SCR-Ident API Guide 3 Encryption

Contents

Changelog

Date	Change
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. Overview

The SCR result data can be accessed through GET operations of the resource *cases* (therefore see 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.

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 2 kbit is recommended (minimum 1 kbit)
  - the supported key length depends on the choose RSA algorithm (see TABELLE)
  - 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.

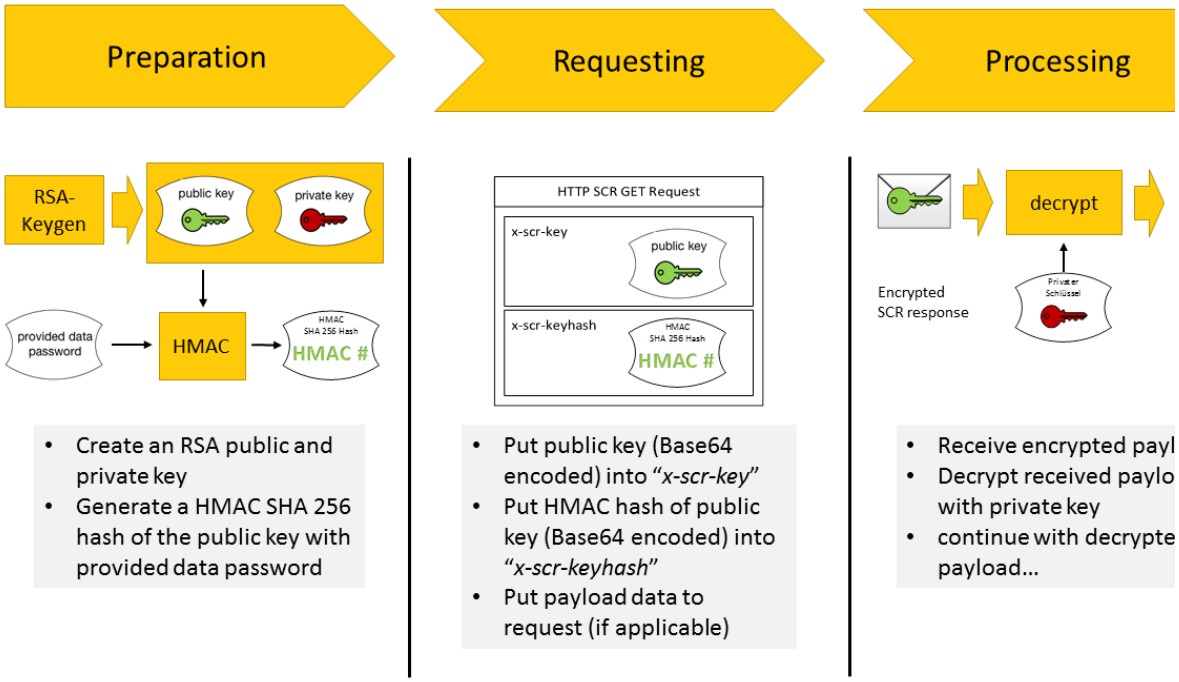
General Flow

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:



2. Encryption Settings using header

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	Submit here the Base64 encoded key for content encryption.
		To use RSA public key encryption you have to submit a RSA public key with a size of 2048, 3072 or 4096 Bits.
		
		Note:
		• **3072 bit RSA public key is recommended**.
		• **The support of 2048-bit keys will be discontinued at the end of 2022.**
			`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`

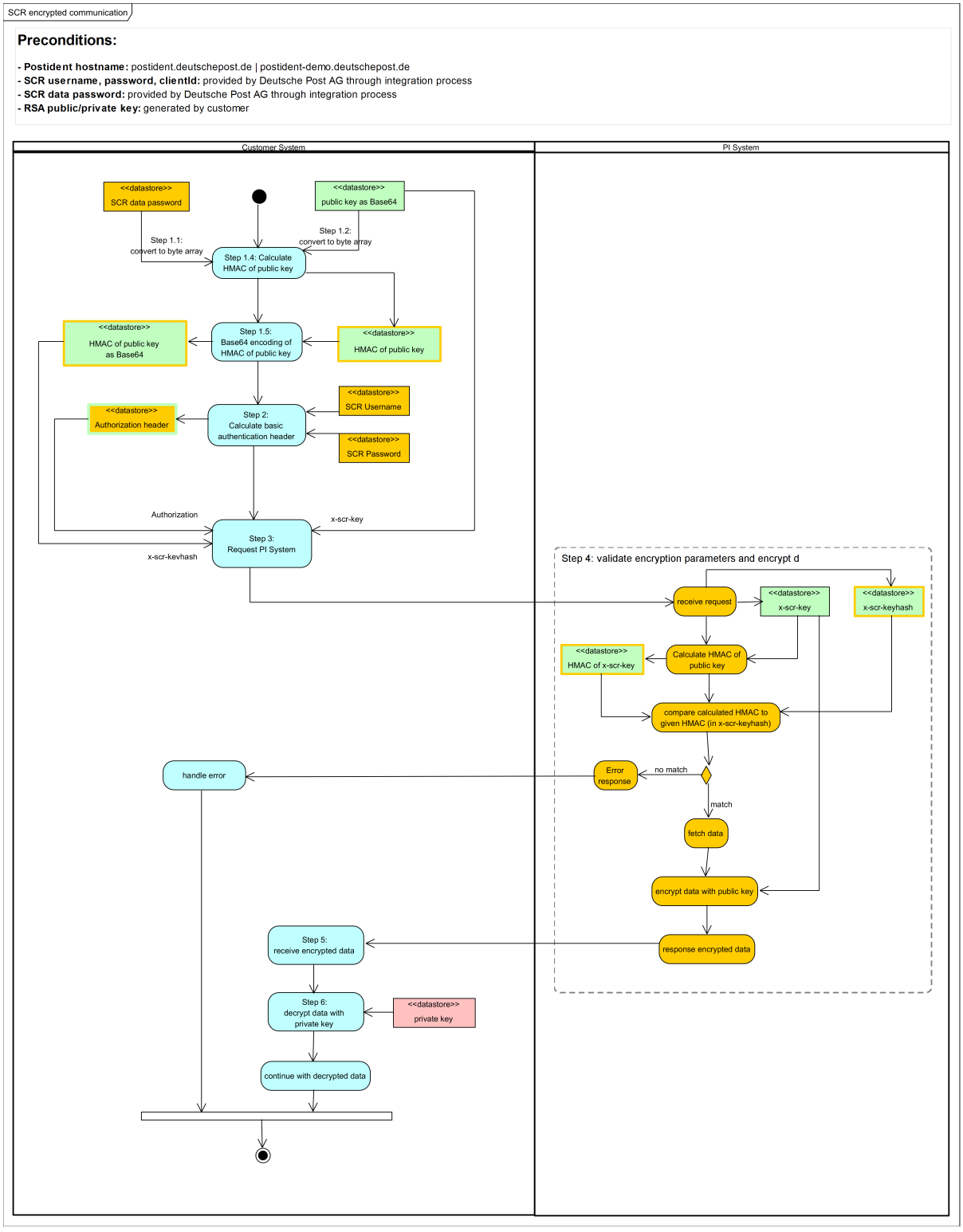
x-scr-alg	no	Algorithm for result encryption To use an asymmetric RSA based public key encryption choose one of the following values: • RSA1_5 (RSAES-PKCS1-V1_5 - RFC 3447) (**default value, but no longer recommended)** • RSA-OAEP (RSAES using Optimal Asymmetric Encryption Padding (OAEP) - RFC 3447) • RSA-OAEP-256 (RSAES using Optimal Asymmetric Encryption Padding (OAEP) - RFC 3447 **(recommended)** with the SHA-256 hash function and the MGF1 with SHA-256 mask generation function. Not supported by PHP SecLib.)	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).** • A192CBC-HS384 (AES_192_CBC_HMAC_SHA_384 authenticated encryption using a 384 bit key) • A128CBC-HS256 (AES_128_CBC_HMAC_SHA_256 authenticated encryption using a 256 bit key) • A256GCM (AES in Galois/Counter Mode (GCM) (NIST.800-38D) using a 256 bit key. Not supported by PHP SecLib.) • A192GCM (AES in Galois/Counter Mode (GCM) (NIST.800-38D) using a 192 bit key. Not supported by PHP SecLib.) • A128GCM (AES in Galois/Counter Mode (GCM) (NIST.800-38D) using a 128 bit key. Not supported by PHP SecLib.)	A256CBC-HS512

Standard Encryption

By default, the following parameters are used:

Property	Value	Description
JWE algorithm of the response	**RSA1_5** ⚠️ **Please note, that RSA1_5 is no longer recommended, please use RSA-OAEP-256**	RSAES-PKCS1-V1_5 (RFC 3447)
JWE encryption of the response	**A256CBC-HS512**	AES_256_CBC_HMAC_SHA_512 authenticated encryption using a 512 bit key

3. Detailed Flow with Samples



Precondition 1: Postident hostname

production: postident.deutschepost.de

Precondition 2: SCR username, password, clientId

The clientid an the credentials for basic authentication will provided by the Deutsche Post technical sales or service team.

Sample:

- Username: SCRDEMO
- Password: 3r#4Mu#GBRmP
- Clientid: 865E6E37

Precondition 3: SCR data password

The data password will provided by the Deutsche Post technical sales or service team.
This is the pre-shared-secret for HMAC calculation.

Sample: xR7ea2_53S(m

Precondition 4: RSA key pair

The keys can be generated before every call or stored in your system.
A key size of 2 kbit is recommended (minimum 1 kbit).

Sample RSA 3072 key pair

```
====================================== RSA Public Key ======================================
pubKeyBytes:
30 82 01 22 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00 03 82 01 0f 00 30 82 01 0a 02 82 01 01
00 d9 55 94 58 67 de 6d 8b 00 82 f3 a8 aa 4d 22 62 46 5b 84 90 95 05 51 43 59 ff a4 91 cd a4 21
42 db f8 96 14 17 e5 5d 44 a2 90 34 7c 5d d0 3e 33 95 43 78 f0 7a 19 7d 99 6a 17 d7 ea d8 35 d0
8e 13 bf 0a c7 43 bb 03 73 34 3c 0f 98 31 36 5a ea 1e 54 60 d6 f2 f1 5f 3e c7 c2 55 93 ef e7 52
48 1f fd 3d 64 87 c8 48 9f 5d 75 56 40 2f ce 9d 4e 95 ab 3b 3c d3 ef 8c f8 56 96 63 2d 25 f2 17
cc a8 5f 12 e5 97 0f e9 4e c6 1d 4f 6c 5c cd fd 61 47 f3 fe c4 53 e0 0e c0 92 8f ed b2 01 7d b7
f9 cd 4e df 50 a3 71 bd ba 31 e7 3e 87 5d d8 75 90 ab a5 39 39 b9 53 05 a2 c7 66 6e 03 22 bf 9c
4f aa 43 bc 72 6a 7a a9 8a 0b f7 7a 4b 9a f1 cd f2 86 fb b4 4e e5 b1 fb 56 c4 e0 3f 02 c1 74 bf
15 93 42 65 fc 66 18 3b 12 50 aa aa 68 e3 d1 a6 a8 e4 03 4f 60 eb 35 40 5c 51 eb 86 2a 8a 01 ea
ef 02 03 01 00 01

pubKeyBase64:
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA2VWUWGfebYsAgvOoqk0iYkZbhJCVBVFDWf+k
kc2kIULb+JYUF+VdRKKQNHxd0D4zlUN48HoZfZlqF9fq2DXQjhO/CsdDuwNzNDwPmDE2WuoeVGDW8vFf
PsfCVZPv51JIH/09ZIfISJ9ddVZAL86dTpWrOzzT74z4VpZjLSXyF8yoXxLllw/pTsYdT2xczf1hR/P+
xFPgDsCSj+2yAX23+c1O31Cjcb26Mec+h13YdZCrpTk5uVMFosdmbgMiv5xPqkO8cmp6qYoL93pLmvHN
8ob7tE7lsftWxOA/AsF0vxWTQmX8Zhg7ElCqqmjj0aao5ANPYOs1QFxR64YqigHq7wIDAQAB

====================================== RSA Private Key ======================================
privKeyBytes:
30 82 04 bf 02 01 00 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00 04 82 04 a9 30 82 04 a5 02 01
00 02 82 01 01 00 d9 55 94 58 67 de 6d 8b 00 82 f3 a8 aa 4d 22 62 46 5b 84 90 95 05 51 43 59 ff
a4 91 cd a4 21 42 db f8 96 14 17 e5 5d 44 a2 90 34 7c 5d d0 3e 33 95 43 78 f0 7a 19 7d 99 6a 17
d7 ea d8 35 d0 8e 13 bf 0a c7 43 bb 03 73 34 3c 0f 98 31 36 5a ea 1e 54 60 d6 f2 f1 5f 3e c7 c2
55 93 ef e7 52 48 1f fd 3d 64 87 c8 48 9f 5d 75 56 40 2f ce 9d 4e 95 ab 3b 3c d3 ef 8c f8 56 96
63 2d 25 f2 17 cc a8 5f 12 e5 97 0f e9 4e c6 1d 4f 6c 5c cd fd 61 47 f3 fe c4 53 e0 0e c0 92 8f
ed b2 01 7d b7 f9 cd 4e df 50 a3 71 bd ba 31 e7 3e 87 5d d8 75 90 ab a5 39 39 b9 53 05 a2 c7 66
6e 03 22 bf 9c 4f aa 43 bc 72 6a 7a a9 8a 0b f7 7a 4b 9a f1 cd f2 86 fb b4 4e e5 b1 fb 56 c4 e0
3f 02 c1 74 bf 15 93 42 65 fc 66 18 3b 12 50 aa aa 68 e3 d1 a6 a8 e4 03 4f 60 eb 35 40 5c 51 eb
86 2a 8a 01 ea ef 02 03 01 00 01 02 82 01 01 00 cd 26 8a 54 75 1f b2 19 9f 0e fd bf bd 99 f8 15
fa 42 13 3d 83 5d b4 9a b6 0e a9 a2 f4 11 b5 4f ee 62 96 10 3b 8a 47 e3 2c ec a0 8e b0 e3 8c 83
96 a7 4d 36 fa 9a ab 43 b0 b0 f7 20 f8 9c bb d4 11 71 a9 53 f5 6c 47 d2 6c 81 31 5b 41 41 04 ca
b0 7c 87 b8 ff 34 b6 ea 85 17 bd 3e 5b 1b e9 40 a9 e9 9b 15 15 53 0a b2 5c 3f 11 7e 9f 62 12 7d
86 ac 4e cb 99 67 2b 93 93 f0 7b 87 b7 a4 f1 ae 81 aa 31 eb fc 89 8c bb 72 c7 18 dc 4a 50 ef 2d
62 95 25 9b b1 0a 73 dd 9e 8f 95 4e d6 c5 c9 5f 2a a2 4c 48 a6 25 b8 08 70 f8 4f 59 b2 b1 26 80
29 08 ed 93 0d 01 2d 04 3e db 45 4c f8 67 80 2b 2d ff b8 c4 6b 4b ce 62 d8 13 e3 40 cc 0d 70 45
a2 72 4e 34 9d 8c 53 f6 be eb 64 a3 95 34 80 cc 50 b9 bd 89 fc cf 7c 57 f0 fc e7 52 4d 83 21 ca
71 4a 61 97 14 02 e5 46 52 96 6c c0 ff 76 8a 79 02 81 81 00 f5 77 ba 3f f0 6d d4 82 34 2c 33 b0
7a 69 26 68 4d 0c f2 74 21 ed f0 84 24 67 fb 08 8c 65 ab 0f 08 a3 59 76 2b 52 3f a0 01 dc 4b ab
b6 a5 f8 89 22 78 99 6d 53 72 9b 29 05 67 bc 83 4f 68 9d 47 e6 95 4c cf 57 29 90 0c af ef ab 4a
23 28 6a af e6 c8 71 4e e3 e9 51 a4 c9 93 67 be 20 b2 11 ba 6f de 11 be 1f 62 f5 ec a6 41 2c e3
0f 7e 28 0f 4e 26 76 cf 1b 91 74 33 33 37 12 f6 3d 4c 9a 43 02 81 81 00 e2 a8 d4 13 61 8b 47 5e
1c 9d 2e f8 7f e7 85 74 0c e9 ef 8d b9 13 cb 83 7c 33 3e e3 b8 fb 76 9d 53 8e 20 27 5f 0e 7a d4
0d c8 a9 7a d7 1e 30 99 a4 f4 1a 3c 17 dc df a1 39 5e f1 4a dd cb 88 7d 4c e0 ff 4e 3e 19 b8 3c
57 ba 58 3f e9 67 49 6a c3 5b c2 64 e4 80 99 3a a8 66 45 95 6c 20 6b 55 08 43 05 11 94 0c 73 db
```

```
6a c9 ca ed 2f 67 d6 28 40 b9 7d 69 73 fe f2 ab 6a 45 ee 2a 10 a6 0f e5 02 81 81 00 bf 59 65 c7
12 15 8b e3 33 e8 81 22 c1 49 d6 b6 d7 d9 8f e7 17 cb f4 02 0e 9f 40 01 99 f0 67 38 80 f3 55 79
d8 ab 75 0d b6 65 94 57 77 3a 4a 54 1c 9b 06 7f 42 dd c4 36 66 10 47 d3 d4 c3 28 58 34 57 8e 58
d3 09 83 51 60 94 e4 62 16 a6 1b 04 a4 52 fc 81 13 09 7e ab 86 b7 71 d7 b4 85 1b 6c ca 67 c2 4d
03 0a 91 ca b8 8f 75 fe 4a c5 7d 54 f1 06 ea f6 e7 f6 ab 2e 7e 6e 20 49 f3 df 13 21 02 81 80 02
15 e1 95 a8 11 1a ff ff ad 66 90 3c c5 09 92 4e c8 1c 3f 26 93 cb 0c 93 a8 f9 0c 29 58 8e f7 d5
9b fa 29 c2 93 24 88 2c f4 4a b2 e9 a5 ca 76 af 70 db 88 f0 03 45 3f 7b 82 a7 1f b4 38 ba 31 c4
f2 51 07 0c 45 3a 4f bb d2 e8 1a f6 6f cf da 1e a0 0a 82 d9 23 61 c5 8b 65 1d 80 c9 74 e6 e7 ea
62 8b 7a 64 ed 54 67 91 6c f7 e4 04 ca a4 ca cc 05 a8 e4 be 7f 7b 06 1e bd 33 fa c9 1a 6e 0d 02
81 81 00 83 2d bb 93 d1 6e b8 77 1a d1 cc ea 15 02 39 f0 e0 9d 99 14 a5 b2 d5 5b d5 90 b9 6b 64
60 ba fa 10 d4 f6 32 59 35 4a b6 13 66 5a 59 c8 ec e8 4b 2d bd 52 70 95 fd 34 3f 4f c1 b5 ce 03
f0 d4 1f c3 a8 65 d8 2d f5 f8 42 bb 75 be ae 48 f8 96 c3 44 25 f7 af 1a 0e 35 aa d4 44 15 71 15
61 4b 86 49 0c 4f 84 7a 60 fa 7f c6 71 36 9a f9 16 49 69 7f e6 67 5b 83 1e 49 27 99 f2 3d 18 7c
cd 84 2d

privKeyBase64:
MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKkwggSlAgEAAoIBAQDZVZRYZ95tiwCC86iqTSJiRluEkJUF
UUNZ/6SRzaQhQtv4lhQX5V1EopA0fF3QPjOVQ3jwehl9mWoX1+rYNdCOE78Kx0O7A3M0PA+YMTZa6h5U
YNby8V8+x8JVk+/nUkgf/T1kh8hIn111VkAvzp1Olas7PNPvjPhWlmMtJfIXzKhfEuWXD+lOxh1PbFzN
/WFH8/7EU+AOwJKP7bIBfbf5zU7fUKNxvbox5z6HXdh1kKulOTm5UwWix2ZuAyK/nE+qQ7xyanqpigv3
ekua8c3yhvu0TuWx+1bE4D8CwXS/FZNCZfxmGDsSUKqqaOPRpqjkA09g6zVAXFHrhiqKAervAgMBAAEC
ggEBAM0milR1H7IZnw79v72Z+BX6QhM9g120mrYOqaL0EbVP7mKWEDuKR+Ms7KCOsOOMg5anTTb6mqtD
sLD3IPicu9QRcalT9WxH0myBMVtBQQTKsHyHuP80tuqFF70+WxvpQKnpmxUVUwqyXD8Rfp9iEn2GrE7L
mWcrk5Pwe4e3pPGugaox6/yJjLtyxxjcSlDvLWKVJZuxCnPdno+VTtbFyV8qokxIpiW4CHD4T1mysSaA
KQjtkw0BLQQ+20VM+GeAKy3/uMRrS85i2BPjQMwNcEWick40nYxT9r7rZKOVNIDMULm9ifzPfFfw/OdS
TYMhynFKYZcUAuVGUpZswP92inkCgYEA9Xe6P/Bt1II0LDOwemkmaE0M8nQh7fCEJGf7CIxlqw8Io1l2
K1I/oAHcS6u2pfiJIniZbVNymykFZ7yDT2idR+aVTM9XKZAMr++rSiMoaq/myHFO4+lRpMmTZ74gshG6
b94Rvh9i9eymQSzjD34oD04mds8bkXQzMzcS9j1MmkMCgYEA4qjUE2GLR14cnS74f+eFdAzp7425E8uD
fDM+47j7dp1TjiAnXw561A3IqXrXHjCZpPQaPBfc36E5XvFK3cuIfUzg/04+Gbg8V7pYP+lnSWrDW8Jk
5ICZOqhmRZVsIGtVCEMFEZQMc9tqycrtL2fWKEC5fWlz/vKrakXuKhCmD+UCgYEAv1llxxIVi+Mz6IEi
wUnWttfZj+cXy/QCDp9AAZnwZziA81V52Kt1DbZllFd3OkpUHJsGf0LdxDZmEEfT1MMoWDRXjljTCYNR
YJTkYhamGwSkUvyBEwl+q4a3cde0hRtsymfCTQMKkcq4j3X+SsV9VPEG6vbn9qsufm4gSfPfEyECgYAC
FeGVqBEa//+tZpA8xQmSTsgcPyaTywyTqPkMKViO99Wb+inCkySILPRKsumlynavcNuI8ANFP3uCpx+0
OLoxxPJRBwxFOk+70uga9m/P2h6gCoLZI2HFi2UdgMl05ufqYot6ZO1UZ5Fs9+QEyqTKzAWo5L5/ewYe
vTP6yRpuDQKBgQCDLbuT0W64dxrRzOoVAjnw4J2ZFKWy1VvVkLlrZGC6+hDU9jJZNUq2E2ZaWcjs6Est
vVJwlf00P0/Btc4D8NQfw6hl2C31+EK7db6uSPiWw0Ql968aDjWq1EQVcRVhS4ZJDE+EemD6f8ZxNpr5
Fklpf+ZnW4MeSSeZ8j0YfM2ELQ==
```

RSA keypair generation using java Snipplet

Step	Description	java Snipplet	Data

p4.1	instantiate and initialize keypair-generator	java.security.KeyPairGenerator keyGen = java.security.KeyPairGenerator.*getInstance*("RSA");
		
		keyGen.initialize(3072);	

p4.2	generate keypair	java.security.KeyPair keypair = keyGen.genKeyPair();	
p4.3	get public key in byte[] form	**byte**[] pubKeyBytes = keypair.getPublic().getEncoded();	see codeblock above
p4.4	get public key in base64 form	String pubKeyBase64 = Base64.*getEncoder*().encodeToString(pubKeyBytes)	see codeblock above
p4.5	get private key in byte[] form	**byte**[] privKeyBytes = keypair.getPrivate().getEncoded();	see codeblock above
p4.6	get private key in base64 form	String privKeyBase64 = Base64.*getEncoder*().encodeToString(privKeyBytes)	see codeblock above

Step 1: Calculate HMAC of public key

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

Code Sample

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

OUT:

Step	Description	java Snipplet	Data
1.1	convert datapassword to byte[]	byte[] dataPasswordBytes = dataPassword.getBytes("UTF-8");	dataPasswordBytes = 78 52 37 65 61 32 5F 35 33 53 28 6D
1.2	convert RSA public key to byte[]	byte[] publicKeyBytes = Base64.getDecoder().decode (publicKeyBase64);	publicKeyBytes = 30 82 01 22 30 ... 02 03 01 00 01
1.3	create and initialize javax.crypto.Mac	SecretKeySpec signingKey = new SecretKeySpec (dataPasswordBytes, "HmacSHA256"); Mac mac = Mac.getInstance("HmacSHA256"); mac.init(signingKey);	
1.4	calculate HMAC hash bytes	mac.update(publicKeyBytes); byte[] hmacHashBytes = mac.doFinal();	hmacHashBytes = 52 00 AC 81 15 DE 1C 28 52 1C FC 52 50 9C FD 6C C5 25 89 83 A5 FF 2A 1F C3 10 C3 96 F2 D8 A2 39
1.5	convert hmac bytes to base64 form	String hmacHashBase64 = Base64.getEncoder().encodeToString(hashBytes);	hmacHashBase64 = UgCsgRXeHChSHPxSUJz9bMUliYOl /yofwxDDlvLYojk=

Step 2: Calculate the basic authorization header

Code Sample

IN: username (String - precondition #2) password (String - precondition #2)
OUT: HTTP authorization header string

Step	Description	java Snipplet	Data
2.1	concatenate username + ":" + password	String authString = username + ":" + password;	authString = SCRDEMO:3r#4Mu#GBRmP
2.2	convert the result of step 2.1 into base64 form	authString = Base64.getEncoder().encodeToString (authString.getBytes("UTF-8"));	authString = U0NSREVNTzozciM0TXUjR0JSbVA=
2.3	prepend "Basic " to authorization header value	authString = "Basic " + authString;	authString = Basic U0NSREVNTzozciM0TXUjR0JSbVA=
2.4	when generating HTTP-request Add authorization header to the request	httpUrlConnection.setRequestProperty("Authorization", authString);	Authorization: Basic U0NSREVNTzozciM0TXUjR0JSbVA=

Step 3: send http GET Request to Postident System

Retrieve result data via SCR API and provide the following headers:

- x-scr-key: Public key as Base64
- x-scr-keyhash: HMAC as Base64
- Authorization

Code Sample

IN: clientid (precondition #2)
IN: hostname (precondition #1)
IN: rsa public key in base64 form (precondition #4, step #4.4)
IN: hmac hash of rsa public key in base64 form (step #1.5)
IN: basic authorization string (step #2)

responseCode = huc.getResponseCode();

Step	Description	java Snipplet	Data

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`
3.2	instantiate anf configure URL connection	URL url = new URL (scrUrl); HttpURLConnection huc = (HttpURLConnection) url.openConnection(); huc.setRequestMethod ("GET"); huc.setConnectTimeout (30000); huc.setReadTimeout (30000); huc.setRequestProperty ("User-agent", "SCR-CLIENT"); huc.setRequestProperty ("Content-Type", "application/json"); huc.setDoOutput (false); huc.setDoInput(true);	
3.3	set Authorization Header	huc.setRequestProperty ("Authorization", authString);	`Authorization: Basic U0NSV1VMOjNyIzRNdSNHQlJtUA==`
3.4	set x-scr-key Header	huc.setRequestProperty ("x-scr-key", pubKeybase64);	x-scr-key : MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA2VWUWGfebYsAgvOoqk0iYkZbhJCVB kc2kIULb+JYUF+VdRKKQNHxd0D4zlUN48HoZfZlqF9fq2DXQjhO/CsdDuwNzNDwPmDE2WuoeV PsfCVZPv51JIH/09ZIfISJ9ddVZAL86dTpWrOzzT74z4VpZjLSXyF8yoXxLllw/pTsYdT2xczf xFPgDsCSj+2yAX23+c1O31Cjcb26Mec+h13YdZCrpTk5uVMFosdmbgMiv5xPqkO8cmp6qYoL93 8ob7tE7lsftWxOA/AsF0vxWTQmX8Zhg7ElCqqmjj0aao5ANPYOs1QFxR64YqigHq7wIDAQAB
3.5	set x-scr-keyhash Header	huc.setRequestProperty ("x-scr-keyhash", hmacHashBase64);	x-scr-keyhash : UgCsgRXeHChSHPxSUJz9bMUliYOl/yofwxDDlvLYojk=
3.6	fire the request	responseCode = huc.getResponseCode();	200

HTTP Request

```
HTTP Request
GET /api/scr/v1/865E6E37/cases/full/ HTTP/1.1
Host: postident.deutschepost.de
Content-Type: application/json
Authorization: Basic U0NSV1VMOjNyIzRNdSNHQlJtUA==
x-scr-key: MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA2VWUWGfebYsAgvOoq....IDAQAB
x-scr-keyhash: UgCsgRXeHChSHPxSUJz9bMUliYOl/yofwxDDlvLYojk=
```

curl Request

```
curl -X GET \
-H "Content-Type: application/json" \
-H "Authorization: Basic U0NSV1VMOjNyIzRNdSNHQlJtUA==" \
-H "x-scr-key:
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA2VWUWGfebYsAgvOoqk0iYkZbhJCVBVFDWf+kkc2kIULb+JYUF+VdRKKQN
```

Step 4: Postident system: validate encryption parameters and encrypt data

Postident system calculates HMAC of public key (from x-scr-key) and compare calculated HMAC to given HMAC (from x-scr-keyhash)

If the HMACs match, the Postident system encrypts data by using public key from x-scr-key to encrypt the result data.

Step 5: receive encrypted data

Code Sample

Step	Description	java Snipplet	Data
5.1	fire Request (same as 3.6)	int responseCode = huc.getResponseCode();	`200`
5.2	read response payload	String encryptedPayload = readInputStream(huc.getInputStream()	

5.3	read InputStream	StringBuilder sb = new StringBuilder();
		BufferedReader in = new BufferedReader(new InputStreamReader(istr, "UTF-8"));
		String row= "";
		while ((row= in.readLine()) != null) { sb.append(row); }
		return sb.toString();	

Sample Response (encrypted):


```
eyJlbmMiOiJBMjU2Q0JDLUhTNTEyIiwiYWxnIjoiUlNBMV81In0.QqnUfc93eFQwq5YBSrKW4AcVka8x
UU6OdaT24hGd7tOJ02pDEFLz-CMDRQKDtHpgtlfxAxVFIYZ0Vj-pzLhavwrsLKbVR-GTz0mytCLPrsAg
wh16BeLE5sG8GYCyuocsN-1XqjAEefP2iOeAuNp7jV_y4Ir0bHWKtr136xzgswAgRbVbHu1nxBRCtqRf
-VfyspGawrxDvCm2qygxiwPfCBvCmvoRn1E99en3EkwBHze2alttUbJz-u5F9wIJJHGA1WL7y8OaQlfk
KXHCh05LEn1eDhwzEi7PQ5Iqo2V7obkZ7Pr_kxMH-oTI3CTQqZv-SVI17hTpcZkHoL6pChlDBA.sWwy7
sFYP4ZEkigPkw98lw.Mi3utzmU5zG8oQIi0gD02BLNTeY8SU3X_XKQAFbrBc4H4mBjGT7cv2oMXbUMGi
B2UQq4KOEZyDQ5zD25Jryp8FB-KfRiHxrs8I-MQCakMYl2wN2zCMZSHwE9gMY-oUYgUzJaBgZ2Yemx1a
bCyyBpCPrqLatvF46vYX3FVeJPdVzCc3OOP9I_7SMICY8iJtP-V-vduG2FpM5QN3WmM3kYwNpkzGpWnh
TiFzJmAHk_WLwvNJSfazwmO3kK9MJ_s6NC9cf3jOFNH-Pjg3dpy3xHgz0kPHEWgtwMjWxsYOZ7VUGECB
vkHnO3sXUBCzPq7KWBg8jyWVNbcvWrdkAjoP5oXtTQOE8uzHcjr-aV5__RkqmLFeL_IDs0bAqAj1GbFS
PITHoJ2TThAfrb3LL7ty1TMckxHPkCWoxDH7ADIPZa3WyH7QQ8pYGWyp5dCM5qddZcK6i8UF_dygJB3-
UV56wFDAhraGoX-zE2BLHmWMN6HiW9o-FuZ2gX6TritsgZHtZ1AalA9lwDjicuON0YqgxoAiJHlL5Wtj
y8PvZjlECi8s5Vk8NwTIcYypFmGbnx83O1N3qCeSZYX8ExCG_aEDbDzCqUZXzMuUPN3ynYCSoSurPU8l
1-i2hzFIazqlbvYC0H4J7ZLn1uVtJsycDIpttCGuOvnR9TagJfsz8tJnLpkjfWli1eRPwdYnPqr-WB1v
3B5UPqutL-K0qAtRK_p_UBKonODB9G9Rr0cizCjMAWIOThbUKLy98sIkVmL_4dPiyKWrhZFBpBme0aum
-P0q93AiTyM7tF11otc3uz3NCPPmHkofg0SgCFMCeBunpfZKwGvlIwX-rdybKT7bORJt4WXtI7aMz6l0
W8_lycH0751-uMTxv-qMJLcV8L9OHBKFBZj9DgXTspmOL1RiQYW9HCTt2u_ZSe5MrI1gmZ0uIqntTW0p
vP2WDRNx-oKRCivzpa1XcFkJ2ejawmD7jPnOCeSeLeaWJjd_jyYEKzYDOi0Prtgfzg0PGqxuCbzzloaR
Ae87fDFdKcNWdKGt8tdUgU_5RNqBivpQXXB6FkI6aEqJmnbwnyiOCOMCyyb_y6wUNrQSuRFazmnbDa7P
rZzQlPEMXKy1w9R7n6w2Ql3erdiFRZO3ef5-wzvX7wvMjeYGoouFuvN79vAxA89HPkpGmpi1_kJCL3GW
3-bpp2GY6QnMzwwcZBXlYBNsTqHNASl5VDzCKrOulDeLRuB4IFXZjw0Os5TR8oi7RJWWThaXHOzlmaVR
6VAxLwvnaFaGoOeC9lXPei388irJs1_i6MeD4MNekKfC_vHynpnXi3aUR1jzXN71NwQMjbk59BLiNHW_
kTnn7mclf5Jo1vFPuBbHZ6gf-vDsVn-L12p7QwA0Br203rFOwQx8fC_OmczRHpKVRibB839OAQRovTXf
acWxNJr1t9rgdoyC5PXwiN0ba2VEFHCq7Dq4HdviOlVbmC215plMNeCZQPnOFbS6NNYTKHsQUfLMCnKw
nTnmOfWB3Fyfw.5lJNi2Ja9mS2tx4jXHFK0A-mWfWSKChDVW-c6Lj9Q_c
```


Step 6: decrypt data with private key

Use the private key to decrypt the encrypted result data and proceed with decrypted data.

Code Sample

IN: rsaPrivKeyBytes (precondition #4)

IN: encrypted Payload (step #5)

Step	Description	java Snipplet	Data
6.1	parse encrypted payload into JWEObject	JWEObject jweObject = JWEObject.parse(encryptedPayload);	

6.2	instantiate RSAPrivateKey	rsaPrivatteKey = (RSAPrivateKey) KeyFactory.getInstance("RSA").generatePrivate(new PKCS8EncodedKeySpec(keybytes));	
6.3	instantiate RSADecrypter	RSADecrypter rsaDecrypter= new RSADecrypter(rsaPrivateKey);	
6.4	decrypt JWEObject	jweObject.decrypt(rsaDecrypter);	
6.5	get decrypted responsestring	String decryptedResponse = jweObject.getPayload().toString();	

Sample Response (decrypted):


```
[
  {
    "caseId": "KGDJYAW2ZAK6",
    "caseStatus": {
      "status": "closed",
      "archived": false,
      "validUntil": "2017-01-04T13:33:13+01:00",
      "created": "2016-10-06T14:33:13+02:00",
      "modified": "2017-01-04T14:00:00+01:00"
    },
    "orderData": {
      "customData": {
        "custom1": "Kampagne 1234"
      },
      "processData": {
        "targetCountry": "DEU",
        "preferredLanguage": "DE_DE",
        "referenceId": "KGDJYAW2ZAK6",
        "callbackUrlCouponRequested": {

        },
        "callbackUrlReviewPending": {

        },
        "callbackUrlIncomplete": {

        },
        "callbackUrlSuccess": {

        },
        "callbackUrlDeclined": {

        }
      },
      "contactDataProvided": {
        "salutation": "1",
        "firstName": "Max",
        "lastName": "Muster",
        "mobilePhone": "0151123456789",
        "email": "max.muster@emailprovider.de",
        "address": {
          "streetAddress": "Musterstrasse 11a",
          "postalCode": "53113",
          "city": "Bonn",
          "country": "DEU"
        }
      },
      "identificationDocumentProvided": {

      },
      "drivingLicenceProvided": {
        "drivingLicenceClasses": [

        ]
      }
    },
    "primaryIdent": {
      "identificationMethod": "photo",
```

```
      "identificationStatus": {
        "status": "declined",
        "subStatus": {
          "code": 12,
          "description": "Operation valid time frame exceeded (case ID)"
        },
        "created": "2016-10-06T14:39:40+02:00",
        "modified": "2016-10-06T15:20:16+02:00"
      }
    },
    "accountingData": {
      "accountingNumber": "71234567163701"
    }
  }
]
```

4. Errors

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

Sample Error Messages

```
// 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."
    }
  ]
}


// 401: invalid credentials
{
  "apiversion": "v1",
  "errors": [
    {
      "errorcode": "90114",
      "reason": "unauthorized",
      "key": "Authorization",
      "message": "Authorization failed. Cause: 1"
    }
  ]
}

// 401: wrong clientid
{
  "apiversion": "v1",
  "errors": [
    {
      "errorcode": "90114",
      "reason": "unauthorized",
      "key": "Authorization",
      "message": "Authorization failed. Cause: 7"
    }
  ]
}

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

// 400: wrong keyhash (wrong datapassword for hash used or manipulated x-scr-keyhash or
manipulated x-scr-key)
{
  "apiversion": "v1",
  "errors": [
    {
      "errorcode": "90107",
      "reason": "hash failure",
      "key": "ScrEncryption",
      "message": "Provided Key doesnt match Keyhash."
    }
  ]
}
// 400: invalid RSA public Key
{
  "apiversion": "v1",
  "errors": [
    {
      "errorcode": "90109",
      "reason": "invalid key",
      "key": "ScrEncryption",
      "message": "Invalid RSAPublicKey format for EncryptionKey."
    }
  ]
}
```

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.

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

5. Code Samples

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.

Java SCR Client Sample

The code of the following SCR Client Sample "ScrClientTool" provides an command line tool to support service consumery in connection to scr service. The ScrClientTool can be used to

- generate RSA Keypairs
- calculate HMAC Hash on RSA Public Keys
- process scr requests
- decrypt SCR results
- and calculate BASIC Authorization Headers.

ScrClientTool java source

ScrClient 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`

The ScrClientTool project consists of following classes:

- ScrClientTool, the commandline interpreter
- ScrCaller, the HTTP request processor
- ScrCryptoHelper, which collects cryptographic functions to prepare scr requests and to decrypt scr responses.
- RsaKeyBytes, a RSA keypair Container

To use the ScrClientTool it has to be compiled into an executable jar file (See usage of ScrClientTool).

public class ScrClientTool

ScrClientTool

```
package de.deutschepost.postident.scrClient;
import java.io.File;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.text.MessageFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.CommandLineParser;
import org.apache.commons.cli.GnuParser;
import org.apache.commons.cli.HelpFormatter;
import org.apache.commons.cli.Option;
import org.apache.commons.cli.OptionBuilder;
import org.apache.commons.cli.Options;
import org.apache.commons.cli.ParseException;
import org.apache.commons.configuration.ConfigurationException;
import org.apache.commons.io.FileUtils;
import com.nimbusds.jose.JOSEException;
/**
 * ScrClientTool - commandline tool to support scr client integration.
 *
 * @author Deutsche Post AG
 * @version 1.0
 */
public class ScrClientTool {
   /** filename of scr call logfile */
   private static final String SCR_CALL_LOG_FILENAME = "SCR_CALL.log";
   // command line parametrization constants
   private static final String ARG_USERNAME = "username";
   private static final String DESC_USERNAME = "username for SCR Service (provided by the Deutsche
Post technical sales or service team)";
   private static final String ARG_PASSWORD = "password";
   private static final String DESC_PASSWORD = "password for SCR Service (provided by the Deutsche
Post technical sales or service team)";
   private static final String ARG_CLIENTID = "clientId";
   private static final String DESC_CLIENT_ID = "SCR ClientID (provided by the Deutsche Post
technical sales or service team)";
```

```
   private static final String ARG_DATAPASSWORD = "datapassword";
   private static final String DESC_DATAPASSWORD = "datapassword to calculate hmac-hash (provided
by the Deutsche Post technical sales or service team)";
   private static final String ARG_VERZEICHIS = "dir";
   private static final String DESC_VERZEICHNIS = "log directory; location for logging input and
output values from called operation";
   private static final String ARG_SCRHOST = "scrhost";
   private static final String DESC_SCRHOST = "scr-hostname ( production: postident.deutschepost.
de,  test and integration: postident-demo.deutschepost.de)";
   private static final String ARG_PUBKEY = "pubkey";
   private static final String DESC_PUBKEY = "RSA public key(Base64)\r\n public key will be
transmitted to scr service and becomes used to engrypt the service response ";
   private static final String ARG_PRIVKEY = "privkey";
   private static final String DESC_PRIVKEY = "RSA private key (Base64)\r\n private key will be
used to decrypt scr response (not transmitted to scr service)";
   private static final String ARG_CIPHER = "cipher";
   private static final String DESC_CIPHER = "RSA encrypted SCR response";
   private static final String ARG_KEYHASH = "keyhash";
   private static final String DESC_KEYHASH = "HMAC-hash of pubkey (Base64)";
   private static final String ARG_HELP = "help";
   private static final char OPT_HELP = '?';
   private static final String DESC_HELP = "show help";
   private static final String ARG_GENKEYPAIR = "genkey";
   private static final char OPT_GENKEYPAIR = 'k';
   private static final String DESC_GENKEYPAIR = "generate RSA 3072 keypair";
   private static final String ARG_GENKEYHASH = "hash";
   private static final char OPT_GENKEYHASH = 'h';
   private static final String DESC_GENKEYHASH = "generate HMAC-hash over <pubkey> using
<datapassword> as secret";
   private static final String ARG_GENAUTHORISATION = "auth";
   private static final char OPT_GENAUTHORISATION = 'a';
   private static final String DESC_GENAUTHORISATION = "generates an HTTP Basic athorization
String (requires <username> and <password>)";
   private static final String ARG_DECRYPT = "decrypt";
   private static final char OPT_DECRYPT = 'd';
   private static final String DESC_DECRYPT = "decrypts <cipher> by using <privkey> ";
   private static final String ARG_CALLSCR = "callscr";
   private static final String DESC_CALLSCR = "Calls scr service and decrypts the scr response. "
         + "\r\n <username>, <password> and <scrhost> are always required "
         + "\r\n  mode 1: <datenpassword> provided - generate a new keypair and handle encrypted
scr request"
         + "\r\n  mode 2: <pubkey>, <privkey> and <keyhash> provided - handle encrypted scr
request with given keys"
         + "\r\n  mode 3: <pubkey>, <privkey> and <datapassword> provided - generate hash and
handle encrypted scr request with given keys"
         + "\r\n  mode 4: nothing else provided - handle unencrypted scr request(only possible in
test environment)";
   private static final char OPT_CALLSCR = 'c';
   /** String scrUrlTemplate to be completed with scrHostname(0) and clientid(1). */
   private static final String SCR_URL_TEMPLATE = "https://{0}/api/scr/v1/{1}/cases";
   public ScrClientTool() {
   }
   /**
    * program start from comamndline
    *
    * @param args
    */
   @SuppressWarnings("static-access")
   public static void main(String[] args) {
      Options options = new Options();
      CommandLine commandLine = null;
      // logging Buffer for scr call input and output values
      StringBuilder sbOutput = new StringBuilder();
      try {
         CommandLineParser parser = new GnuParser();
         configureCommandLineArgs(options);
         // parse commandline
         commandLine = parser.parse(options, args);
         String cmdLineParamUsername = commandLine.getOptionValue(ARG_USERNAME);
```

```
         String cmdLineParamPassword = commandLine.getOptionValue(ARG_PASSWORD);
         String cmdLineParamDatapassword = commandLine.getOptionValue(ARG_DATAPASSWORD);
         String cmdLineParamClientId = commandLine.getOptionValue(ARG_CLIENTID);
         String cmdLineParamLogDir = commandLine.getOptionValue(ARG_VERZEICHIS);
         String cmdLineParamScrHost = commandLine.getOptionValue(ARG_SCRHOST);
         String cmdLineParamRsaPubkey = commandLine.getOptionValue(ARG_PUBKEY);
         String cmdLineParamRsaPrivkey = commandLine.getOptionValue(ARG_PRIVKEY);
         String cmdLineParamKeyhash = commandLine.getOptionValue(ARG_KEYHASH);
         String cmdLineParamCipher = commandLine.getOptionValue(ARG_CIPHER);
         // append input values to log
         log(sbOutput, ARG_USERNAME, cmdLineParamUsername);
         log(sbOutput, ARG_PASSWORD, cmdLineParamPassword);
         log(sbOutput, ARG_DATAPASSWORD, cmdLineParamDatapassword);
         log(sbOutput, ARG_CLIENTID, cmdLineParamClientId);
         log(sbOutput, ARG_VERZEICHIS, cmdLineParamLogDir);
         log(sbOutput, ARG_SCRHOST, cmdLineParamScrHost);
         log(sbOutput, ARG_PUBKEY, cmdLineParamRsaPubkey);
         log(sbOutput, ARG_PRIVKEY, cmdLineParamRsaPrivkey);
         log(sbOutput, ARG_KEYHASH, cmdLineParamKeyhash);
         log(sbOutput, ARG_CIPHER, cmdLineParamCipher);

         if (commandLine.hasOption(ARG_HELP)) {
            // help command called
            showHelp(options);
            System.exit(0);
         } else if (commandLine.hasOption(ARG_GENKEYPAIR)) {
            // generate keypair command called
            genKeyPair(cmdLineParamLogDir, sbOutput);
            System.exit(0);
         } else if (commandLine.hasOption(ARG_GENKEYHASH)) {
            // generate keyhash command called
            genKeyHash(cmdLineParamDatapassword, cmdLineParamRsaPubkey, cmdLineParamLogDir,
sbOutput);
            System.exit(0);
         } else if (commandLine.hasOption(ARG_CALLSCR)) {
            // call scr command called
            callScr(cmdLineParamScrHost, cmdLineParamUsername, cmdLineParamPassword,
cmdLineParamClientId,
                  cmdLineParamDatapassword, cmdLineParamRsaPubkey, cmdLineParamRsaPrivkey,
cmdLineParamKeyhash,
                  cmdLineParamLogDir, sbOutput, options);
            System.exit(0);
         } else if (commandLine.hasOption(ARG_GENAUTHORISATION)) {
            // generate basic authorization command calleed
            genBasicAuth(cmdLineParamUsername, cmdLineParamPassword, sbOutput, cmdLineParamLogDir);
            System.exit(0);
         } else if (commandLine.hasOption(ARG_DECRYPT)) {
            // decrypt scr response
            decryptCipher(cmdLineParamCipher, cmdLineParamRsaPrivkey, cmdLineParamLogDir,
sbOutput);
            System.exit(0);
         } else {
            // default: show help
            showHelp(options);
            System.exit(0);
         }
      } catch (ParseException exception) {
         out("invalid parameter: {}", exception.getMessage());
         showHelp(options);
         System.exit(0);
      } catch (Throwable e) {// NOSONAR 3rdparty calls
         out("unexpected Error: {0}", e.getMessage());
      }
   }
   /**
    * Calculates Basic Athorization String.
    * @param cmdLineParamUsername
    * @param cmdLineParamPassword
    * @param sbOutput
```

```
    * @param cmdLineParamLogVerzeichnis
    *
    * @throws ConfigurationException
    * @throws IOException
    */
   private static void genBasicAuth(String cmdLineParamUsername, String cmdLineParamPassword,
StringBuilder sbOutput,
         String cmdLineParamLogVerzeichnis) throws ConfigurationException, IOException {
      out("Calculate basic authorization header user {0} password {1}", cmdLineParamUsername,
cmdLineParamPassword);
      String auth = ScrCryptoHelper.calcBasicAuthString(cmdLineParamUsername,
cmdLineParamPassword);
      out(auth);
      log(sbOutput, "berechneter Authorization String: ", auth);
      saveCallLog(sbOutput, cmdLineParamLogVerzeichnis);
   }

   /**
    * generates x-scr-keyhash
    *
    * @param commandLine
    */
   private static void genKeyHash(String cmdLineParamDatapassword, String cmdLineParamRsaPubkey,
         String cmdLineParamLogVerzeichnis, StringBuilder sbOutput) {
      try {
         out("generate hmac hash");
         if (cmdLineParamDatapassword == null || cmdLineParamDatapassword.length() == 0) {
            out("Error: " + ARG_DATAPASSWORD + " missing.");
            return;
         }
         if (cmdLineParamRsaPubkey == null || cmdLineParamRsaPubkey.length() == 0) {
            out("Error: " + ARG_PUBKEY + " missing.");
            return;
         }
         out("SCR flow precondition 3: SCR datapassword: {0}", cmdLineParamDatapassword);
         out("RSA flow precondition 4.4: public key base64: " + cmdLineParamRsaPubkey);
         String hash = ScrCryptoHelper.hmacHashOverKey(cmdLineParamDatapassword,
cmdLineParamRsaPubkey);
         out("HMAC-HASH Base64: " + hash);
         log(sbOutput, "berechneter HmacHash", hash);
         saveCallLog(sbOutput, cmdLineParamLogVerzeichnis);
      } catch (NoSuchAlgorithmException e) {
         out("Fehler bei der Hashberechnung {0}", e);
      } catch (Throwable e) {// NOSONAR 3rdparty calls
         out("Unvorhergesehener Fehler bei der Hashberechnung {0}", e);
      }
   }
   /**
    * generates RSA keypair
    * SCR flow precondition 4 see {@link RsaKeyBytes#build(int)} for details.
    *
    * @param cmdLineParamLogVerzeichnis
    * @param sbOutput
    */
   private static void genKeyPair(String cmdLineParamLogVerzeichnis, StringBuilder sbOutput) {
      try {
         RsaKeyBytes rsaKeyBytes = RsaKeyBytes.build(3072);
         out("SCR flow precondition 4: " + rsaKeyBytes.toString());
         log(sbOutput, "generated public key", rsaKeyBytes.getPublicKeyStr());
         log(sbOutput, "generated private key", rsaKeyBytes.getPrivateKeyStr());
         saveCallLog(sbOutput, cmdLineParamLogVerzeichnis);
      } catch (NoSuchAlgorithmException e) {
         out("ERROR during key generation {0}", e);
      } catch (Throwable e) { // NOSONAR 3rdparty calls
         out("Unexpected ERROR during key generation {0}", e);
      }
   }
   /** handels an full scr request. dispatches the corresponding methodout of
    *   {@link #callScrWithKeyGeneration(String, String, String, String, String, String,
```

```
StringBuilder)},
    *   {@link #callScrWithoutEncryption(String, String, String, String, String, StringBuilder)}
    *   {@link #callScrWithProvidedKey(String, String, String, String, String, String, String,
String, String, StringBuilder)}
    *
    * @param cmdLineParamScrHost
    * @param cmdLineParamUsername
    * @param cmdLineParamPassword
    * @param cmdLineParamClientId
    * @param cmdLineParamDatapassword
    * @param cmdLineParamRsaPubkey
    * @param cmdLineParamRsaPrivkey
    * @param cmdLineParamKeyhash
    * @param cmdLineParamLogVerzeichnis
    * @param sbOutput
    *           output buffer 4 logging
    * @param options
    *           commandline parser options to show help page
    * @throws NoSuchAlgorithmException
    * @throws UnsupportedEncodingException
    * @throws InvalidKeyException
    * @throws java.text.ParseException
    * @throws JOSEException
    * @throws ScrException
    * @throws IOException
    * @throws ConfigurationException
    */
   private static void callScr(String cmdLineParamScrHost, String cmdLineParamUsername, String
cmdLineParamPassword,
         String cmdLineParamClientId, String cmdLineParamDatapassword, String
cmdLineParamRsaPubkey,
         String cmdLineParamRsaPrivkey, String cmdLineParamKeyhash, String
cmdLineParamLogVerzeichnis,
         StringBuilder sbOutput, Options options)
         throws NoSuchAlgorithmException, UnsupportedEncodingException, InvalidKeyException, java.
text.ParseException,
         JOSEException, ScrException, IOException, ConfigurationException {
      out("Info: calling scr Service");
      out("SCR flow precondition 1: postident hostname: {0}", cmdLineParamScrHost);
      out("SCR flow precondition 2: SCR username, password, clientId ({0}, {1}, {2})",
cmdLineParamUsername, cmdLineParamPassword, cmdLineParamClientId);
      out("SCR flow precondition 3: SCR datapassword: {0}", cmdLineParamDatapassword);
      if ((cmdLineParamDatapassword == null || cmdLineParamDatapassword.length() == 0)
            && ((cmdLineParamKeyhash == null || "".equals(cmdLineParamKeyhash)))) {
         // no datapassword and no keyhash ->
         // fallback-case: handle call w/o encryption
         callScrWithoutEncryption(cmdLineParamScrHost, cmdLineParamUsername, cmdLineParamPassword,
cmdLineParamClientId,
               cmdLineParamLogVerzeichnis, sbOutput);
      } else if (cmdLineParamRsaPubkey != null) {
         // pubkey provided -> handle call with provided key
         out("Info: provided rsa public key will be used.");
         callScrWithProvidedKey(cmdLineParamScrHost, cmdLineParamUsername, cmdLineParamPassword,
cmdLineParamClientId,
               cmdLineParamDatapassword, cmdLineParamRsaPubkey, cmdLineParamRsaPrivkey,
cmdLineParamKeyhash,
               cmdLineParamLogVerzeichnis, sbOutput);
      } else if (cmdLineParamDatapassword != null && cmdLineParamDatapassword.length() > 0) {
         // pubkey not provided -> generate keypair and handle call
         callScrWithKeyGeneration(cmdLineParamScrHost, cmdLineParamUsername, cmdLineParamPassword,
cmdLineParamClientId,
               cmdLineParamDatapassword, cmdLineParamLogVerzeichnis, sbOutput);
      } else {
         out("Error: required parameter missing.");
         showHelp(options);
      }
   }
   /**
    * generates RSA keypair and then
```

```
    * calls {@link #callScrWithProvidedKey(String, String, String, String, String, String, String,
String, String, StringBuilder)}
    *
    * @param cmdLineParamScrHost
    * @param cmdLineParamUsername
    * @param cmdLineParamPassword
    * @param cmdLineParamClientId
    * @param cmdLineParamDatenpassword
    * @param cmdLineParamLogVerzeichnis
    * @param sbOutput
    * @throws NoSuchAlgorithmException
    * @throws UnsupportedEncodingException
    * @throws InvalidKeyException
    * @throws java.text.ParseException
    * @throws JOSEException
    * @throws ScrException
    * @throws IOException
    * @throws ConfigurationException
    */
   private static void callScrWithKeyGeneration(String cmdLineParamScrHost, String
cmdLineParamUsername,
         String cmdLineParamPassword, String cmdLineParamClientId, String
cmdLineParamDatenpassword,
         String cmdLineParamLogVerzeichnis, StringBuilder sbOutput)
         throws NoSuchAlgorithmException, UnsupportedEncodingException, InvalidKeyException, java.
text.ParseException,
         JOSEException, ScrException, IOException, ConfigurationException {
      out("datapassword provided -> generate keypair");
      RsaKeyBytes rsakeybytes = RsaKeyBytes.build(3072);
      log(sbOutput, "generated RSA PublicKey", rsakeybytes.getPublicKeyStr());
      log(sbOutput, "generated RSA PrivateKey", rsakeybytes.getPrivateKeyStr());
      out("SCR flow precondition 4: generated RSA keypair: {0}", rsakeybytes);
      callScrWithProvidedKey(cmdLineParamScrHost, cmdLineParamUsername, cmdLineParamPassword,
cmdLineParamClientId,
            cmdLineParamDatenpassword, rsakeybytes.getPublicKeyStr(), rsakeybytes.
getPrivateKeyStr(), null,
            cmdLineParamLogVerzeichnis, sbOutput);
   }
   /**
    * calls SCR with provided Key
    *
    * @param cmdLineParamScrHost
    * @param cmdLineParamUsername
    * @param cmdLineParamPassword
    * @param cmdLineParamClientId
    * @param cmdLineParamDatenpassword
    * @param cmdLineParamRsaPubkey
    * @param cmdLineParamRsaPrivkey
    * @param cmdLineParamKeyhash
    * @param cmdLineParamLogVerzeichnis
    * @param sbOutput
    * @throws UnsupportedEncodingException
    * @throws NoSuchAlgorithmException
    * @throws InvalidKeyException
    * @throws java.text.ParseException
    * @throws JOSEException
    * @throws ScrException
    * @throws IOException
    * @throws ConfigurationException
    */
   private static void callScrWithProvidedKey(String cmdLineParamScrHost, String
cmdLineParamUsername,
         String cmdLineParamPassword, String cmdLineParamClientId, String
cmdLineParamDatenpassword,
         String cmdLineParamRsaPubkey, String cmdLineParamRsaPrivkey, String cmdLineParamKeyhash,
         String cmdLineParamLogVerzeichnis, StringBuilder sbOutput)
         throws UnsupportedEncodingException, NoSuchAlgorithmException, InvalidKeyException, java.
text.ParseException,
         JOSEException, ScrException, IOException, ConfigurationException {
```

```
      String keyhash = "";
      if (cmdLineParamKeyhash == null || "".equals(cmdLineParamKeyhash)) {
         // SCR flow step 1. calculate keyhash (if not provided)
         // the case that neither keyhash nor datapassword had been provided has been handeld in
calling Method
         keyhash = ScrCryptoHelper.hmacHashOverKey(cmdLineParamDatenpassword,
cmdLineParamRsaPubkey);
         log(sbOutput, "SCR flow generated x-scr-keyhash", keyhash);
      } else {
         keyhash = cmdLineParamKeyhash;
         out( "SCR flow 1. provided x-scr-keyhash {0}", keyhash);
      }
      // SCR flow step 2: calculate basic authorization header
      String authString = ScrCryptoHelper.calcBasicAuthString(cmdLineParamUsername,
cmdLineParamPassword);
      // SCR flow step 3.1 build scr url
      // fill host and clientid into urltemplate
      String callUrl = MessageFormat.format(SCR_URL_TEMPLATE, cmdLineParamScrHost,
cmdLineParamClientId);
      out("SCR flow 3.1: build scr url: " + callUrl);
      log(sbOutput, "called ScrURL", callUrl);
      // SCR flow steps 3 (fire request) and 5 (read response):
      // use ScrCaller to handle Request
      ScrCaller scrCaller = new ScrCaller();
      String scrResponse = scrCaller.callScr(callUrl, authString, cmdLineParamRsaPubkey, keyhash);
      log(sbOutput, "scr response", scrResponse); // log scr response
      // 3. Entschlüssele SCR antwort sofern möglich
      if (!(scrResponse.startsWith("Fehler")) || (scrResponse.startsWith("Error")) && !scrResponse.
startsWith("{")) {
         if (cmdLineParamRsaPrivkey != null) {
            // SCR flow step 6: decrypt data with private key
            out("decrypt with rsa private key");
            // decrypt SCR response
            String decryptedPayload = ScrCryptoHelper.decryptPayload(scrResponse,
cmdLineParamRsaPrivkey);
            out("SCR flow 6.5: response (decrypted):\r\n" + decryptedPayload);
            log(sbOutput, "scr response (decrypted): ", decryptedPayload);
         } else {
            System.out.println("rsa private Key not provided --> scr response will not be
decrypted");
         }
      }
      saveCallLog(sbOutput, cmdLineParamLogVerzeichnis);// log scr call
   }
   /**
    * calls scr without encryption
    *
    * @param cmdLineParamScrHost
    * @param cmdLineParamUsername
    * @param cmdLineParamPassword
    * @param cmdLineParamClientId
    * @param cmdLineParamLogVerzeichnis
    * @param sbOutput
    * @throws IOException
    * @throws ConfigurationException
    */
   private static void callScrWithoutEncryption(String cmdLineParamScrHost, String
cmdLineParamUsername,
         String cmdLineParamPassword, String cmdLineParamClientId, String
cmdLineParamLogVerzeichnis,
         StringBuilder sbOutput) throws IOException, ConfigurationException {
      out("Warning: neither datapassword nor keyhash provided -> handle unencrypted SCR request.");
      // SCR flow step 2: calculate basic authorization header
      String authString = ScrCryptoHelper.calcBasicAuthString(cmdLineParamUsername,
cmdLineParamPassword);
      // SCR flow step 3.1 build scr url
      // fill host and clientid into urltemplate
      String callUrl = MessageFormat.format(SCR_URL_TEMPLATE, cmdLineParamScrHost,
cmdLineParamClientId);
```

```
      out("SCR flow 3.1: build scr url: " + callUrl);
      log(sbOutput, "called ScrURL", callUrl);
      // SCR flow steps 3 (fire request) and 5 (read response):
      // use ScrCaller to handle Request
      ScrCaller scrCaller = new ScrCaller();
      String scrResponse = scrCaller.callScr(callUrl, authString);
      log(sbOutput, "scr response", scrResponse); // log scr response
      saveCallLog(sbOutput, cmdLineParamLogVerzeichnis);
   }
   /**
    * decrypt an RSA encrypted message
    *
    * @param cmdLineParamCipher
    * @param cmdLineParamRsaPrivkey
    * @param cmdLineParamLogVerzeichnis
    * @param sbOutput
    * @throws java.text.ParseException
    * @throws JOSEException
    * @throws ScrException
    * @throws ConfigurationException
    * @throws IOException
    */
   private static void decryptCipher(String cmdLineParamCipher, String cmdLineParamRsaPrivkey,
         String cmdLineParamLogVerzeichnis, StringBuilder sbOutput)
         throws java.text.ParseException, JOSEException, ScrException, ConfigurationException,
IOException {
      out("Decrypt Cipher");
      out("SCR flow precondition 4.6: rsa private key in base64 form: " + cmdLineParamRsaPrivkey);
      out("Cipher: " + cmdLineParamCipher);
      if (cmdLineParamCipher == null || cmdLineParamCipher.length() == 0) {
         out("Error: " + ARG_CIPHER + " missing.");
         System.exit(0);// NOSONAR commandline app
      }
      if (cmdLineParamRsaPrivkey == null || cmdLineParamRsaPrivkey.length() == 0) {
         out("Error: " + ARG_PRIVKEY + " missing.");
         System.exit(0); // NOSONAR commandline app
      }
      // decrypt Cipher
      String decryptedPayload = ScrCryptoHelper.decryptPayload(cmdLineParamCipher,
cmdLineParamRsaPrivkey);
      out("SCR flow 6.5: cipher (decrypted):\r\n" + decryptedPayload);
      log(sbOutput, "cipher (decrypted)", decryptedPayload);
      saveCallLog(sbOutput, cmdLineParamLogVerzeichnis);
   }
   /**
    * show Program Help
    *
    * @param options
    */
   private static void showHelp(Options options) {
      HelpFormatter formatter = new HelpFormatter();
      formatter.printHelp("ScrClientTool", options);
   }
   /**
    * Configure commandline parsing options
    *
    * @param options
    */
   @SuppressWarnings("static-access")
   private static void configureCommandLineArgs(Options options) {
      Option optUsername = OptionBuilder.withLongOpt(ARG_USERNAME).withArgName(ARG_USERNAME).
hasArg()
            .withDescription(DESC_USERNAME).create();
      Option optPassword = OptionBuilder.withLongOpt(ARG_PASSWORD).withArgName(ARG_PASSWORD).
hasArg()
            .withDescription(DESC_PASSWORD).create();
      Option optClientID = OptionBuilder.withLongOpt(ARG_CLIENTID).withArgName(ARG_CLIENTID).
hasArg()
            .withDescription(DESC_CLIENT_ID).create();
```

```
      Option optDataPasswd = OptionBuilder.withLongOpt(ARG_DATAPASSWORD).withArgName
(ARG_DATAPASSWORD).hasArg()
            .withDescription(DESC_DATAPASSWORD).create();
      Option optWorkDir = OptionBuilder.withLongOpt(ARG_VERZEICHIS).withArgName(ARG_VERZEICHIS).
hasArg()
            .withDescription(DESC_VERZEICHNIS).create();
      Option optScrHost = OptionBuilder.withLongOpt(ARG_SCRHOST).withArgName(ARG_SCRHOST).hasArg()
            .withDescription(DESC_SCRHOST).create();
      Option optPubKey = OptionBuilder.withLongOpt(ARG_PUBKEY).withArgName(ARG_PUBKEY).hasArg()
            .withDescription(DESC_PUBKEY).create();
      Option optPrivKey = OptionBuilder.withLongOpt(ARG_PRIVKEY).withArgName(ARG_PRIVKEY).hasArg()
            .withDescription(DESC_PRIVKEY).create();
      Option optCipher = OptionBuilder.withLongOpt(ARG_CIPHER).withArgName(ARG_CIPHER).hasArg()
            .withDescription(DESC_CIPHER).create();
      Option optKeyHash = OptionBuilder.withLongOpt(ARG_KEYHASH).withArgName(ARG_KEYHASH).hasArg()
            .withDescription(DESC_KEYHASH).create();
      Option optHelp = OptionBuilder.withLongOpt(ARG_HELP).withArgName(ARG_HELP).withDescription
(DESC_HELP)
            .create(OPT_HELP);
      Option genKeyPair = OptionBuilder.withLongOpt(ARG_GENKEYPAIR).withArgName(ARG_GENKEYPAIR)
            .withDescription(DESC_GENKEYPAIR).create(OPT_GENKEYPAIR);
      Option genKeyHash = OptionBuilder.withLongOpt(ARG_GENKEYHASH).withArgName(ARG_GENKEYHASH)
            .withDescription(DESC_GENKEYHASH).create(OPT_GENKEYHASH);
      Option genAuth = OptionBuilder.withLongOpt(ARG_GENAUTHORISATION).withArgName
(ARG_GENAUTHORISATION)
            .withDescription(DESC_GENAUTHORISATION).create(OPT_GENAUTHORISATION);
      Option callScr = OptionBuilder.withLongOpt(ARG_CALLSCR).withArgName(ARG_CALLSCR).
withDescription(DESC_CALLSCR)
            .create(OPT_CALLSCR);
      Option callDecrypt = OptionBuilder.withLongOpt(ARG_DECRYPT).withArgName(ARG_DECRYPT).
withDescription(DESC_DECRYPT)
            .create(OPT_DECRYPT);
      options.addOption(optHelp);
      options.addOption(genAuth);
      options.addOption(callScr);
      options.addOption(callDecrypt);
      options.addOption(genKeyPair);
      options.addOption(genKeyHash);
      options.addOption(optUsername);
      options.addOption(optPassword);
      options.addOption(optClientID);
      options.addOption(optDataPasswd);
      options.addOption(optWorkDir);
      options.addOption(optScrHost);
      options.addOption(optPubKey);
      options.addOption(optPrivKey);
      options.addOption(optCipher);
      options.addOption(optKeyHash);
   }

   /**
    * log the SCR call
    *
    * @param callLogSB
    *           StringBuilder containing log messages
    * @param path
    *           path to whrite the Log
    * @throws IOException
    * @throws ConfigurationException
    */
   private static void saveCallLog(StringBuilder callLogSB, String path) throws IOException,
ConfigurationException {
      File file = new File((path != null) ? (path + File.separator + getTimestamp() + "_" +
SCR_CALL_LOG_FILENAME)
            : (getTimestamp() + "_" + SCR_CALL_LOG_FILENAME));
      if (!file.exists()) {
         file.createNewFile();
      }
      FileUtils.writeByteArrayToFile(file, callLogSB.toString().getBytes("utf-8"));
```

```
   }
   /**
    * get current timestam as string
    *
    * @return
    */
   private static String getTimestamp() {
      SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd_HH-mm-ss");
      return df.format(new Date());
   }
   /**
    * console out with parameter substitution
    *
    * @param message
    * @param args
    */
   public static void out(String message, Object... args) {
      if (args.length == 0) {
         System.out.println(message);
      } else {
         System.out.println(MessageFormat.format(message, args));
      }
   }
   /**
    * write to log Buffer
    *
    * @param message
    * @param args
    */
   public static void log(StringBuilder out, String key, String val) {
      out.append(key).append(" : ").append(val).append("\r\n");
   }
}
```

public class ScrCaller

ScrCaller

```
package de.deutschepost.postident.scrClient;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.ConnectException;
import java.net.HttpURLConnection;
import java.net.SocketTimeoutException;
import java.net.URL;
import java.net.UnknownHostException;
import java.text.MessageFormat;
/**
 * SCR Request Handler
 *
 * methods to handle SCR HTTP requests
 *
 * @author Deutsche Post AG
 * @version 1.0
 */
public class ScrCaller {
   /**
    * calls scr service and returns the service response as string
    *
    * @param scrUrl
    *           url to SCR Service e.g. https://postident.deutschepost.de/api/scr/v1/865E6E37/cases
    * @param authString for BASIC authentication
    * @param rsaPubkey
    *           rsa public key in base64 form
```

```
    * @param rsaKeyhash
    *           hmac hash over rsa pubkey in base64 form
    * @return the SCR Response
    */
   public String callScr(String scrUrl, String authString,  String rsaPubkey, String rsaKeyhash) {
      int responseCode = -1;
      StringBuilder sbRet = new StringBuilder();
      if (scrUrl == null || scrUrl.isEmpty()) {
         out("Error: URL not scecified.");
         sbRet.append("Error: URL not scecified.");
         return sbRet.toString();
      }
      try {
         // SCR flow step 3.2 instantiate anf configure URL connection
         URL url = new URL(scrUrl);
         HttpURLConnection huc = (HttpURLConnection) url.openConnection();
         huc.setRequestMethod("GET");
         huc.setConnectTimeout(30000);
         huc.setReadTimeout(30000);
         huc.setRequestProperty("User-agent", "SCR-CLIENT");
         huc.setRequestProperty("Content-Type", "application/json");
         out("Info: " + "HEADER User-agent: SCR-CLIENT ");
         // SCR flow step 3.3 set Authorization Header
         huc.setRequestProperty("Authorization", authString);
         out("SCR flow 3.3: HEADER Authorization: " + authString);
         huc.setDoOutput(false);
         huc.setDoInput(true);
         if (rsaPubkey != null) {
            huc.setRequestProperty("x-scr-key", rsaPubkey);
            out("SCR flow 3.4: HEADER x-scr-key: " + rsaPubkey);
         }
         if (rsaKeyhash != null) {
            huc.setRequestProperty("x-scr-keyhash", rsaKeyhash);
            out("SCR flow 3.5:  x-scr-keyhash: " + rsaKeyhash);
         }
         out("Info: " + "HEADER Content-Type: application/json");
         // SCR flow step #3 send the http GET Request to Postident System
         responseCode = huc.getResponseCode();
         String encryptedPayload = "";
         if (responseCode == 200) {
            encryptedPayload = readInputStream(huc.getInputStream(), true);
            sbRet.append(encryptedPayload);
         } else {// Fehlerfall
            encryptedPayload = readInputStream(huc.getErrorStream(), true);
            sbRet.append("Error:  " + responseCode + " " + encryptedPayload);
         }
         out("SCR flow 5: " + "HTTP Response: " + responseCode + " Payload: " + encryptedPayload);
         return sbRet.toString();
      } catch (SocketTimeoutException e) { // NOSONAR squid:S1166 SocketTimeout
         String msg = "Socket Timeout Exception. " + e.getMessage();
         out("Error:  " + msg);
         responseCode = -4;
         sbRet.append(msg);
      } catch (UnknownHostException e) { // NOSONAR squid:S1166 UnknownHostException
         String msg = "UnknownHostException. " + e.getMessage();
         out("Error:  " + msg);
         responseCode = -5;
         sbRet.append(msg);
      } catch (ConnectException e) { // NOSONAR squid:S1166 ConnectException ist
                                     // eindeutig - Stacktrace sinnlos
         String msg = "ConnectException. " + e.getMessage();
         out("Error:  " + msg, e);
         responseCode = -6;
         sbRet.append(msg);
      } catch (javax.net.ssl.SSLHandshakeException e) { // NOSONAR squid:S1166
                                                        // fuer
                                                        // SSLHandshakeException
                                                        // ist der
                                                        // Stacktrace-Inhalt
```

```
                                                        // ohne Belang
         String msg = "SSLHandshakeException beim Senden. " + e.getMessage();
         out("Error:  " + msg, e);
         responseCode = -7;
         sbRet.append(msg);
      } catch (Throwable e) { // NOSONAR
                              // checkstyle:com.puppycrawl.tools.checkstyle.checks.coding.
IllegalCatchCheck
                              // 3rdParty (HTTP) Calls mit diversen
                              // Exceptionfaellen
         String msg = "Error during scr request. " + e.toString();
         out("Error:  " + msg);
         responseCode = -8;
         sbRet.append(msg + " " + e.getMessage());
      }
      out("Info: " + "scr response: " + sbRet.toString());
      return sbRet.toString();
   }

   /**
    * reads out http response stream.
    *
    * @param istr
    *           stream to read
    * @param supressException
    *           in case of error an empty string will returned
    * @return stream content as sting
    * @throws IOException
    * @throws IllegalArgumentException
    */
   public static String readInputStream(InputStream istr, boolean supressException) throws
IOException {
      StringBuilder sb = new StringBuilder();
      if (istr == null && supressException) {
         return "";
      }
      if (istr == null) {
         throw new IllegalArgumentException("istr must not be null");
      }
      BufferedReader in = null;
      try {
         in = new BufferedReader(new InputStreamReader(istr, "UTF-8"));
         String row = "";
         while ((row = in.readLine()) != null) {
            sb.append(row);
         }
         in.close();
      } catch (IOException e) {
         out("Error: readInputStream() Fehler beim Lesen eines HTTP Streams. {0}", e);
         if (!supressException) {
            throw e;
         }
      } finally {
         if (in != null) {
            in.close();
         }
      }
      return sb.toString();
   }
   /**
    * Call scr without encryption.
    * calls {@link #callScr(String, String, String, String, String)} with emphy key and empty
keyhash.
    *
    * @param scrUrl
    * @param authString
    * @return
    */
   public String callScr(String scrUrl, String authString) {
```

```
      return callScr(scrUrl, authString, "", "");
   }
   /**
    * console output with parameter substitution
    *
    * @param message
    * @param args
    */
   public static void out(String message, Object... args) {
      if (args.length == 0) {
         System.out.println(message);
      } else {
         System.out.println(MessageFormat.format(message, args));
      }
   }
}
```

public class ScrCryptoHelper

ScrCryptoHelper

```
package de.deutschepost.postident.scrClient;
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.interfaces.RSAPrivateKey;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.text.MessageFormat;
import java.text.ParseException;
import java.util.Base64;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;
import com.nimbusds.jose.JOSEException;
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 {
      RSADecrypter ret;
      ret = new RSADecrypter(base64ToPrivateKey(rsaPrivateKey));
      return ret;
   }
   /**
    * Converts a Base64 String to RSAPrivateKey.
    * SCR flow 6.2: instantiate RSAPrivateKey
    *
    * @param base64Bytes
    * @return the RsaPrivateKey
    * @throws ScrException
    */
   public static RSAPrivateKey base64ToPrivateKey(String base64Bytes) throws ScrException {
      byte[] keybytes;
      try {
         keybytes = Base64.getDecoder().decode(base64Bytes);
      } catch (Throwable e) { // NOSONAR
         // checkstyle:com.puppycrawl.tools.checkstyle.checks.coding.IllegalCatchCheck
         // 3rdParty Calls
         out("Warning: " + "PrivateKey creation error. Base64 conversion error. {0}", e);
         throw new ScrException("PrivateKey creation error. Base64 conversion error.", e);
      }
      RSAPrivateKey retKey;
      try {
         retKey = byteToPrivateKey(keybytes);
      } catch (InvalidKeySpecException | NoSuchAlgorithmException e) {
         out("Warning: " + "PrivateKey creation error {0}", e);
         throw new ScrException("PrivateKey creation error.", e);
      }
      return retKey;
   }
   /**
    * Convert a ByteArray to RSAPrivateKey.
    * Part of SCR flow 6.2: instantiate RSAPrivateKey
    *
    * @param keybytes
    *           Bytes of Key.
    * @return the RsaPublicKey
    * @throws InvalidKeySpecException
    * @throws NoSuchAlgorithmException
    */
   public static RSAPrivateKey byteToPrivateKey(byte[] keybytes)
```

```
         throws InvalidKeySpecException, NoSuchAlgorithmException {
      return (RSAPrivateKey) KeyFactory.getInstance("RSA").generatePrivate(new PKCS8EncodedKeySpec
(keybytes));
   }
   /**
    * Calculates sha256 hmac over an base64 encoded payload.
    * SCR flow step 1: Calculate HMAC of public key
    *
    * @param dataPassword
    *           HMAC secret - will be converted in teh utf8 byte representation.
    * @param publicKeyBase64
    *           Base64 encoded payload - will be decoded to bytearray befor hashing
    * @return der Base64 encoded HMAC hashes
    * @throws UnsupportedEncodingException
    * @throws NoSuchAlgorithmException
    * @throws InvalidKeyException
    */
   public static String hmacHashOverKey(String dataPassword, String publicKeyBase64)
         throws UnsupportedEncodingException, NoSuchAlgorithmException, InvalidKeyException {
      String hmacHashBase64 = "";
      // SCR flow step 1.1: convert datapassword to byte[]
      byte[] dataPasswordBytes = dataPassword.getBytes("UTF-8");
      out("SCR flow 1.1: dataPassword Bytes (.getBytes(\"UTF-8\")): " + toHexString
(dataPasswordBytes));
      // SCR flow step 1.2: convert RSA public key to byte[]
      byte[] publicKeyBytes = Base64.getDecoder().decode(publicKeyBase64);
      System.out.println("SCR flow 1.2: public Key Bytes (.getBytes(\"UTF-8\")): " + toHexString
(publicKeyBytes));
       // SCR flow step 1.3:  create and initialize javax.crypto.Mac
       // i). create HmacSha secretKey from Datapassword
      SecretKeySpec signingKey = new SecretKeySpec(dataPasswordBytes, HMAC_SHA256_ALGORITHM);
      // ii) instantiate and initialize mac
      Mac mac = Mac.getInstance(HMAC_SHA256_ALGORITHM);
      mac.init(signingKey);
      // SCR flow step 1.4: calculate HMAC hash bytes
      mac.update(publicKeyBytes);
      byte[] hmacHashBytes = mac.doFinal();
      out("SCR flow 1.4: (hmac hash bytes): " + toHexString(hmacHashBytes));
      // SCR flow step 1.5: convert hmac bytes to base64 form
      hmacHashBase64 = Base64.getEncoder().encodeToString(hmacHashBytes);
      out("SCR flow 1.5: (hmac hash base 64): " + hmacHashBase64);
      return hmacHashBase64;
   }
   /**
    * calcutates HTTP Basic Authorization String (SCR flow step #2)
    *
    * @param username
    * @param password
    * @return the Authorization string
    * @throws UnsupportedEncodingException
    */
   public static String calcBasicAuthString(String username, String password) throws
UnsupportedEncodingException {
      String authString = username + ":" + password;
      out("SCR flow 2.1: Basic Auth usernamepassword string:  {0} ", authString);
      out("SCR flow 2.1a: Basic Auth usernamepassword bytes[]:  {0} ",
            ScrCryptoHelper.toHexString(authString.getBytes("UTF-8")));
      authString = Base64.getEncoder().encodeToString(authString.getBytes("UTF-8"));
      out("SCR flow 2.2: Basic Auth usernamepassword base64:  {0} ", authString);
      authString = "Basic " + authString;
      out("SCR flow 2.3: complete Basic Auth string:  {0} ", authString);
      return authString;
   }
   /**
    * wandelt eine Bytefolge in einen HexString um.
    *
    * @param pArray
    *           bytearray.
    * @return Strin mit hexadezimalziffern.
```

```
    */
   public static String toHexString(byte[] pArray) {
      return DatatypeConverter.printHexBinary(pArray);
   }
   /**
    * wandelt einen HexString in ein Bytearray um.
    *
    * @param pHexStr
    *           string mit Hexadezimalziffern.
    * @return das dem String entsprechende Bytearray.
    */
   public static byte[] toByteArray(String pHexStr) {
      return DatatypeConverter.parseHexBinary(pHexStr);
   }
   /**
    * Konsolenausgabe mit Paramterersetzung
    *
    * @param message
    * @param args
    */
   public static void out(String message, Object... args) {
      if (args.length == 0) {
         System.out.println(message);
      } else {
         System.out.println(MessageFormat.format(message, args));
      }
   }
}
```

public class RsaKeyBytes

RsaKeyBytes

```
package de.deutschepost.postident.scrClient;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
/**
 * RsaKeyBytes. RSA keypair container and generator
 * @author Deutsche Post AG
 * @version 1.0
 */
   public class RsaKeyBytes {
      private int keyLength = 0;
      private byte[] privateKeyBytes = new byte[0];
      private byte[] publicKeyBytes = new byte[0];

      /**
       * generates an RSA keypair
       *
       * @param keyLength 2048, 3072 oder 4096 Bit
       * @return generated keypair
       * @throws NoSuchAlgorithmException
       */
      public static RsaKeyBytes build(int keyLength) throws NoSuchAlgorithmException {
         RsaKeyBytes rkb = new RsaKeyBytes();
         rkb.setKeyLength(keyLength);
         // use KeyPairGenerater to generate RSA keypair
         java.security.KeyPairGenerator keyGen = java.security.KeyPairGenerator.getInstance("RSA");
         keyGen.initialize(keyLength);
         java.security.KeyPair kp = keyGen.genKeyPair();
         rkb.setPrivateKeyBytes(kp.getPrivate().getEncoded());
         rkb.setPublicKeyBytes(kp.getPublic().getEncoded());
         return rkb;
      }
      public String getPrivateKeyStr() {
         return Base64.getEncoder().encodeToString(privateKeyBytes);
      }
      public String getPublicKeyStr() {
         return Base64.getEncoder().encodeToString(publicKeyBytes);
      }

      public byte[] getPrivateKeyBytes() {
         return privateKeyBytes;
      }
      public byte[] getPublicKeyBytes() {
         return publicKeyBytes;
      }
      public void setPrivateKeyBytes(byte[] privateKeyBytes) {
         this.privateKeyBytes = privateKeyBytes;
      }
      public void setPublicKeyBytes(byte[] publicKeyBytes) {
         this.publicKeyBytes = publicKeyBytes;
      }
      public int getKeyLength() {
         return keyLength;
      }
      public void setKeyLength(int keyLength) {
         this.keyLength = keyLength;
      }
      /* (non-Javadoc)
       * @see java.lang.Object#toString()
       */
      public String toString(){
         String ret = "RSA keypair ( keysize " + keyLength + " bit)";
         ret += "\r\nPublic Key (Base64) : " + Base64.getEncoder().encodeToString(publicKeyBytes);
         ret += "\r\nPrivate Key (Base64): " + Base64.getEncoder().encodeToString(privateKeyBytes);
         return ret;
      }

   }
```

Usage of ScrClientTool

usage: ScrClientTool
-?,--help show help
-a,--auth generates an HTTP Basic athorization String (requires <username> and <password>)
-c,--callscr Calls scr service and decrypts the scr response.
<username>, <password> and <scrhost> are always required
mode 1: <datenpassword> provided - generate a new keypair and handle encrypted scr request
mode 2: <pubkey>, <privkey> and <keyhash> provided - handle encrypted scr request with given keys
mode 3: <pubkey>, <privkey> and <datapassword> provided - generate hash and handle encrypted scr request with given keys
mode 4: nothing else provided - handle unencrypted scr request(only possible in test environment)
--cipher <cipher> RSA encrypted SCR response
--clientId <clientId> SCR ClientID (provided by the Deutsche Post technical sales or service team)
-d,--decrypt decrypts <cipher> by using <privkey>
--datapassword <datapassword> datapassword to calculate hmac-hash (provided by the Deutsche Post technical sales or service team)
--dir <dir> log directory; location for logging input and output values from called operation
-h,--hash generate HMAC-hash over <pubkey> using <datapassword> as secret
-k,--genkey generate RSA 3072 keypair
--keyhash <keyhash> HMAC-hash of pubkey (Base64)
--password <password> password for SCR Service (provided by the Deutsche Post technical sales or service team)
--privkey <privkey> RSA private key (Base64) private key will be used to decrypt scr response (not transmitted to scr service)
--pubkey <pubkey> RSA public key(Base64) public key will be transmitted to scr service and becomes used to engrypt the service response
--scrhost <scrhost> scr-hostname (production: postident.deutschepost.de, test and integration: postident-demo.deutschepost.de)
--username <username> username for SCR Service (provided by the Deutsche Post technical sales or service team)

Hints

- Compile the Code above into an executable jar file (e.g. ScrClientTool.jar)
- Please set Commandline and Java in UTF-8 Mode
  - Commandline: *chcp 65001*
  - Java *-Dfile.encoding=UTF8*
  - ⚠️ Otherwise special characters in passwords will be missinterpreted (for instance "§")
- You have to install Java Chryptography extensions to your Java Environment
  - See http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html
  - ⚠️ Otherwise RSA operations with 2k key shrength will fail.
- ⚠️ please take care when calling the scr Service; accounts will be locked after 5 consecutive failed login attempts

ScrClientTool -k : generate RSA 3072 keypair

```
c:\<your_path>\scr>java -jar .\ScrClientTool.jar --dir c:/<your_path>/scr/ -k
RSA Keypaar ( Keysize 3072 Bit)
Public Key (Base64) : MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAssfvz3Ej7Mu
/NdVmeUp7rVL3f4MlZiBrc3BaM6JOF/Q+zsUQVlqwsh9Rp69
/TL8HvpkRzMLZxroZMpXhkxnQ2lWtQd1Q8+zKHy0TJ588g99H1AHKgLXeIa9DcXZMUDqrfyc6Bfy/9inB
/Y17chThbNzICGcu4p3IBwyp7dyKOK7yifQS3i6x8uZ9VRIGOh32+PSniX8HzCGWegamDyiOjSaH5Y1QO9cBleeKoqP
/5JGqie+50bdGQyUAZoGyfqHHTcJg+y237I84X/YRLenH5ds4rycsM1PkTypZRu9Ajtnyc9VnqNSuR56kSOePU
/bvDlPahh0cOuyIGToBhgPxGwIDAQAB
Private Key (Base64): MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQCyx+/PcSPsy7811WZ5SnutUvd
/gyVmIGtzcFozok4X9D7OxRBWWrCyH1Gnr39Mvwe+mRHMwtnGuhkyleGTGdDaVa1B3VDz7MofLRMnnzyD30fUAcqAtd4hr0NxdkxC
```

ScrClientTool -h : calculate keyhash

```
c:\{your_path}>chcp 65001
Aktive Codepage: 65001.
c:\{your_path}>java -Dfile.encoding=UTF8 -jar .\ScrClientTool.jar -h --datapassword xR7ea2_53S(m --
pubkey
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA2VWUWGfebYsAgvOoqk0iYkZbhJCVBVFDWf+kkc2kIULb+JYUF+VdRKKQM
```

ScrClientTool -a : calculate Basic auth String

```
c:\{your_path}>chcp 65001
Aktive Codepage: 65001.
c:\{your_path}>java -Dfile.encoding=UTF8 -jar .\ScrClientTool.jar -a --username SCRDEMO --password
3r#4Mu#GBRmP
Calculate basic authorization header user SCRDEMO password 3r#4Mu#GBRmP
SCR flow 2.1: Basic Auth usernamepassword string:  SCRDEMO:3r#4Mu#GBRmP
SCR flow 2.1a: Basic Auth usernamepassword bytes[]:  53435244454D4F3A337223344D75234742526D50
SCR flow 2.2: Basic Auth usernamepassword base64:  U0NSREVNTzozciM0TXUjR0JSbVA=
SCR flow 2.3: complete Basic Auth string:  Basic U0NSREVNTzozciM0TXUjR0JSbVA=
Basic U0NSREVNTzozciM0TXUjR0JSbVA=
```

ScrClientTool -c : call SCR Service and decrypt Response

Mode: create new RSA Keypair

```
c:\{your_path}>chcp 65001
Aktive Codepage: 65001.
c:\{your_path}>java -Dfile.encoding=UTF8 -jar .\ScrClientTool.jar -c --username "SCRDEMO" --
password "y!7V@bAq8T@a"  --datapassword "N9HXc-jCqC§Q"  --dir c:/{your_path}/scr/  --clientId
865E6E37 --scrhost postident.deutschepost.de
Info: calling scr Service
SCR flow precondition 1: postident hostname: postident.deutschepost.de
SCR flow precondition 2: SCR username, password, clientId (SCRDEMO, y!7V@bAq8T@a, 865E6E37)
SCR flow precondition 3: SCR datapassword: N9HXc-jCqC§Q
datapassword provided -> generate keypair
SCR flow precondition 4: generated RSA keypair: RSA keypair ( keysize 3072 bit)
Public Key (Base64) :
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAhweW7boCgMEXd1m215I10vnNi4cP5kC3j7u2tLOCdyGPASFdq9B3wYC1x
```

ScrClientTool -c : call SCR Service with provided keys and decrypt Response

```
c:\{your_path}>chcp 65001
Aktive Codepage: 65001.
c:\{your_path}>java -Dfile.encoding=UTF8 -jar .\ScrClientTool.jar -c --username "SCRDEMO" --
password "y!7V@bAq8T@a"  --datapassword "N9HXc-jCqC§Q"  --dir c:/{your_path}/scr/  --clientId
865E6E37 --scrhost postident.deutschepost.de --pubkey
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA2VWUWGfebYsAgvOoqk0iYkZbhJCVBVFDWf+kkc2kIULb+JYUF+VdRKKQM
```

ScrClientTool -d decrypt

```
c:\{your_path}>chcp 65001
Aktive Codepage: 65001.
c:\{your_path}>java -Dfile.encoding=UTF8 -jar .\ScrClientTool.jar -d  --dir c://{your_path}/scr --
privkey MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKkwggSlAgEAAoIBAQDZVZRYZ95tiwCC86iqTSJiRluEkJUFUUNZ
/6SRzaQhQtv4lhQX5V1EopA0fF3QPjOVQ3jwehl9mWoX1+rYNdCOE78Kx0O7A3M0PA+YMTZa6h5UYNby8V8+x8JVk+/nUkgf
/T1kh8hIn111VkAvzp1Olas7PNPvjPhWlmMtJfIXzKhfEuWXD+lOxh1PbFzN/WFH8
/7EU+AOwJKP7bIBfbf5zU7fUKNxvbox5z6HXdh1kKulOTm5UwWix2ZuAyK
/nE+qQ7xyanqpigv3ekua8c3yhvu0TuWx+1bE4D8CwXS
/FZNCZfxmGDsSUKqqaOPRpqjkA09g6zVAXFHrhiqKAervAgMBAAECggEBAM0milR1H7IZnw79v72Z+BX6QhM9g120mrYOqaL0EbVP
```

Java Snipplets

Java Snipplet to Calculate the HMAC-Hash

hmac Hashing

```
/** used hashing Algorithm. */
private static final String HMAC_SHA256_ALGORITHM = "HmacSHA256";
/**
* Calculates hmacsha256 hash over a sequence of bytes, provided in base64 format.
* The provided secret String will be converted into bytes by utf-8 encoding.
*
* @param dataPassword HMAC secret (String)
* @param base64encodedKey public key base64 encoded - to calculate the hash, this base64 will be
decoded into bytes
* @return hmac hash in base64 form
*/
 public static String hmacHashOverKey(String dataPassword, String base64encodedKey) {
     String ret = "";
     try {
        byte[] secret = dataPassword.getBytes("UTF-8");
        SecretKeySpec signingKey = new SecretKeySpec(secret, HMAC_SHA256_ALGORITHM);
        Mac mac = Mac.getInstance(HMAC_SHA256_ALGORITHM);
        mac.init(signingKey);
        byte[] encryptionKeyBytes = Base64.getDecoder().decode(base64encodedKey);
        mac.update(encryptionKeyBytes);
        byte[] payloadHash = mac.doFinal();
        ret = Base64.getEncoder().encodeToString(payloadHash);
     } catch
         (NoSuchAlgorithmException | InvalidKeyException | UnsupportedEncodingException e) {
            LOGGER.error("hmacHashOverKey Error", e); }
      return ret;
   }
```

RSA Java Snipplet to Decrypt the JWE Response

rsaDecrypt

```
public static String decryptRSA(String jweString, String rsaPrivKey) throws
InvalidKeySpecException, NoSuchAlgorithmException, ParseException, JOSEException{
       String ret = "";
       RSAPrivateKey rpk = base64ToPrivateKey(rsaPrivKey) ;
       JWEObject jweObject = JWEObject.parse(jweString);
       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));
}
```

PHP Client Sample

See PHP JOSE (Javascript Object Signing and Encryption) implementation with JWE sample: https://github.com/nov/jose-php. Requires library phpseclib: http://phpseclib.sourceforge.net.

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.

```
/* Installation:
- Download JOSE-PHP from https://github.com/nov/jose-php
- extract and run: php composer.phar install
- copy this file into directory examples
- run: php piscr.php */
require dirname(__FILE__) . '/../vendor/autoload.php';
use phpseclib\Crypt\RSA;
class PISCR {
  var $jwe;
  var $url;
  var $header;
  var $xscr_alg;
  var $xscr_enc;
  var $public_key;
  var $private_key;
  var $authorization;
  var $payload;

/* getPayload
- send http-Request
- decode and decrypt response using JOSE_JWE
- user/pass authorisation possible  */
function getPayload($user=false,$pass=false)  {
      $ch = curl_init();
      if($ch === false)   {
          die('Failed to create curl object');
      }
      curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
      curl_setopt($ch,CURLOPT_HTTPHEADER,$this->header);
      $timeout = 5;
      url_setopt($ch, CURLOPT_URL, $this->url);
      curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
      curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, $timeout);
      if($user != false || $pass != false) {
          curl_setopt($ch,CURLOPT_USERPWD,$user.":".$pass);
      }
      $data = curl_exec($ch);
      if($data === false) { echo 'Curl error: ' . curl_error($ch);   }
      curl_close($ch);
      $this->jwe = JOSE_JWE::decode($data);
      $this->jwe->decrypt($this->private_key);
      $this->payload = $this->jwe->plain_text;
   }

// generate array with header-content
function setHttpHeader() {
   if(strlen($this->xscr_enc) > 0)
        $this->header[] = 'x-scr-enc:'.$this->xscr_enc;
   if(strlen($this->xscr_alg) > 0)
        $this->header[] = 'x-scr-alg:'.$this->xscr_alg;
   if(strlen($this->public_key) > 0)
        $this->header[] ='x-scr-key:'.$this->exctractKeyFromCert($this->public_key);
   if(strlen($this->public_key) > 0)
       $this->header[] = 'x-scr-keyhash:'.base64_encode(hash_hmac('sha256', base64_decode($this-
>exctractKeyFromCert($this->publicKey)), $this->dataPassword, TRUE));
   if(strlen($this->authorization) > 0)
       $this->header[] ='Authorization:'.$this->authorization;
  }

// file-content can be surrounded by BEGIN/END text - then remove
function exctractKeyFromCert($cert) {
     $key = preg_replace('#.*?^-+[^-]+-+#ms', '', $cert, 1);      // remove the -----BEGIN
CERTIFICATE----- and -----END CERTIFICATE----- stuff
     $key = preg_replace('#-+[^-]+-+#', '', $key);         // remove new lines             $key =
str_replace(array("\r", "\n", ' '), '', $key);
     return $key;
  }
}
```

```
$pi_scr = new PISCR();

//********* use keygeneration **************
$rsa = new RSA();
$rsakeys = $rsa->createKey(3072);
$pi_scr->private_key = $rsakeys['privatekey'];
$pi_scr->public_key = $rsakeys['publickey'];
//********* use keygeneration **************

//********* use certificates from file ******
//$pi_scr->public_key  = file_get_contents(dirname(__FILE__) . '/../test/fixtures/public_key.
pem');
//$pi_scr->private_key = file_get_contents(dirname(__FILE__) . '/../test/fixtures/private_key.
pem');
//********* use certificates from file ******

//********* URL **********************
$pi_scr->url = 'http://postident.deutschepost.de/api/scr/v1/5AA2A915/cases/V7EBT0W8W8KQ';
//********* URL **********************

//********* Http-Header **************
$pi_scr->authorization = 'Basic U0NSV1VMOlFqVk53d1QzUjgodQ==';
$pi_scr->xscr_alg = 'RSA1_5';
$pi_scr->xscr_enc = 'A256CBC-HS512';
//********* Http-Header **************
$pi_scr->setHttpHeader();
$pi_scr->getPayload();
var_dump(json_decode($pi_scr->payload));
```