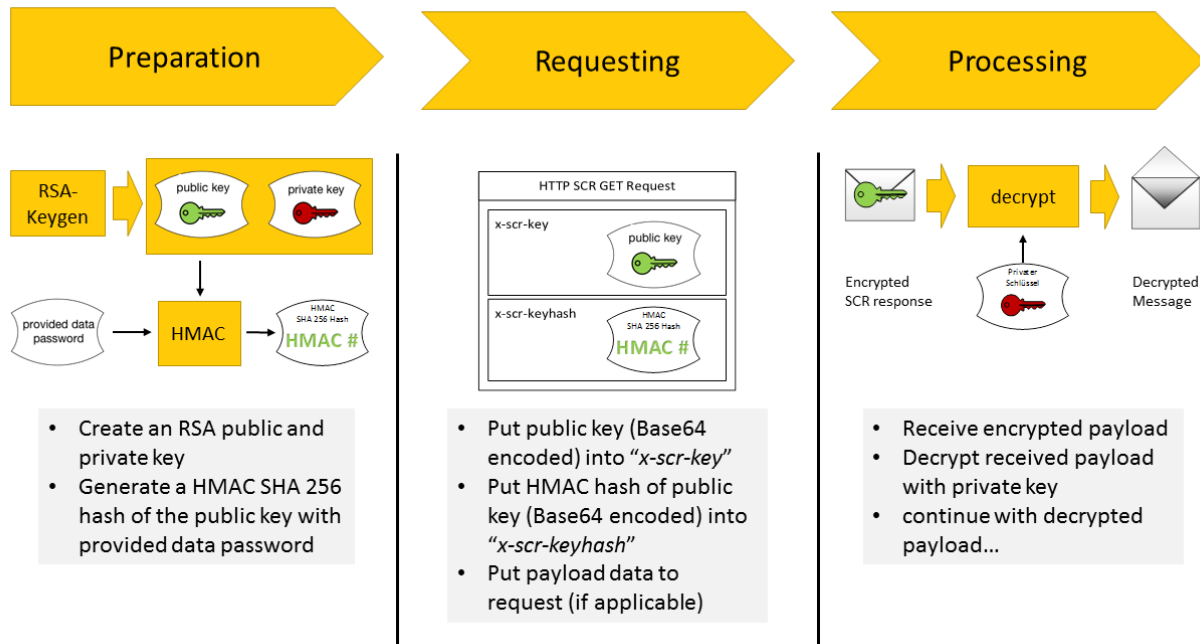
3.1.2 General Flow

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

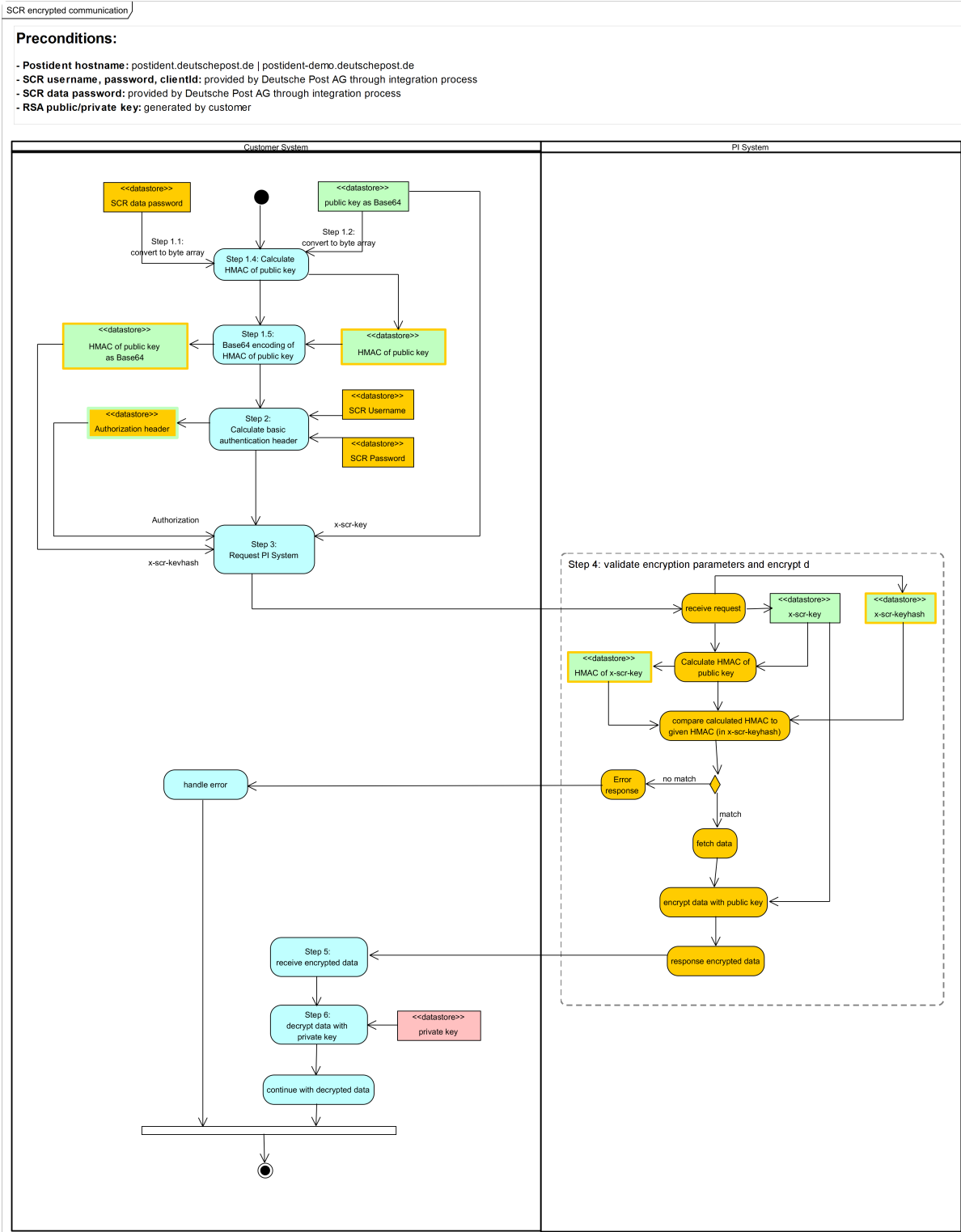
3.2.1 HTTP-Header

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	Default (if header is empty)	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(...)Fope02Z6TrwIDAQAB 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.		YAcCWwCyEyE6Fg0wuCgip3Aj0k2mU/rU/UGuTW506p0= Used DataPassword: EAHqr_9NvCw2BuI23\$a.0vRsS

Element	Mandatory	Description	Default (if header is empty)	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	

3.3 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
P 1	Postident hostname	production: postident.deutschepost.de itu test environment: postident-itu.deutschepost.de	
P 2	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	
P 3	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.
P 4	auth	"Basic " + Base64.getEncoder().encodeToString((username + ":" + password).getBytes("UTF-8"));	build the basic auth header
P 5	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	
P 6	x-scr-keyhash	base64(HmacSHA256 (base64decode(x-scr-key), data password.getBytes(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

1	===== RSA Public Key =====
2	pubKeyBase64:
3	MIIBojANBgkqhkiG9w0BAQEFAAOCAy8AMIIBigKCAYEAyF8XXjW+iZaYYH6l6i9wj50f73Xom4CcdYAh
4	MIIBojANBgkqhkiG9w0BAQEFAAOCAy8AMIIBigKCAYEA27wHDk8QQ2Jf0pWVRXCuB4WoWysx5un1NPHp
5	HLBRzEetu7/cZHURRPuVLf7ahF94H7P9smzVrkPqEwicUvt+UwdTHqolWYZn0UB+FJ9HLLGFRSGOT0a
6	tIJTX8AfW3qFLJqTKPk2urr59n15f9FayQjtu9YIU/Rpf/8Bxxnvxw/QUu0wJNYEnBoktur+PMc4DWrN

```

7 LapJ6f86luE96pBw4jEl+aSID0K+o6lezgVMuTJEpD0z56HQVs108IhwdF5P07WxtCy92V3WNz4B611v
8 UHy30JujpZQ4rvHsIYPAQD67unS4kga2gGVkmyyMQ7deTUsdd001/xU6Czd1Fv6BXMA98wDTfktfQm5P
9 oUciEtF0mjby2AeF4kQQ9m9JHP7ToqfKD4nUBcqk7vWBcoAmRXX2E69VqkfjqSnY/2c4bX+M7Yy743xY
10 EFpNtWF/PnPXbdjoXmSmr0+fzaV+iH0+iWsegqEVVtJC00a8XrvLwEUD6NN1uXu81ELZixs+5Co7AgMB
11 AAE=
12
13
14 ===== RSA Private Key
15 =====
16 privKeyBase64:
17 MIIG/QIBADANBgqhkiG9w0BAQEFAASCBucwggbjAgEAAoIBgQDbvAcOTxBDYL/SLZVFcK4HhahbKzHm
18 6eU08ekcsFHMR627v9xkdRFE+5Ut/tqEX3gfs/2ybNWuQ+oTCJyJS+35TB1MeqiVzhmfRQH4Un0cssYV
19 FIY5PRq0glNfwB9beoWumpMo+Ta6uvn2eXl/0vRJC0271ghT9GL//wHHGe/HD9BS47Ak1gScGiS26v48
20 xzgNas0tqknp/zqW4T3qkHDiMSX5pIgPQR6jqV70BUy5MkSkPTPnodBWzU7wiHB0Xk/TtbG0LL3ZXdY3
21 PgHrXW9QfLFQm60LLdiu8ewhg8BAPru6dLiSBraAZWSbLIxDt15NSx1047X/FToLN3UW/oFcd3zANMW
22 2R9Cbk+hRyIS0U6aAntjYB4XiRBD2b0kc/t0ip8oPidQFyqTu9YFygCZFdfYTr1WqR+OpKdj/Zzhtf4zt
23 jLvJfFgQWk21YX8+c9dt20heZKavT5/NpX6Ift6Jax6CoRVW0kI45rxeu8vARQP03W5e7zUSVmlGz7k
24 KjsCAwEAAQKCAyBSs93p/ks00af8MA2rQaJwTXESw0C4ex8oQsas5BYdb3sN18QhUUilMK+wzK09P9uL
25 /yhE8B6qx0gEevDD4j6y3nmGkAIEEnRwWsxfvOUjatNTGGV/9iitiTerVdRYfn2+EMEFPCb64wMPCK2oqm
26 14q2cLRdxsYCoNLWeInYQMRVJXgQwp7I9n5srJBWAnNnN8o6jpCkF9xTBjuuLx+pOe0qzTnQUKNfLw1m
27 tvg2TZtXdvqL4Hc5PteV3BdSg9/jhJ009+LpPC6XPeyX5W2wjcmLujHqleZr25JUGTPz2Xc/PyMF9a
28 ajjrFvI2JQaapCbbogX5xT1WBxgUIj0gSXattZE2YA+LYN0p3kWHALfjflJ/hEgXQlvThcc/i7NPuwyZ
29 9LmBUl0nnsUh+ZrBg+qb20d6hR801uUD9r0iIQCPqM+4MsZPEuPLGgw7XyD0rwr8ridyQ0uIOs/MkwW8
30 WUwGkBzTgfJhxn0cAjbFnEVvdqpV9brxcUai7HZpCyu0B5ECgcEA/3gQGwVJebtBW+ZBWFyP2XeJMAVm
31 3ouKmDcixAaPPKGO3WqjW2Pu0c0jc3reL2qGNBdp15RuXhoQgzrriBhbd7+fLxFoEUh8r3r4Ee59fC4V
32 R+duZcAgzn7mGKL06Dmxn1pWRoIMi/oiFF4qASqKHixRG16Qre/qx/Aq148wvpVHW8aj3qi/08WhMpyL
33 C0B9a67x9iZuZhpv4oECM/8m2VwKUQDTLTfGIPT8x4uZm79FwoYnu4Sov0oqZelrKCLAoHBANww8zU2
34 bAtbUxHmcdL+Ep4A4GJ7IEsLYxpCA6rE032h1JPjhsGkFCpCmm9ULuPB7GV08FCi/G8np04k4tQCGAyK
35 owVo91F3/bHDzKg53eQNrLHHK+5v0Wig5FdrJYjzesTzXNE5vHGXIzLStLJvxydbN16f+GkolUYrRiUD
36 q1rXb1H/fzcH/OeoHTZMkditXG104MeLZsVI1XyyabU0+Mbn1tCABa/QGRwvxkrsUbMxhRwHZh9OpA/g
37 w+2Ktj+jEQKBwQct1NKQsE5IyMQ2dzsLT9EqHPu+x3+PFE4NfwwCo6ZvB4sR42UeDD3mUm1IFvjQ8bB
38 11Sr3lIjMfUedo8h8gC0fJ0id+OqLvq4pSH7gZ5dSzyr0SxrrjApGuE51YRzoj6lvVxbxrzg+Xxumf2Wx
39 mXKWCW2/A9/Kmz/UgHWGa+av4MfHiSikL4rAbz+6oo5yP5WYGLiGN6xSL0XHSCVpCqdKGXUrI1MMdMP
40 B0ynVlgXV1paofCMH/8KbNyOWYngJvUCgcAbg25Qxt1/SckGepIMeDzhS2C+TX5KhYBtnoQqYyLXsGn8
41 LLiZt5Bhe2YL7Svyv5+HRSbYHJWIWh8Yr5k8PqQ8wbGxEL2c625i0QF4n8pvHZqDk2yU3ZoABekiXbwr
42 8cKMa5BUgG9gj7flqsHhNoQu9/4UXra8Q+99dhM9blqn3KkgLVmDfR3CdWknEq5FwhvsPAJXwzXZ/Nt
43 Imn8+sLxL7Ty1qkqDKmmkP2pDQbkLKePvpuvQfZCA/TVbrSFtECgcAUga7lVcG4j2z6UXL5DG/68YYG
44 REN+8ZfD3SmZn8rrAP8QuhgFHPe3u2ROCMjwWxYagQ9Yf8uUKapZ/50/fKcrf1ET/G26gsCxxj6i4PRq
45 ch6akyXuVlyR7WbGvqd89N3sdTS7j0/K/p8JB0GKwS2c0UdRsv7GdXBIcWl3ZL8ls7WWNLmucc4kk34G
LOSgzJbr4Y8d0DDboL+eUuIhbXZqqkKyb08x8N/86ytYQNVe0M02qTpN4LVAf03i0qDADA=

```

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	<pre>java.security.KeyPairGenerator keyGen = java.security.KeyPairGenerator.getInstance("RSA"); keyGen.initialize(3072);</pre>	
2	generate keypair	<pre>java.security.KeyPair keypair = keyGen.genKeyPair();</pre>	
4	get public key in base64 form	<pre>String pubKeyBase64 = Base64.encodeBase64String(keypair.getPublic().getEncoded());</pre>	see codeblock above

Keygen Step	Description	java Snippet	Data
6	get private key in base64 form	String privKeyBase64 = Base64.encodeBase64String(keypair.getPrivate().getEncoded());	see codeblock above

initializeKeypair

```

1      /**
2      * generate the RSA key pair. the key pair and the base64 representations of
the
3      * public and private key are stored in static class variables of JUnit
testclass
4      */
5      public static void inititalizeKeypair() throws NoSuchAlgorithmException {
6          // use KeyPairGenerater to generate RSA keypair
7          java.security.KeyPairGenerator keyGen =
java.security.KeyPairGenerator.getInstance("RSA");
8          keyGen.initialize(key_length);
9          // generate keypair
10         keypair = keyGen.genKeyPair();
11         // store keys in base64 format
12         pubkey_base64 =
Base64.encodeBase64String(keypair.getPublic().getEncoded());
13         privkey_base64 =
Base64.encodeBase64String(keypair.getPrivate().getEncoded());
14         out("pubkey: " + pubkey_base64);
15         out("privkey: " + privkey_base64);
16     }

```

3.3.3 Calculate HMAC of public key

Calcuclate an HMAC of your private key as bytearray with the SCR data password 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(publicKeyBase64);	publicKeyBytes = 30 82 01 A2 30 ... 02 03 01 00 01

HMAC Step	Description	java Snippet	sample Data
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+aJLSPf83Cmfk eXI8U00hLHgI=

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 before
hashing
9      * @return der Base64 encoded HMAC hashes
10     * @throws UnsupportedOperationException
11     * @throws NoSuchAlgorithmException
12     * @throws InvalidKeyException
13     */
14     public static String hmacHashOverKey(String dataPassword, String
publicKeyBase64)
15     throws UnsupportedOperationException, NoSuchAlgorithmException,
InvalidKeyException {
16         String hmacHashBase64 = "";
17         // HMAC Step 1: convert datapassword to byte[]
18         byte[] dataPasswordBytes = dataPassword.getBytes("UTF-8");
19         // HMAC Step 2: convert RSA public key to byte[]
20         byte[] publicKeyBytes = Base64.getDecoder().decode(publicKeyBase64);
21         // HMAC Step 3: create and initialize javax.crypto.Mac
22         // i). create hmacSha secretKey from Datapassword
23         SecretKeySpec hmacKey = new SecretKeySpec(dataPasswordBytes,
HMAC_SHA256_ALGORITHM);
24         // ii) instantiate and initialize mac
25         Mac mac = Mac.getInstance(HMAC_SHA256_ALGORITHM);
26         mac.init(hmacKey);
27         // HMAC Step 4: calculate HMAC hash bytes
28         mac.update(publicKeyBytes);
29         byte[] hmacHashBytes = mac.doFinal();

```

```

30         // HMAC Step 5: convert hmac bytes to base64 form
31         hmacHashBase64 = Base64.getEncoder().encodeToString(hmacHashBytes);
32         return hmacHashBase64;
33     }

```

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

```

1      /**
2       * Processing of an SCR get request (unencrypted answer)
3       *
4       * The result is unencrypted because the headers x-scr-key and x-scr-keyhash
are
5       * not be set. Note: The production environment suppresses unencrypted result
6       * querys
7       */
8      @Test
9      void testGetCasesUnencrypted() {
10         out("JUNIT Test testGetCasesUnencrypted");
11         String ret = ScrHttpRequestHandler.callScr(scr_url_full_all, scr_user,
scr_password);
12         out(ret);
13         assertThatNoException();
14         assertTrue(ret.startsWith("["));
15     }

```

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

Required Data for the Request

```

1      /* keyLength of the RSA key pair used in the test ( 3072 or 4096 Bit) */
2      static int key_length = 3072;
3      /* Username for Basic Auth */
4      static String scr_user = "<your username>";
5      /* Password for Basic Auth */
6      static String scr_password = "<your password>";
7      /* Data password for HMAC calculation */
8      static String scr_datapassword = "<your datapassword>";
9      /* clientid, used as request parameter */
10     static String scr_clientid = "<your clientid>";
11     /*
12      * Host of the SCR endpoint postident-itu.deutschepost.de (test environment) or
13      * postident.deutschepost.de (productive system)
14      */
15     static String scr_host = "postident-itu.deutschepost.de";
16     /* URL for the SCR GET request getting all cases for clientid */
17     static String scr_url_full_all = "https://" + scr_host + "/api/scr/v1/" +
18     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

```

1      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

```

1      /**
2      * Processing of an SCR get request (encrypted response without decryption)
3      *
4      * The result is encrypted because the headers x-scr-key and x-scr-keyhash are
5      * set. no decryption takes place in this test, the first ones Characters of
the
6      * encrypted response are output on the console
7      */
8      @Test
9      void testGetCasesEncrypted() throws ParseException, JOSEException, ScrException
{

```

```

10         out("JUnit Test testGetCasesEncrypted");
11         String ret = ScrHttpRequestHandler.callScr(scr_url_full_all, scr_user,
scr_password, pubkey_base64, hmac_hash);
12         out(ret.substring(0, 80) + " ...");
13         assertThatNoException();
14         assertTrue(ret.startsWith("eyJlbmMi")); // base64 representation of
'{"enc":""'
15     }
16

```

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

Calculate the required Parameter

```

1     hmac_hash = ScrEncryptionHandler.hmacHashOverKey(scr_datapassword, pubkey_base64);
2     // use KeyPairGenerator to generate RSA keypair
3     java.security.KeyPairGenerator keyGen =
java.security.KeyPairGenerator.getInstance("RSA");
4     keyGen.initialize(key_length);
5     // generate keypair
6     keypair = keyGen.genKeyPair();
7     // store keys in base64 format
8     pubkey_base64 = Base64.encodeBase64String(keypair.getPublic().getEncoded());
9     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 Data for the Request

```

1     /**
2     * Processing of an SCR get request (encrypted response with decryption)
3     *
4     * The result is encrypted because the headers x-scr-key and x-scr-keyhash are
5     * set. The decrypted payload of the Encrypted Response is sent to Console
6     * output
7     */
8     @Test
9     void testGetCasesEncryptedWithDecrypt() throws ParseException, JOSEException,
ScrException {
10         out("JUnit Test testGetCasesEncryptedWithDecrypt");
11         String ret = ScrHttpRequestHandler.callScr(scr_url_full_all, scr_user,
scr_password, pubkey_base64, hmac_hash);
12         out(ret); //out(ret.substring(0, 80) + " ...");
13         String decrypted = ScrEncryptionHandler.decryptPayload(ret,
privkey_base64);
14         assertThatNoException();
15         assertTrue(ret.startsWith("eyJlbmMi")); // base64 representation of
'{"enc":""'
16         assertTrue(decrypted.startsWith("["); // begin of json array

```

```

17         out(decrypted);
18     }
19
20

```

3.3.8 Sample Response of SCR getCases full:

 This json data corresponds to the scr specification April 2023

```

1  [
2      {
3          "caseId": "K6JNXGBG2XVU",
4          "caseStatus": {
5              "status": "closed",
6              "archived": false,
7              "validUntil": "2023-03-23T14:02:04+01:00",
8              "created": "2023-03-09T14:02:04+01:00",
9              "modified": "2023-03-09T15:11:39+01:00"
10         },
11         "orderData": {
12             "customData": {},
13             "processData": {
14                 "targetCountry": "DEU",
15                 "preferredLanguage": "DE_DE",
16                 "referenceId": "K6JNXGBG2XVU",
17                 "callbackUrlCouponRequested": {},
18                 "callbackUrlReviewPending": {},
19                 "callbackUrlIncomplete": {},
20                 "callbackUrlSuccess": {},
21                 "callbackUrlDeclined": {}
22             },
23             "contactDataProvided": {},
24             "identificationDocumentProvided": {},
25             "drivingLicenceProvided": {
26                 "drivingLicenceClasses": []
27             }
28         },
29         "contactData": {
30             "title": {},
31             "firstName": {
32                 "status": "new",
33                 "value": "Matthias"
34             },
35             "lastName": {
36                 "status": "new",
37                 "value": "Schulz"
38             },
39             "mobilePhone": {},
40             "email": {
41                 "status": "new",
42                 "value": "terter.alpha@email.de"
43             },
44             "epost": {},
45             "address": {
46                 "streetAddress": {},
47                 "appendix": {},
48                 "postalCode": {}

```

```

49         "city": {},
50         "country": {}
51     },
52 },
53 "identifications": [
54     {
55         "identificationMethod": "basic",
56         "identificationStatus": {
57             "status": "success",
58             "identificationTime": "2023-03-09T15:11:38+01:00",
59             "created": "2023-03-09T14:02:19+01:00",
60             "modified": "2023-03-09T15:11:39+01:00"
61         },
62         "identificationDocument": {
63             "type": {
64                 "status": "new",
65                 "value": "1"
66             },
67             "number": {
68                 "status": "new",
69                 "value": "sdadasdsa"
70             },
71             "firstName": {
72                 "status": "new",
73                 "value": "Matthias"
74             },
75             "lastName": {
76                 "status": "new",
77                 "value": "Mustermann"
78             },
79             "birthName": {},
80             "birthDate": {
81                 "status": "new",
82                 "value": "1977-01-11"
83             },
84             "birthPlace": {
85                 "status": "new",
86                 "value": "Paddington"
87             },
88             "nationality": {
89                 "status": "new",
90                 "value": "DEU"
91             },
92             "address": {
93                 "streetAddress": {
94                     "status": "new",
95                     "value": "Sackgasse 1"
96                 },
97                 "appendix": {},
98                 "postalCode": {
99                     "status": "new",
100                     "value": "12345"
101                 },
102                 "city": {
103                     "status": "new",
104                     "value": "Rostock"
105                 },
106                 "country": {
107                     "status": "new",

```

```

108         "value": "DEU"
109     }
110 },
111 "dateIssued": {
112     "status": "new",
113     "value": "2020-11-11"
114 },
115 "dateOfExpiry": {
116     "status": "new",
117     "value": "2027-11-12"
118 },
119 "authority": {
120     "status": "new",
121     "value": "Sas"
122 },
123 "placeOfIssue": {},
124 "countryOfDocument": {
125     "status": "new",
126     "value": "DEU"
127 },
128 "records": [
129     {
130         "recordId": "935254843",
131         "fileName": "K6JNXGBG2XVU_idimage_1.jpg",
132         "belongsTo": "identificationdocument",
133         "type": "idimage",
134         "mimeType": "image/jpeg"
135     }
136 ],
137 },
138 "records": [
139     {
140         "recordId": "935254842",
141         "fileName": "K6JNXGBG2XVU_usersignature.jpg",
142         "belongsTo": "method",
143         "type": "usersignature",
144         "mimeType": "image/jpeg"
145     }
146 ],
147 "additionalDataBasic": {
148     "couponDownloadCount": 1,
149     "couponDownloadLastTimestamp": "2023-03-09T14:02:20+01:00",
150     "postOfficeStreetAddress": "Platz der Deutschen Post",
151     "postOfficeCity": "Bonn"
152 }
153 },
154 ],
155 "accountingData": {
156     "accountingNumber": "506606638437A2",
157     "accountingProduct": "Postident Basic Zusatzprodukt 2"
158 }
159 }
160 ]

```

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

```

1 // 400: Base64 format error in keyhash
2 {"apiversion":"v1","errors":[{"errorcode":"90104","reason":"base64 error","key":"x-
3 scr-keyhash","message":"Base64 format error."}]}
4
5 // 400: Base64 format error in key Error:
6 400 {"apiversion":"v1","errors":[{"errorcode":"90104","reason":"base64
7 error","key":"x-scr-key","message":"Base64 format error."}]}
8
9 // 400: No keyhash provided
10 {"apiversion":"v1","errors":[{"errorcode":"90106","reason":"missing
11 keyhash","key":"x-scr-keyhash","message":"No keyhash value provided in header x-
12 scr-keyhash."}]}
13
14 // 400: wrong keyhash
15 {"apiversion":"v1","errors":[{"errorcode":"90107","reason":"hash
16 failure","key":"","message":"Provided encryption key does not match keyhash.
17 Possible reasons: wrong data password or x-scr-key has been manipulated."}]}
18
19 // 400: invalid RSA public Key
20 {"apiversion":"v1","errors":[{"errorcode":"90109","reason":"invalid key","key":"x-
21 scr-key","message":"Invalid RSAPublicKey format for encryption key."}]}
22
23 // 401: invalid credentials
24 {"apiversion":"v1","errors":
25 [{"errorcode":"90114","reason":"unauthorized","key":"Authorization","message":"Auth
26 orization failed."}]}
27
28 // 403: wrong clientid
29 {"apiversion":"v1","errors":
30 [{"errorcode":"90127","reason":"forbidden","key":"Authorization","message":"User
31 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.

HTTP Status	Errorcode	Reason	Key	Message
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
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 click 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 version

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/
26      *      v1/865E6E37/cases
27      * @param authString for BASIC authentication
28      * @param rsaPubkey
29      *      rsa public key in base64 form
30      * @param rsaKeyhash
31      *      hmac hash over rsa pubkey in base64 form
32      * @return the SCR Response
33      */
34     public String callScr(String scrUrl, String authString, String rsaPubkey,
35                          String rsaKeyhash) {
36         int responseCode = -1;
37         StringBuilder sbRet = new StringBuilder();
38         if (scrUrl == null || scrUrl.isEmpty()) {
39             out("Error: URL not scecified.");
40             sbRet.append("Error: URL not scecified.");
41             return sbRet.toString();
42         }
43         try {
44             // SCR flow step 3.2 instantiate anf configure URL connection
45             URL url = new URL(scrUrl);
46             HttpURLConnection huc = (HttpURLConnection) url.openConnection();
47             huc.setRequestMethod("GET");
48             huc.setConnectTimeout(30000);
49             huc.setReadTimeout(30000);
50             huc.setRequestProperty("User-agent", "SCR-CLIENT");
51             huc.setRequestProperty("Content-Type", "application/json");
52             out("Info: " + "HEADER User-agent: SCR-CLIENT ");
53             // SCR flow step 3.3 set Authorization Header
54             huc.setRequestProperty("Authorization", authString);
55             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
// eindeutig - Stacktrace sinnlos
88         String msg = "ConnectException. " + e.getMessage();
89         out("Error: " + msg, e);
90         responseCode = -6;
91         sbRet.append(msg);
92     } catch (javax.net.ssl.SSLHandshakeException e) { // NOSONAR squid:S1166
// fuer
// SSLHandshakeException
// ist der
// Stacktrace-Inhalt
// ohne Belang
93         String msg = "SSLHandshakeException beim Senden. " + e.getMessage();
94         out("Error: " + msg, e);
95         responseCode = -7;
96         sbRet.append(msg);
97     } catch (Throwable e) { // NOSONAR
//
100         checkstyle:com.puppycrawl.tools.checkstyle.checks.coding.IllegalCatchCheck
// 3rdParty (HTTP) Calls mit diversen
// Exceptionfaellen
101         String msg = "Error during scr request. " + e.toString();
102         out("Error: " + msg);
103         responseCode = -8;
104     }

```

```

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)
128     throws IOException {
129     StringBuilder sb = new StringBuilder();
130     if (istr == null && supressException) {
131         return "";
132     }
133     if (istr == null) {
134         throw new IllegalArgumentException("istr must not be null");
135     }
136     BufferedReader in = null;
137     try {
138         in = new BufferedReader(new InputStreamReader(istr, "UTF-8"));
139         String row = "";
140         while ((row = in.readLine()) != null) {
141             sb.append(row);
142         }
143         in.close();
144     } catch (IOException e) {
145         out("Error: readInputStream() Fehler beim Lesen eines HTTP Streams. {0}",
146             e);
147         if (!supressException) {
148             throw e;
149         }
150     } finally {
151         if (in != null) {
152             in.close();
153         }
154     }
155     return sb.toString();
156 }
157 /**
158  * Call scr without encryption.
159  * calls {@link #callScr(String, String, String, String, String)} with emphy key
160  * and empty keyhash.
161  *
162  * @param scrUrl
163  * @param authString
164  * @return
165  */
166 public String callScr(String scrUrl, String authString) {
167     return callScr(scrUrl, authString, "", "");
168 }

```

```

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 }

```

JWT Bearer version:

This variant uses a pre-issued Keycloak access token instead of Basic Auth.

The client sends the token in the Authorization header as Bearer token.

On the backend, the token is validated against the configured JWT endpoint.

Required SCR roles are read from resource_access.scr-ident-oauth.roles or resource_access.scr-ident.roles.

Client IDs are read from the client_ids claim.

The bearer token must be a valid Keycloak-issued JWT and must contain the roles required by the SCR endpoint. Roles are read from resource_access.scr-ident-oauth.roles (or resource_access.scr-ident.roles), and client IDs are read from client_ids. Authentication is independent of the SCR encryption headers x-scr-key and x-scr-keyhash, which are still used for encrypted SCR responses.

callScrWithBearerToken

```

1  /**
2   * Calls the SCR service using JWT Bearer authentication and returns the response
3   * as a String.
4   *
5   * @param scrUrl      URL to the SCR service, e.g. https://.../api/scr/v1/.../cases
6   * @param bearerToken JWT access token used for Bearer authentication
7   * @param rsaPubkey   RSA public key in Base64 form
8   * @param rsaKeyhash  HMAC hash over the RSA public key in Base64 form
9   * @return the SCR response or an error message
10  */
11  public String callScrWithBearerToken(String scrUrl, String bearerToken, String
12  rsaPubkey, String rsaKeyhash) {
13      int responseCode = -1;
14      StringBuilder sbRet = new StringBuilder();
15
16      // Validate required input
17      if (scrUrl == null || scrUrl.isEmpty()) {
18          out("Error: URL not specified.");
19          return "Error: URL not specified.";
20      }
21
22      try {
23          // Create the HTTP connection to the SCR endpoint
24          URL url = new URL(scrUrl);
25          HttpURLConnection huc = (HttpURLConnection) url.openConnection();

```

```

24
25 // Configure request
26 huc.setRequestMethod("GET");
27 huc.setConnectTimeout(30000);
28 huc.setReadTimeout(30000);
29 huc.setRequestProperty("User-agent", "SCR-CLIENT");
30 huc.setRequestProperty("Content-Type", "application/json");
31
32 // Set JWT Bearer authentication header
33 huc.setRequestProperty("Authorization", "Bearer " + bearerToken);
34
35 huc.setDoOutput(false);
36 huc.setDoInput(true);
37
38 // Optional SCR headers for encrypted requests
39 if (rsaPubkey != null && !rsaPubkey.isEmpty()) {
40     huc.setRequestProperty("x-scr-key", rsaPubkey);
41 }
42 if (rsaKeyhash != null && !rsaKeyhash.isEmpty()) {
43     huc.setRequestProperty("x-scr-keyhash", rsaKeyhash);
44 }
45
46 // Execute request and read response
47 responseCode = huc.getResponseCode();
48 if (responseCode == 200) {
49     sbRet.append(readInputStream(huc.getInputStream(), true));
50 } else {
51     sbRet.append("Error: ")
52         .append(responseCode)
53         .append(" ")
54         .append(readInputStream(huc.getErrorStream(), true));
55 }
56
57 return sbRet.toString();
58 } catch (Exception e) {
59     return "Error during SCR request. " + e.getMessage();
60 }
61 }

```

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;
16 import com.nimbusds.jose.JWEObject;
17 import com.nimbusds.jose.Payload;
18 import com.nimbusds.jose.crypto.RSADecrypter;
19 /**
20  * ScrEncryptionHelper cryptographic methods required for scr call handling
21  *
22  * @author Deutsche Post AG
23  * @version 1.0
24  */
25 public class ScrCryptoHelper {
26     /** HMAC Hash Algorithm to use. */
27     private static final String HMAC_SHA256_ALGORITHM = "HmacSHA256";
28     /**
29      * SCR flow step 6: decrypt data with private key.
30      *
31      * @param encryptedPayload
32      *      data to decrypt.
33      * @param rsaPrivateKeyBase64
34      *      base64 encoded rsy private key, used to decrypt payload
35      * @return decrypted payload as string
36      * @throws ParseException
37      * @throws JOSEException
38      * @throws ScrException
39      */
40     public static String decryptPayload(String encryptedPayload, String
rsaPrivateKeyBase64)
41         throws ParseException, JOSEException, ScrException {
42         // SCR flow step 6.1: parse encrypted payload into JWEObject
43         JWEObject jweObject = JWEObject.parse(encryptedPayload);
44         out("SCR flow 6.1: parsed JWE header" +
jweObject.getHeader().toJSONObject().toJSONString());
45         try {
46             // SCR flow step 6.2 and 6.3: {@link #createRSADecrypter(String) }
47             // SCR flow step 6.4: decrypt JWEObject
48             jweObject.decrypt(createRSADecrypter(rsaPrivateKeyBase64));
49         } catch (JOSEException e) {
50             if (e.getMessage().contains("Illegal key size")) {
51                 out("Error: "
52                     + "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}",
53                     e);
54             } else {
55                 out("Error during decryprPayload. {0}", e);
56             }
57         }
58         // SCR flow step 6.5: get decrypted responsestring
59         Payload payload = jweObject.getPayload();
60         return payload.toString();
61     }
62     /**
63      * Instatiates an RSA decrypter from rsaprivate key in bas64 form.
64      * SCR flow 6.3: instantiate RSADecrypter
65      * @param rsaPrivateKey

```

```

66     * @return
67     * @throws ScrException
68     */
69     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             //
88             checkstyle:com.puppycrawl.tools.checkstyle.checks.coding.IllegalCatchCheck
89             // 3rdParty Calls
90             out("Warning: " + "PrivateKey creation error. Base64 conversion error.
{0}", e);
91             throw new ScrException("PrivateKey creation error. Base64 conversion
error.", e);
92         }
93         RSAPrivateKey retKey;
94         try {
95             retKey = byteToPrivateKey(keybytes);
96         } catch (InvalidKeySpecException | NoSuchAlgorithmException e) {
97             out("Warning: " + "PrivateKey creation error {0}", e);
98             throw new ScrException("PrivateKey creation error.", e);
99         }
100        return retKey;
101    }
102    /**
103    * Convert a ByteArray to RSAPrivateKey.
104    * Part of SCR flow 6.2: instantiate RSAPrivateKey
105    *
106    * @param keybytes
107    *         Bytes of Key.
108    * @return the RsaPrivateKey
109    * @throws InvalidKeySpecException
110    * @throws NoSuchAlgorithmException
111    */
112    public static RSAPrivateKey byteToPrivateKey(byte[] keybytes)
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 UnsupportedOperationException
125     * @throws NoSuchAlgorithmException
126     * @throws InvalidKeyException
127     */
128     public static String hmacHashOverKey(String dataPassword, String
publicKeyBase64)
129     throws UnsupportedOperationException, 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     * calculates HTTP Basic Authorization String (SCR flow step #2)
154     *
155     * @param username
156     * @param password
157     * @return the Authorization string
158     * @throws UnsupportedOperationException
159     */
160     public static String calcBasicAuthString(String username, String password)
throws UnsupportedOperationException {
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} ",
ScrCryptoHelper.toHexString(authString.getBytes("UTF-8")));
164         authString =
Base64.getEncoder().encodeToString(authString.getBytes("UTF-8"));
165         out("SCR flow 2.2: Basic Auth usernamepassword base64: {0} ", authString);
166         authString = "Basic " + authString;
167         out("SCR flow 2.3: complete Basic Auth string: {0} ", authString);
168         return authString;
169     }
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;
15  import org.junit.jupiter.api.BeforeAll;
16  import org.junit.jupiter.api.Test;
17
18  import com.nimbusds.jose.JOSEException;
19
20  /**
21   * Scr Call JUnit Tests
22   *
23   * JUNIT tests to demonstrate SCR result data retrieval with decryption
24   *
25   * @author Deutsche Post AG
26   * @version 1.0
27   */
28  public class ScrCallTests {
29      /* keyLength of the RSA key pair used in the test ( 3072 or 4096 Bit) */
30      static int key_length = 3072;
31      /* Username for Basic Auth */
32      static String scr_user = "<your username>";
33      /* Password for Basic Auth */
34      static String scr_password = "<your password>";
35      /* Data password for HMAC calculation */
36      static String scr_datapassword = "<your datapassword>";
37      /* clientid, used as request parameter */
38      static String scr_clientid = "<your clientid>";
39      /*
40       * Host of the SCR endpoint postident-itu.deutschepost.de (test environment) or
41       * postident.deutschepost.de (productive system)
42       */
43      static String scr_host = "postident-itu.deutschepost.de";
44      /* URL for the SCR GET request getting all cases for clientid */
45      static String scr_url_full_all = "https://" + scr_host + "/api/scr/v1/" +
scr_clientid
46          + "/cases/full";
47      /* The SCR key pair used in the tests */
48      static KeyPair keypair;
49      /* Base64 representation of the public key of the above key pair */
50      static String pubkey_base64;
51      /* Base64 representation of the private key of the key pair above */
52      static String privkey_base64;
53      /*
54       * the HMAC hash of the public key (calculated with the scr_datapassword as
55       * secret)
56       */
57      static String hmac_hash;
58
59      /**
60       * Generation of the RSA key pair used in the tests and calculation of the HMAC
61       * hash
62       *
63       */
64      @BeforeAll
65      public static void inititalizeAll()

```

```

66         throws NoSuchAlgorithmException, InvalidKeyException,
        UnsupportedEncodingException {
67             initializeKeypair();
68             hmac_hash = ScrEncryptionHandler.hmacHashOverKey(scr_datapassword,
        pubkey_base64);
69         }
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 initializeKeypair() throws NoSuchAlgorithmException {
76             // use KeyPairGenerator 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         * queries
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
out(decrypted);
136     }
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	<code>public static String decryptRSA(String jweString, String rsaPrivKey) throws</code>
2	<code>InvalidKeySpecException, NoSuchAlgorithmException, ParseException, JOSEException{</code>
3	<code>String ret = "";</code>
4	<code>RSAPrivateKey rpk = base64ToPrivateKey(rsaPrivKey) ;</code>
5	<code>JWEObject jweObject = JWEObject.parse(jweString);</code>
6	<code>jweObject.decrypt(new RSADecrypter(rpk));</code>
7	<code>ret = jweObject.getPayload().toString();</code>
8	<code>return ret;</code>
9	<code>}</code>
10	<code>public static RSAPrivateKey base64ToPrivateKey(String base64Bytes) throws</code>
11	<code>InvalidKeySpecException, NoSuchAlgorithmException{</code>
12	<code>byte[] keybytes = Base64.getDecoder().decode(base64Bytes);</code>
13	<code>return (RSAPrivateKey) KeyFactory.getInstance("RSA").generatePrivate(new</code>
	<code>PKCS8EncodedKeySpec(keybytes));</code>
	<code>}</code>

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	
1	<code>/*</code>
2	<code>SCR PHP client sample code</code>
3	<code>*/</code>
4	<code>this sample uses php seclib 3</code>
5	<code>see https://phpseclib.com/docs/install for installation instructions</code>
6	<code>*/</code>
7	<code><?php</code>
8	<code></code>
9	<code>include 'c:\php\vendor\autoload.php';</code>
10	<code>use phpseclib3\Crypt\PublicKeyLoader;</code>
11	<code>use phpseclib3\Crypt\AES;</code>
12	<code>use phpseclib3\Crypt\RSA;</code>
13	<code></code>
14	<code>// scr credentials</code>
15	<code>\$user_name = "<your user>";</code>
16	<code>\$password = "<your password>";</code>
17	<code>\$clientid = "<your clientid>";</code>
18	<code>\$data_password = "<your datapassword>";</code>
19	<code>// possible hosts: postident-itu.deutschepost.de (test) or</code>
	<code>postident.deutschepost.de (production)</code>

```

20 $url_API = 'https://postident-itu.deutschepost.de/api/scr/v1/' . $clientid . '/
cases/full';
21
22 // Generate key pair with 3k size
23 $private_key = RSA::createKey(3072);
24 $public_key = $private_key->getPublicKey();
25
26 $ch = curl_init();
27 curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
28 curl_setopt($ch, CURLOPT_SSL_VERIFYSTATUS, false);
29 // for error analysis enable curl verbose
30 // curl_setopt($ch, CURLOPT_VERBOSE, true);
31
32
33 $headers = [
34     'Content-type: application/JSON; CHARSET=UTF-8',
35     'Authorization: Basic ' . base64_encode("$user_name:$password"),
36     'x-scr-enc: A256GCM', // PHP SecLib3 only supports GCM
37     'x-scr-alg: RSA-OAEP-256',
38     'x-scr-key: ' . extractKeyFromCert($public_key),
39     'x-scr-keyhash: ' . base64_encode(hash_hmac('sha256',
base64_decode(extractKeyFromCert($public_key)), $data_password, TRUE))
40 ];
41 var_dump('headers ', $headers );
42
43 curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
44 curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
45 curl_setopt( $ch, CURLOPT_URL, $url_API );
46
47 $result = curl_exec( $ch );
48 if ($result === FALSE) {
49     printf("cUrl error (%#d): %s<br>\n",
50         curl_errno($ch),
51         htmlspecialchars(curl_error($ch)))
52     ;
53 }
54 echo " R E S U L T :\r\n";
55 var_dump($result);
56 $decrypted = jweRsaDecryptFromBase64Token($private_key, $result);
57 var_dump($decrypted);
58
59 // cert is surrounded by BEGIN/END text - this has to be removed
60 function extractKeyFromCert($cert) {
61     $key = preg_replace('/\\-\\-\\-\\-\\.\\*\\-\\-\\-\\-\\/',' ', $cert); // remove
the -----BEGIN CERTIFICATE----- and -----END CERTIFICATE----- stuff
62     $key = preg_replace("/[\n\r]/", ' ', $key); // remove new lines
63     return $key;
64 }
65
66 function jweRsaDecryptFromBase64Token($rsaPrivateKey, $jweTokenBase64)
67 {
68     list($jweHeaderBase64, $jweEncryptedKeyBase64, $nonceBase64,
$encryptedDataBase64, $gcmTagBase64) = explode('.', $jweTokenBase64, 5);
69
70     $keyAlg = getJweHeaderKeyAlg($jweHeaderBase64);
71     $encryptionAlg = getJweHeaderEncryptionAlg($jweHeaderBase64);
72     $aesDecryptionKey = "";
73     switch ($keyAlg) {
74         case 'RSA-OAEP-256':

```

```

75         {
76             $rsa = PublicKeyLoader::load($rsaPrivateKey)
77                 ->withHash('sha256')
78                 ->withMGFHash('sha256');
79             $aesDecryptionKey =
$rsa->decrypt(decodeBase64Url($jweEncryptedKeyBase64));
80             break;
81         }
82         default:
83         {
84             // Alg nicht unterstützt
85             return '*** Alg nicht unterstützt ***';
86         }
87     }
88     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, prepares and fires a request and decrypts the response .

SCR Python Client

```
1  '''
2
3  Postident SCR Python Reference Client
4
5  Created on 26.06.2022
6
7  @author: @author Deutsche Post AG
8
9  # RSA
10 # https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/
11
12
13 run pip install cryptography
14 run pip install jwcrypto
15
16 '''
17
18 import base64
19 import http.client
20
21 from cryptography.hazmat.backends import default_backend
22 from cryptography.hazmat.primitives import hashes, hmac
23 from cryptography.hazmat.primitives import serialization
24 from jwcrypto import jwk, jwe
25 from jwcrypto.common import json_decode
26
27
28 #HTTP CONFIGURATION
29 HTTP_USER_AGENT="SCR-CLIENT"
30 HTTP_CONTENT_TYPE="application/json"
31 # ! please take care to use the right hostname
32 POSTIDENT_HOSTNAME_STR="postident-itu.deutschepost.de"
33
34
35 #ENCRYPTION CONFIGURATION
36 SCR_ENC_ALG="RSA-OAEP-256"
37 SCR_ENC="A256CBC-HS512"
38
39
40 # CLIENT CONFIGURATION
41 USERNAME_STR = "<your username here>"
42 PASSWORD_STR = "<your password here>"
```

```

43 CLIENTID_STR = "<your clientid here>"
44 DATAPASSWORD_STR = "<your datapassword here>"
45
46 rsa_private_key_loaded3k_base64=\
47
48 "MIIG4gIBAAKAYEApmc13IbTFltQJyH7CRipWfK4ibMikIxeDV01XjpndbBNHDFk0C+jZLnzNGpy3BvaR
49 r1TqWbV"\
50 "06paOazw6v2iocaNEGYA+6bxwf7rKhmrAYzce91E8bzIgbmyLAHU/
51 d0yPI3lcxqL01QxaF6gi+uSA88Q5loSK7VCp"\
52 "S0bb9sXG02iJL40p0An9sln6/8NXxyu41tA/GxMTmbSWjH8VlRc09xr7/ja88aIHIMFjgG9+yr/
53 IPyr6i0kt0ciF"\
54 "vT+scEseVTxVU7adS0rQPW00D/yjspIqwHr9Wwqv0DGY8/
55 ha+mY2A0zaYUWYoWS7MnfFaBsDHL01z4vwdl0BwE3Y2"\
56 "lCkrIK6IohkV2RqB5pB7nUrUAzEiDy23Cv31DdF1fJ0R2mW/
57 GeTQW6oQL7n5pRUsdmsiwVy7pGG1STUJ6qmZgsQsN"\
58
59 "vYkxWrkbn+C6qSK+xjAs3eqJsSrmIvb0PLYVMWmXXbVvRvonm4KgumCpJGZ4f+1KeI92MSMKekxdy4AClH
60 BAgMBAA"\
61 "ECggGAJYW8dX/JdY8dodVdwA1ZkrjSt+o27xVX1/
62 V6Pkc5MSaSHLzRQYML9NKdhmf4u7EcjyP+C4Lu0I3nTn4azU"\
63
64 "Jo08aD5Kc7tANPHImZCa0LHepfhWa9tyz0oZv3+0hme0A6BcGRbcUcpoi jo39DAIocINO+YdMJqeVzo71F
65 vZFPwo5"\
66 "c86APudMTr+ldd6/JmhrjrMzU+JaLFfRfMFCV5WxYmvjDT/
67 gbFiJbJ9nPs+rqV1l85rHyNX37SZdC1yWG7iR7k06wg"\
68
69 "lMi9oI1KM2o40QsTMI+zyxDushE3w6lZPPxxNaawq0JdpJfzsEhRGJz0IS2UZZ50t02x1zImh0o5uVwXy
70 HeMMnNC"\
71 "vowFo+BvTeAbFlDn5cXpEWx+mX7i2Q8qPLT5/rvYxjSjRvAUhhErvuF15/TFzhXDe/
72 J0gFuXVzkAtAJeCOVfgrkk9Q"\
73 "vJ9HkzF/
74 q7WltAKE0EHY5bB1J1e0d0fnDg5hJvsQcSt+89LbNFQ35MeyPX54mvyY3oWw8I7XpAoHBANczIViDxLmB"\
75
76 "ej20GmVTG+1Cl7h7xyeQDetWwARYBx4G+e6jxAXAzvRRXqSPNhwv7b3lVLUszWU5W4cAMEPlhhRceyhkXf
77 kEDBDiG"\
78 "CvE0RjUCoJ7E+l6AAB+gSbUbpUqjRBLATinnwmnuY/
79 2X+00FUEl30nsXkaFhr2WfQLLehuTa6GUyAK0ieYmEMDcl0"\
80 "C+C5E5swMuSpmuszLeuGxgwaqY29JpDSBvc3FxmMYaMVMi/
81 VxeEsHvCJ6+wHyB9pvTwKBWQDF87RsBmjUIij0eq3"\
82 "aigXptwLjjuwXK0h9P0pF9hLDbKDn8U3tkcq5IICri4wf506AD/OrQlu/7fUZgjKBbzKchCgQxg/
83 JbB9cskHAsY0A"\
84 "+aHne0fRI+kNeI+Wpx6q8UFGHlcp0gSMHaabJE01PRbxbF4PUwY2cs0/kW/
85 P30h1hRxSxbQt3bNsB8Se75xzJ0oMy"\
86 "j2BF/pYM9g7uDEs1psyBoVg4UKLybHlqxEQgkq306eYd9EoMrT7h7u/
87 afPQ+28CgcAGJMiL7V8daKvju/3W7LN8Z4"\
88 "1LUAVUhNFQ6a4bsa0qYMQb3IMJIwMFijKqG4iQv/AKntR3Q6ste6C4TvIRziiwxh8h/
89 R0Nu2bYyIu17LewLMukCn1"\
90 "8CeacHQB06lWp3ogecrPB0U/aB7bNfFwquV8z9+/S/X0HkeJR4Uz6WnLG/
91 MNy3nuDUEIrLuSdj06og4RzBfGtYpHw"\
92 "p0MonSKovW5p/
93 2kvLZ6ZUXU7R0IT6naBQ3kvbajgg30EgvxNhugpexMCGcBEHITaHqJ3S4Gq+j9T00YT8u0CGq1Vk"\
94 "rfcT9d0VyVBp0EAReDYe6+7X1mmrQ978sTp/
95 5MiPGUbD0k8i7YBH1fUzwxSKTRJZDCuXo+NVAchm8N/uMWPSsHn5r"\
96
97 "HRbpN40iZZkBsBwsfZxmILZ4nslKa0T3FV6IVcusRr6AkHB5cKfy6ttGU42u3foBShc2Troan+2J+tCakU
98 LFKcydX"\
99 "xza382o20NjQFkVLq6Z+nhI1dDzC9n4ySP1BTs/J//F7ua3UCgcA/ipe/
100 YMbP8+N79+w0tWDJWMnwJ+tJ0tG9F3Bj"

```

```

71     "Mh2Se4KDQZbhgacReEo3jb6WQ2NxxELs13ciozZ996VwHTQ6LJHoRJZeN/
72     3ugUQkxItsMetHt23EbU1BFF6CPtel4"\
73     "MoVi85FxF/mhGvvy/Bc7ri1BaaKVHMriRpxqVJ+diXdfwK/
74     zLe+dwLhBd70vP38YhtTdbJ2dPsm2gW2h+0+g04NE4"\
75     "prWDKhjNr6EBmjw/3kQEhXF4MJ3FT5j13W0KKx3xQ="
76
77     rsa_private_key_loaded_bytes = base64.b64decode(rsa_private_key_loaded_base64)
78     rsa_private_key_loaded = serialization.load_der_private_key(
79         data=rsa_private_key_loaded_bytes ,
80         password=None
81     )
82
83     ''' OPTION:
84     to calculate rsa keypair use this
85     rsa_private_key_generated = rsa.generate_private_key(
86         public_exponent=65537,
87         key_size=3096,
88     )
89
90     # work with loaded private key - it will be more secure often to change the private
91     # key
92     rsa_private_key = rsa_private_key_loaded # alternatively:
93     rsa_private_key_generated
94     rsa_public_key = rsa_private_key.public_key()
95
96     # bring the private key in pem form
97     rsa_private_key_pem = rsa_private_key.private_bytes(
98         encoding=serialization.Encoding.PEM,
99         format=serialization.PrivateFormat.TraditionalOpenSSL,
100         encryption_algorithm=serialization.NoEncryption()
101     )
102
103     publicKeyDerBytes =
104     rsa_public_key.public_bytes( encoding=serialization.Encoding.DER, format =
105     serialization.PublicFormat.SubjectPublicKeyInfo)
106     public_key_der_base64 = base64.b64encode( publicKeyDerBytes ).decode("ascii")
107
108     print( "p4.3 get public key in byte[] form " , publicKeyDerBytes.hex())
109     print( "p4.4 get public key in Base64 String Form " , public_key_der_base64)
110     print( "p4.5 get private key in byte[] form " , rsa_private_key_pem.hex())
111     print( "p4.6 get private key " , rsa_private_key_pem)
112
113
114     # ALL STEPS
115     '''
116     1.1 convert datapassword to byte[]
117     '''
118     dataPasswordAsBytes = str.encode(DATAPASSWORD_STR, "utf-8")
119     datapassword_as_hexstring = dataPasswordAsBytes.hex()
120     print( "1.1 dataPasswordBytes " , DATAPASSWORD_STR, " ",
121     datapassword_as_hexstring)
122
123
124     1.2 convert RSA public key to byte[] and base64
125     '''
126     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");
129 mac.init(signingKey);
130 '''
131 hmacHashser = hmac.HMAC(dataPasswordAsBytes, hashes.SHA256(),
132 backend=default_backend())
133
134 1.4 calculate HMAC hash bytes'''
135 hmacHashser.update(publicKeyDerBytes)
136 hmacHashBytes = hmacHashser.finalize()
137
138 print( "1.4 hmacHashBytes" , hmacHashBytes.hex())
139 '''
140 1.5 convert hmac bytes to base64 form
141
142 String HMAC_HASH_BASE64 = Base64.getEncoder().encodeToString(hashBytes);
143 '''
144 hmac_hash_base64 = base64.b64encode( hmacHashBytes ).decode("ascii")
145 print( "1.5 HMAC_HASH_BASE64" , hmac_hash_base64 )
146
147
148 """Step 2: Calculate the basic authorization header
149 Code Sample
150 IN: username (String - precondition #2) password (String - precondition #2)
151 OUT: HTTP authorization header string
152 """
153
154
155 """2.1 concatenate username + ":" + password String userAndPass = username +
156 ":" + password; userAndPass =
157 SCRDEMO:3r#4Mu#GBRmP
158 """
159 userAndPass = USERNAME_STR + ":" + PASSWORD_STR
160 print( "2.1 userAndPass step 1" , userAndPass )
161
162 """2.2 convert the result of step 2.1 into base64 form userAndPass =
163 Base64.getEncoder().encodeToString(userAndPass.getBytes("UTF-8")); userAndPass =
164
165 U0NSREVNTzozciM0TXUjR0JSbVA=
166 """
167 userAndPass = base64.b64encode(str.encode(userAndPass, "utf-8")).decode("ascii")
168 #decode transforms bytearray to string
169 print( "2.2 userAndPass step 2 base64" , userAndPass )
170
171 """2.3 prepend "Basic " to authorization header value userAndPass = "Basic "
172 + userAndPass; userAndPass =
173 Basic U0NSREVNTzozciM0TXUjR0JSbVA=
174
175 2.4 when generating HTTP-request
176 httpURLConnection.setRequestProperty("Authorization", userAndPass);
177 Authorization: Basic U0NSREVNTzozciM0TXUjR0JSbVA=
178
179 Add authorization header to the request
180 """
181 HTTP_REQUEST_HEADERS = { 'Authorization' : 'Basic %s' % userAndPass }

```

```

173 print( "2.3 + 2.4 3 prepend \"Basic \" assign to Authorization" ,
      HTTP_REQUEST_HEADERS )
174
175
176 '''3.1
177
178 build SCR URL
179 String scrUrl = MessageFormat.format("https://{0}/api/scr/v1/{1}/cases",host,
      clientid);
180
181 https://postident.deutschepost.de/api/scr/v1/865E6E37/cases
182 '''
183 SCR_PATH = "/api/scr/v1/%s/cases" % CLIENTID_STR
184 print( "3.1 build scr url" , SCR_PATH )
185
186
187 '''3.2 instantiate and configure URL connection'''
188
189 http_connection = http.client.HTTPSConnection(POSTIDENT_HOSTNAME_STR)
190
191 HTTP_REQUEST_HEADERS["User-agent"]=HTTP_USER_AGENT
192 HTTP_REQUEST_HEADERS["Content-Type"]=HTTP_CONTENT_TYPE
193
194
195
196 '''
197 3.3 set Authorization Header (alredy done in 2.4)
198 3.4 set x-scr-key Header
199 3.5 set x-scr-keyhash Header
200 '''
201
202 HTTP_REQUEST_HEADERS["x-scr-alg"]=SCR_ENC_ALG
203 HTTP_REQUEST_HEADERS["x-scr-enc"]=SCR_ENC
204 HTTP_REQUEST_HEADERS["x-scr-key"]=public_key_der_base64
205 HTTP_REQUEST_HEADERS["x-scr-keyhash"]=hmac_hash_base64
206
207 print( "3.3, 3.4, 3.5 add x-scr header" , str(HTTP_REQUEST_HEADERS).replace(","," ",
      \n") )
208
209 payload = ""
210
211 '''
212 3.6 fire the Request
213 '''
214 http_connection.request("GET", "/api/scr/v1/%s/cases" % CLIENTID_STR , payload,
      HTTP_REQUEST_HEADERS)
215 print("send Request to : /api/scr/v1/%s/cases" % CLIENTID_STR)
216
217 ''' 4. Postident System is processing the Request '''
218
219 ''' 5. receive encrypted data '''
220 http_response = http_connection.getresponse()
221 encrypted_response = http_response.read()
222 print("returncode: %s" % http_response.getcode() )
223 print(encrypted_response)
224
225
226 ''' Step 6: decrypt data with private key
227 use JWE to decrypt

```

```

228
229     6.1 parse encrypted payload into JWEObjekt (implicit in 6.4)
230     6.2 instantiate RSAPrivateKey '''
231     jwk_private_key = jwk.JWK.from_pem(rsa_private_key_pem)
232     ''' 6.3 instantiate Decrypter '''
233     jwe_decrypt_token = jwe.JWE()
234     ''' 6.4. decrypt JWEObjekt'''
235     jwe_decrypt_token.deserialize(encrypted_response.decode("utf-8"), jwk_private_key)
236     ''' 6.5. get decrypted Response'''
237     payload = jwe_decrypt_token.payload
238     print(payload.decode('utf-8'))

```