POSTIDENT

SCR-Ident API Guide 3 - Encryption

Standard Connect & Result (SCR) API

Version 1.1
06.03.2023

SCR-Ident API Guide 3 Encryption

Contents

1. Overview . . . 2
   - Preconditions . . . 2
   - General Flow . . . 2
2. Encryption Settings using header . . . 3
   - HTTP-Header . . . 3
   - Standard Encryption . . . 4
3. Detailed Flow with Samples . . . 4
   - Preconditions . . . 5
   - RSA key pair generation . . . 6
     - Sample RSA 3072 key pair . . . 6
     - Code sample RSA keypair generation . . . 7
   - Calculate HMAC of public key . . . 8
     - Code Sample for HMAC calculation . . . 8
   - Handle the get Request in 3 JUnit Tests . . . 9
   - Test case 1 Processing of the HTTP request without encryption . . . 9
   - Test case 2 Processing of the HTTP request with encryption Parameters (no decryption on client side) . . . 10
   - Test case 3 Processing of the HTTP request with encryption Parameters (with decryption on client side) . . . 11
   - Sample Response of SCR getCases full: . . . 12
4. Errors . . . 14
   - Sample Error Messages . . . 14
   - Encryption Errors . . . 15
   - Typical error situations and error messages . . . 15
5. Code Samples . . . 16
   - Disclaimer . . . 16
   - Import the provided Eclipse Project . . . 16
   - Java SCR Sample JUnit Tests . . . 16
     - java source . . . 16
       - public class ScrCaller . . . 17
       - public class ScrCryptoHelper . . . 19
       - Class ScrCallTests . . . 23
   - Java Snipplets . . . 25
     - RSA Java Snipplet to Decrypt the JWE Response . . . 25
   - PHP Client Sample . . . 26
   - Python Client Sample . . . 28

© 2022 Deutsche Post AG

Changelog

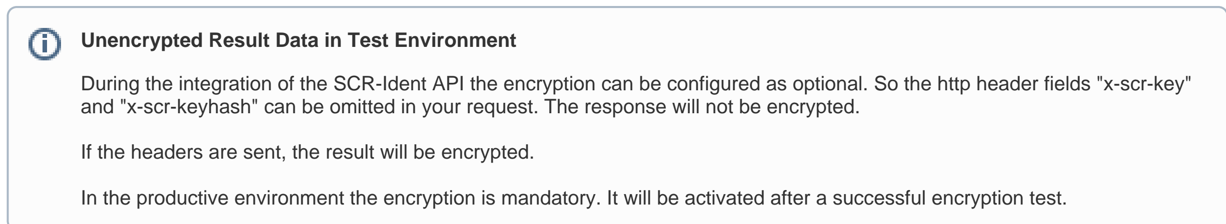
Date	Change
06.03.2023	adaptation of the description to the current implementation, revision of the implementation guidelines
30.07.2021	Several Encryption settings marked as deprecated (Q3/2022)
05.10.2020	Clarification on the format of the format of the public key in x-scr-key
22.09.2020	Updated recommendations 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. 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 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.

General Flow

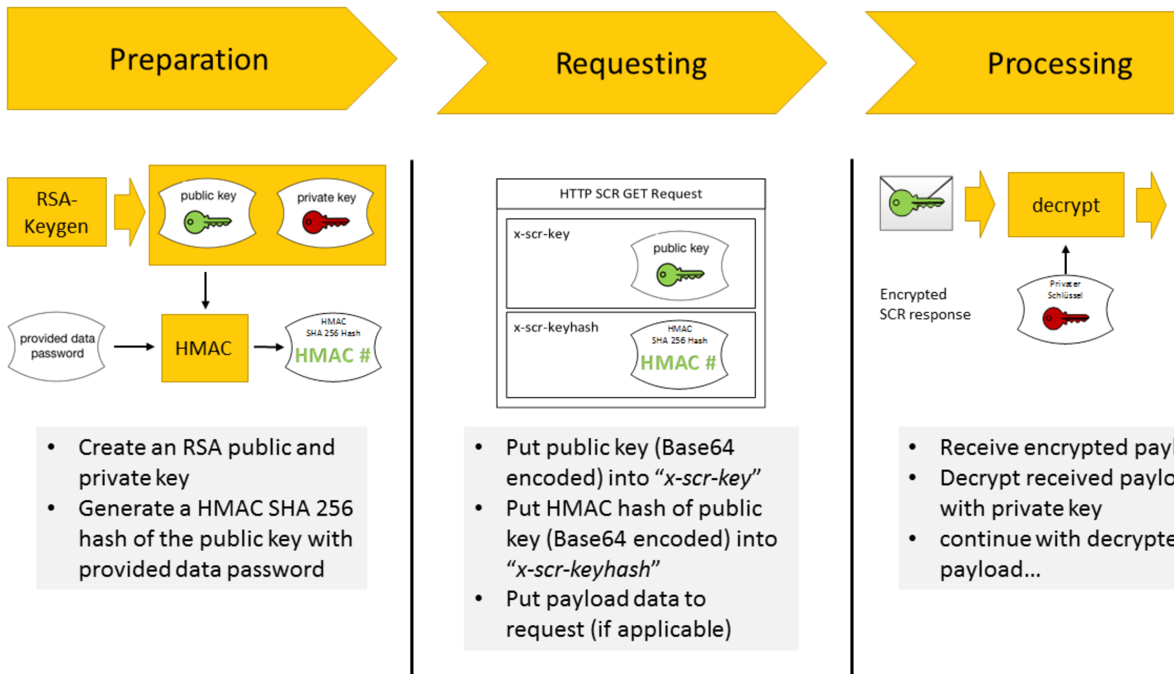
The asymmetrical encryption works as follows:

- 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:

© 2022 Deutsche Post AG



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

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

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

Standard Encryption

By default, the following parameters are used:

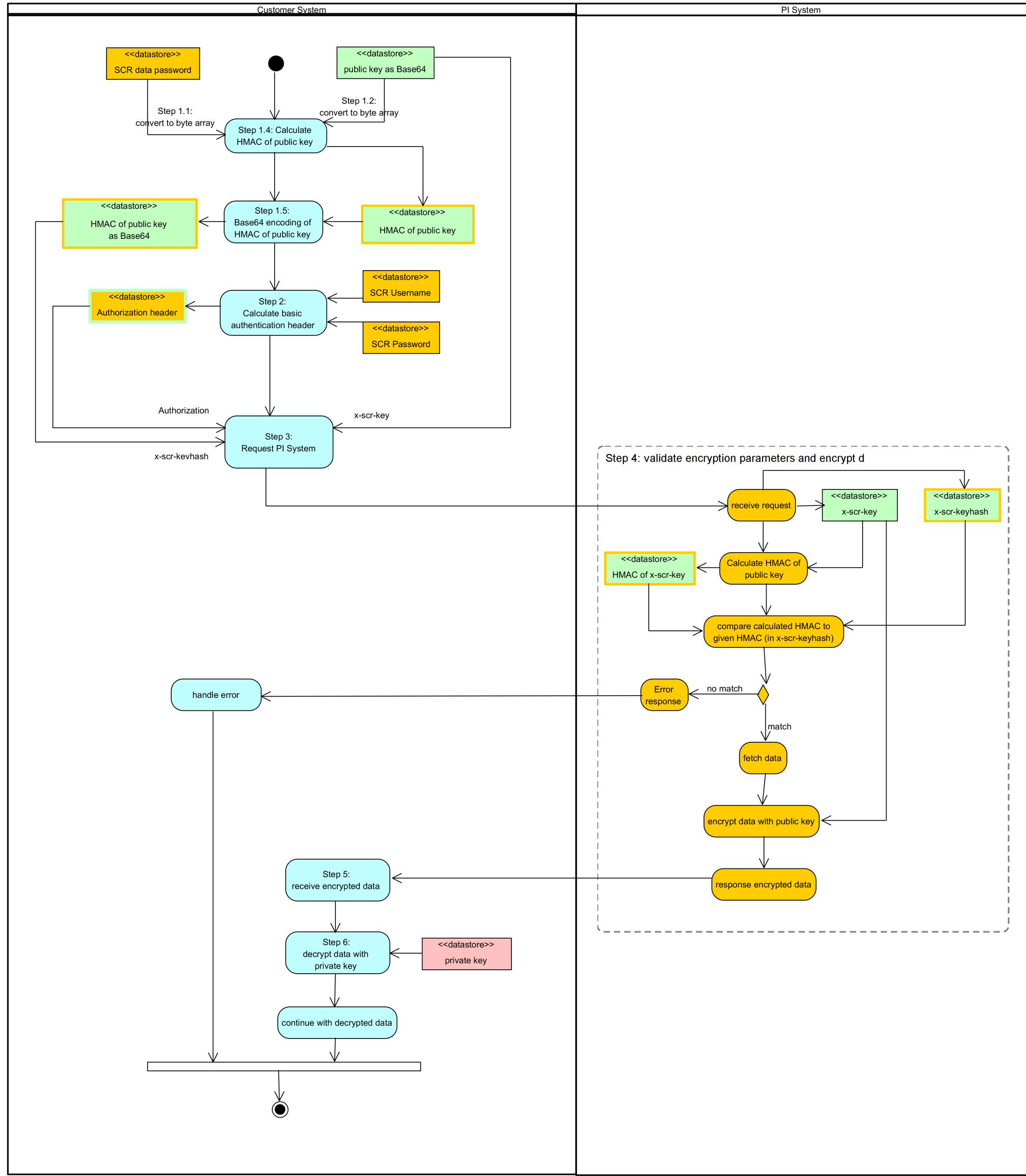
Property	Value	Description
JWE algorithm of the response	**RSA-OAEP-256**	RSAES using Optimal Asymmetric Encryption Padding (OAEP) - RFC 3447 with the SHA-256 hash function and the MGF1 with SHA-256 mask generation function
JWE encryption of the response	**A256CBC-HS512**	AES_256_CBC_HMAC_SHA_512 authenticated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

3. Detailed Flow with Samples

SCR encrypted communication

Preconditions:

- **Postident hostname:** postident.deutschepost.de | postident-demo.deutschepost.de
- **SCR username, password, clientId:** provided by Deutsche Post AG through integration process
- **SCR data password:** provided by Deutsche Post AG through integration process
- **RSA public/private key:** generated by customer

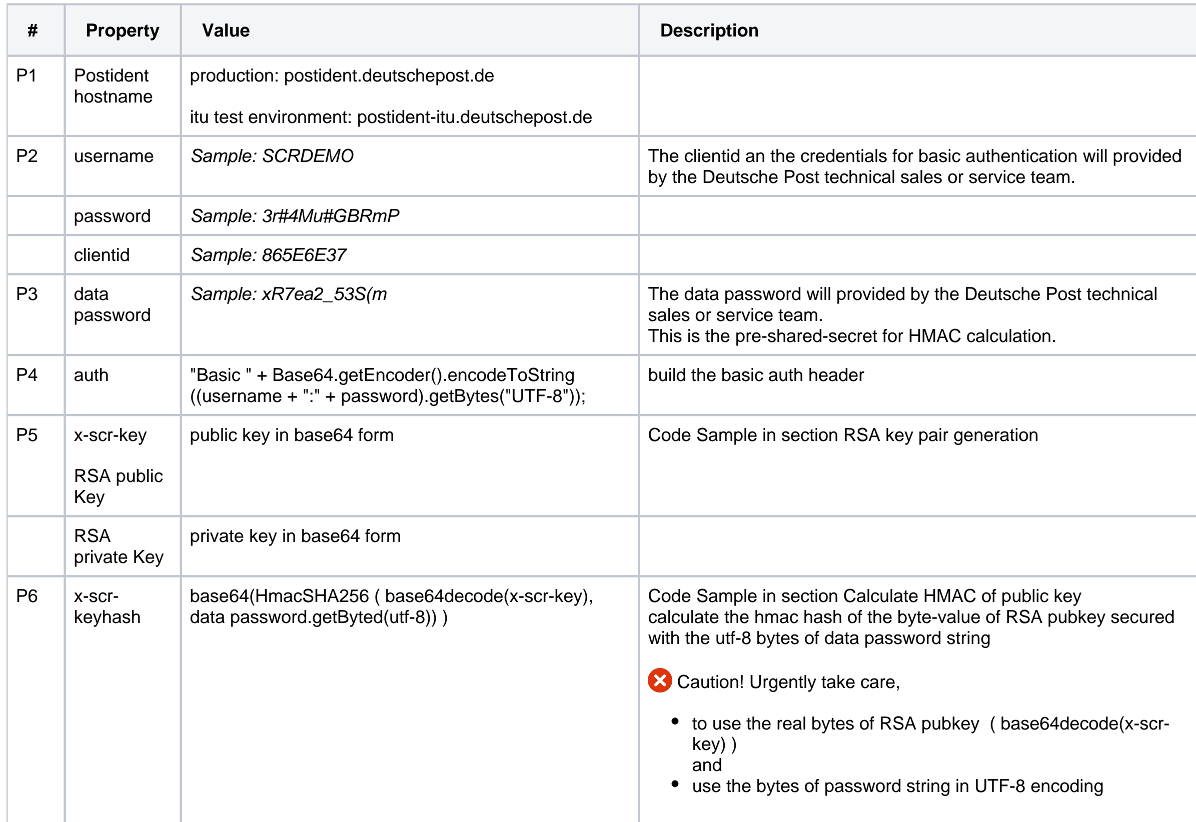


Preconditions

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

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

© 2022 Deutsche Post AG



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

RSA key pair generation

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

Sample   RSA 3072 key pair

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


  ======================================= RSA Private Key =======================================
privKeyBase64:
MIIG/QIBADANBgkqhkiG9w0BAQEFAASCBucwggbjAgEAAoIBgQDbvAcOTxBDYl/SlZVFcK4HhahbKzHm
6eU08ekcsFHMR627v9xkdRFE+5Ut/tqEX3gfs/2ybNWuQ+oTCJyJS+35TB1MeqiVZhmfRQH4Un0cssYV
FIY5PRq0glNfwB9beoWUmpMo+Ta6uvn2eXl/0VrJCO271ghT9Gl//wHHGe/HD9BS47Ak1gScGiS26v48
xzgNas0tqknp/zqW4T3qkHDiMSX5pIgPQr6jqV7OBUy5MkSkPTPnodBWzU7wiHB0Xk/TtbG0LL3ZXdY3
PgHrXW9QfLfQm6OllDiu8ewhg8BAPru6dLiSBraAZWSbLIxDt15NSx1047X/FToLN3UW/oFcwD3zANMW
2R9Cbk+hRyIS0U6aNtjYB4XiRBD2b0kc/tOip8oPidQFyqTu9YFygCZFdfYTr1WqR+OpKdj/Zzhtf4zt
jLvjfFgQWk21YX8+c9dt2OheZKavT5/NpX6IfT6Jax6CoRVW0kI45rxeu8vARQPo03W5e7zUSVmLGz7k
KjsCAwEAAQKCAYBSs93p/ksOOaf8MA2rQaJWtXEsw0C4ex8oQsas5BYdb3sN18QhUUiLMK+wzKO9P9uL
/yhE8B6qxOgEevDD4j6y3nmGkAIEnRWWsxfvOUjatNTGGV/9iiiTeRvdRYfn2+EMEfPCb64wMPCk2oqm
l4q2cLRdxsyCoNLWeInYQMRVJXgQwp7I9n5srJBWAnNnN8o6jpCkF9xTBJuuLx+pOe0qzTnQUKNfLw1m
tvg2TZtXdvqL4Hc5PteV3BdSg9/jhJ0O9+LpPc6XPeyeX5W2wjcMlUjHqleZr25JQUGTpz2Xc/PyMF9a
ajjrFvI2JQaapCbbogX5xT1WBxgUIj0gSXattZE2YA+LYN0p3kWHALfjflJ/hEgXQlvThcc/i7NPuwyZ
9LmBUl0nnsUh+ZrBg+qb20d6hR801uUD9rOiIQCPqM+4MszPZEuPLGgw7XyDOrw8ridyQOuIOs/MkwW8
WUwGkBzTgfJhxNOcAjbFnEVvdqpV9brxcUai7HZpCyuOB5ECgcEA/3gQGwVJebtBW+ZBWFyP2XeJMAVm
3ouKmDcixAaPPkGO3WqjW2PuOcOjc3reL2qGNBdp15RuXhoQgzrriBhbD7+fLxFoEUh8r3r4Ee59fC4V
R+duZcAgzn7mGKL06Dmxn1pWRoIMi/oiFf4qASqKHixRG16Qre/qx/Aq148wwpVHW8aj3qi/O8WhMpyL
C0B9a67x9iZuZHpv4oECM/8m2VwKUQDTlTfgIfPT8x4uZm79FWoYNu4Sov0oqZelrKCLAoHBANww8zU2
bAtbUxHmcdL+Ep4A4GJ7IEslYxpCA6rE032h1JPjhsgkFCpCMm9ULuPB7GVO8FCi/G8np04k4tQCGAyk
owVo91F3/bHDzKg53eQNrlHHK+5v0Wig5FdrJYjzesTzXNE5vHGXIZlStLJvxydbNi6f+GKolUYrRiUD
q1rXblH/fzcH/OeoHTZMkditXGl04MeLZsVI1XyyabUO+Mbn1tCAbA/QGRwvxkrsUbMxhRwHZh9OpA/g
w+2Ktj+jEQKBwQCt1NKQsE5IyMQ2djzslT9EqHPu+x3+PFE4NfwwCo6ZvB4sR42UeDD3mUm1IFvjQ8bB
11Sr3lIjMfUedo8h8gC0fJ0id+OqLvq4pSH7gZ5dSzYr0SxrjApGuE51YRzoj6lvVxbxrzg+Xxumf2Wx
mXKWCW2/A9/Kmz/UgHWGa+av4MfHiSikL4rAbz+6oo5yP5WYGliGN6xSLoXHSCVpoCqdKGXUrI1MMdMP
B0ynV1gXV1paofCMH/8KbNyoWYngJvUCgcAbg25QXt1/SckGepIMeDzhS2C+TX5KhYBtnoQQyYlXsGn8
lLiZt5Bhe2YL7Svyv5+HRSbYHJWIWh8Yr5k8Poq8wbGxEl2c625i0QF4n8pvHZqDk2yU3ZoABeKiXbwR
8cKMa5BUgG9gji7flqsHhNoQu9/4UXra8Q+99dhM9blqn3KkglVmDfR3CdWknEq5FWhvsPAJXwzXZ/Nt
Imn8+sLxL7Ty1qkqDKmmkP2pDQBkcLKePvpuvQfZCA/TVbrSFtECgcAUga7lVcG4j2z6UXl5DG/68YYG
REN+8ZfD3SmZn8rrAP8QuhgFHPe3u2ROCMjwWxYAgQ9Yf8uUKapZ/5O/fKcrf1ET/G26gsCxxj6i4PRq
ch6akyXuVlyR7WbGvqd89N3sdTS7jO/K/p8JB0GKwS2cOUdRsv7GdXBIcWl3ZL8ls7WWNLmucc4kk34G
LOSGzJbr4YY8d0DDbol+eUuIhbXZqqkKyb08x8N/86ytYQNVeOMO2qTpN4lVAF03i0qDAdA=
```


Code sample RSA keypair generation

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

Keygen Step	Description	java Snipplet	Data

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

2	generate keypair	java.security.KeyPair keypair = keyGen.genKeyPair();	
4	get public key in base64 form	String pubKeyBase64 = Base64.encodeBase64String(keypair.getPublic().getEncoded());	see codeblock above
6	get private key in base64 form	String privKeyBase64 = Base64.encodeBase64String(keypair.getPrivate().getEncoded());	see codeblock above

© 2022 Deutsche Post AG

initializeKeypair

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

Calculate HMAC of public key

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

Code Sample for HMAC calculation

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

OUT: base64 encoded HMAC hash

HMAC Step	Description	java Snipplet	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
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+aJLSPf83CmfkeXI8U0OhLHgI=

© 2022 Deutsche Post AG

hmacHashOverKey

```
        /**
            * 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 the 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 = "";
              // HMAC Step 1:  convert datapassword to byte[]
              byte[] dataPasswordBytes = dataPassword.getBytes("UTF-8");
              // HMAC Step 2:  convert RSA public key to byte[]
              byte[] publicKeyBytes = Base64.getDecoder().decode(publicKeyBase64);
               // HMAC Step 3:  create and initialize javax.crypto.Mac
               // i). create HmacSha secretKey from Datapassword
              SecretKeySpec hmacKey = new SecretKeySpec(dataPasswordBytes, HMAC_SHA256_ALGORITHM);
              // ii) instantiate and initialize mac
              Mac mac = Mac.getInstance(HMAC_SHA256_ALGORITHM);
              mac.init(hmacKey);
              // HMAC Step 4: calculate HMAC hash bytes
              mac.update(publicKeyBytes);
              byte[] hmacHashBytes = mac.doFinal();
              // HMAC Step 5: convert hmac bytes to base64 form
              hmacHashBase64 = Base64.getEncoder().encodeToString(hmacHashBytes);
              return hmacHashBase64;
           }
```

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

Test case 1 Processing of the HTTP request without encryption

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

Entry point: JUnit Test ScrCallTests.testGetCasesUnencrypted

Required Date for the Request

```
        /**
         * Processing of an SCR get request (unencrypted answer)
         *
         * The result is unencrypted because the headers x-scr-key and x-scr-keyhash are
         * not be set. Note: The production environment suppresses unencrypted result
         * querys
         */
        @Test
        void testGetCasesUnencrypted() {
                out("JUNIT Test testGetCasesUnencrypted");
                String ret = ScrHttpRequestHandler.callScr(scr_url_full_all, scr_user,
scr_password);
                out(ret);
                assertThatNoException();
                assertTrue(ret.startsWith("["));
        }
```

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

Required Data for the Request

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

The Processing of the HTTP Request is done by calling ScrRequestHandler

Calculate the required Parameter

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

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

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

Required Date for the Request

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

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

Calculate the required Parameter

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

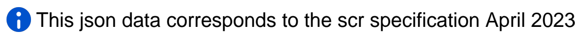
Test case 3 Processing of the HTTP request with encryption Parameters (with decryption on client side)

© 2022 Deutsche Post AG

Required Date for the Request

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

Sample Response of SCR getCases full:


This json data corresponds to the scr specification April 2023


```
[
	{
		"caseId": "K6JNXGBG2XVU",
		"caseStatus": {
			"status": "closed",
			"archived": false,
			"validUntil": "2023-03-23T14:02:04+01:00",
			"created": "2023-03-09T14:02:04+01:00",
			"modified": "2023-03-09T15:11:39+01:00"
		},
		"orderData": {
			"customData": {},
			"processData": {
				"targetCountry": "DEU",
				"preferredLanguage": "DE_DE",
				"referenceId": "K6JNXGBG2XVU",
				"callbackUrlCouponRequested": {},
				"callbackUrlReviewPending": {},
				"callbackUrlIncomplete": {},
				"callbackUrlSuccess": {},
				"callbackUrlDeclined": {}
			},
			"contactDataProvided": {},
			"identificationDocumentProvided": {},
			"drivingLicenceProvided": {
				"drivingLicenceClasses": []
			}
		},
		"contactData": {
			"title": {},
			"firstName": {
				"status": "new",
				"value": "Matthias"
			},
			"lastName": {
				"status": "new",
```

```
                "value": "Schulz"
            },
            "mobilePhone": {},
            "email": {
                "status": "new",
                "value": "terter.alpha@email.de"
            },
            "epost": {},
            "address": {
                "streetAddress": {},
                "appendix": {},
                "postalCode": {},
                "city": {},
                "country": {}
            }
        },
        "identifications": [
            {
                "identificationMethod": "basic",
                "identificationStatus": {
                    "status": "success",
                    "identificationTime": "2023-03-09T15:11:38+01:00",
                    "created": "2023-03-09T14:02:19+01:00",
                    "modified": "2023-03-09T15:11:39+01:00"
                },
                "identificationDocument": {
                    "type": {
                        "status": "new",
                        "value": "1"
                    },
                    "number": {
                        "status": "new",
                        "value": "sdadasdsa"
                    },
                    "firstName": {
                        "status": "new",
                        "value": "Matthias"
                    },
                    "lastName": {
                        "status": "new",
                        "value": "Mustermann"
                    },
                    "birthName": {},
                    "birthDate": {
                        "status": "new",
                        "value": "1977-01-11"
                    },
                    "birthPlace": {
                        "status": "new",
                        "value": "Paddington"
                    },
                    "nationality": {
                        "status": "new",
                        "value": "DEU"
                    },
                    "address": {
                        "streetAddress": {
                            "status": "new",
                            "value": "Sackgasse 1"
                        },
                        "appendix": {},
                        "postalCode": {
                            "status": "new",
                            "value": "12345"
                        },
                        "city": {
                            "status": "new",
                            "value": "Rostock"
                        },
```

```
                                "country": {
                                        "status": "new",
                                        "value": "DEU"
                                }
                        },
                        "dateIssued": {
                                "status": "new",
                                "value": "2020-11-11"
                        },
                        "dateOfExpiry": {
                                "status": "new",
                                "value": "2027-11-12"
                        },
                        "authority": {
                                "status": "new",
                                "value": "Sas"
                        },
                        "placeOfIssue": {},
                        "countryOfDocument": {
                                "status": "new",
                                "value": "DEU"
                        },
                        "records": [
                                {
                                        "recordId": "935254843",
                                        "fileName": "K6JNXGBG2XVU_idimage_1.jpg",
                                        "belongsTo": "identificationdocument",
                                        "type": "idimage",
                                        "mimeType": "image/jpeg"
                                }
                        ]
                },
                "records": [
                        {
                                "recordId": "935254842",
                                "fileName": "K6JNXGBG2XVU_usersignature.jpg",
                                "belongsTo": "method",
                                "type": "usersignature",
                                "mimeType": "image/jpeg"
                        }
                ],
                "additionalDataBasic": {
                        "couponDownloadCount": 1,
                        "couponDownloadLastTimestamp": "2023-03-09T14:02:20+01:00",
                        "postOfficeStreetAddress": "Platz der Deutschen Post",
                        "postOfficeCity": "Bonn"
                }
        }
    ],
    "accountingData": {
            "accountingNumber": "506606638437A2",
            "accountingProduct": "Postident Basic Zusatzprodukt 2"
    }
  }
]
```


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

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

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

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

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

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

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

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.

Import the provided Eclipse Project

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

Java SCR Sample JUnit Tests

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

- generate RSA Keypairs
- calculate HMAC Hash on RSA Public Keys
- process scr requests
- decrypt SCR results

java source

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

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

provided classes:

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

public class ScrCaller

pure HTTP request handling

ScrCaller

```
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

cryptographic methods required for scr call handling

© 2022 Deutsche Post AG

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);
```

© 2022 Deutsche Post AG

```
        } else {
            System.out.println(MessageFormat.format(message, args));
        }
    }
}
```

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

```
package de.deutschepost.postident.demo.scr;

import static org.assertj.core.api.Assertions.assertThatNoException;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.KeyPair;
import java.security.NoSuchAlgorithmException;
import java.text.MessageFormat;
import java.text.ParseException;

import org.apache.tomcat.util.codec.binary.Base64;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

import com.nimbusds.jose.JOSEException;

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

© 2022 Deutsche Post AG

```
	/* Base64 representation of the public key of the above key pair */
	static String pubkey_base64;
	/* Base64 representation of the private key of the key pair above */
	static String privkey_base64;
	/*
	 * the HMAC hash of the public key (calculated with the scr_datapassword as hash
	 * secret)
	 */
	static String hmac_hash;

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

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

	/**
	 * Processing of an SCR get request (unencrypted answer)
	 *
	 * The result is unencrypted because the headers x-scr-key and x-scr-keyhash are
	 * not be set. Note: The production environment suppresses unencrypted result
	 * querys
	 */
	@Test
	void testGetCasesUnencrypted() {
		out("JUNIT Test testGetCasesUnencrypted");
		String ret = ScrHttpRequestHandler.callScr(scr_url_full_all, scr_user,
scr_password);
		out(ret);
		assertThatNoException();
		assertTrue(ret.startsWith("["));
	}

	/**
	 * Processing of an SCR get request (encrypted response without decryption)
	 *
	 * The result is encrypted because the headers x-scr-key and x-scr-keyhash are
	 * set. no decryption takes place in this test, the first ones Characters of the
	 * encrypted response are output on the console
	 */
	@Test
	void testGetCasesEncrypted() throws ParseException, JOSEException, ScrException {
		out("JUNIT Test testGetCasesEncrypted");
		String ret = ScrHttpRequestHandler.callScr(scr_url_full_all, scr_user,
```

© 2022 Deutsche Post AG

```
scr_password, pubkey_base64, hmac_hash);
                out(ret.substring(0, 80) + " ...");
                assertThatNoException();
                assertTrue(ret.startsWith("eyJlbmMi"));// base64 representation of '{"enc":"'
        }

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

        /**
         * 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));
                }
        }

}
```

Java Snipplets

RSA Java Snipplet to Decrypt the JWE Response

© 2022 Deutsche Post AG

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

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

PHP Client Sample

```
/*
SCR PHP client sample code

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

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

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

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

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


$headers = [
    'Content-type: application/JSON; CHARSET=UTF-8',
    'Authorization: Basic '. base64_encode("$user_name:$password"),
```

```
    'x-scr-enc: A256GCM', // PHP Seclib3 only supports GCM
    'x-scr-alg: RSA-OAEP-256',
    'x-scr-key: '.exctractKeyFromCert($public_key),
    'x-scr-keyhash: '.base64_encode(hash_hmac('sha256', base64_decode(exctractKeyFromCert
($public_key)), $data_password, TRUE))
];
var_dump('headers ', $headers );

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

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

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

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

    $keyAlg = getJweHeaderKeyAlg($jweHeaderBase64);
    $encryptionAlg = getJweHeaderEncryptionAlg($jweHeaderBase64);
    $aesDecryptionKey = "";
    switch ($keyAlg) {
        case 'RSA-OAEP-256':
        {
            $rsa = PublicKeyLoader::load($rsaPrivateKey)
                ->withHash('sha256')
                ->withMGFHash('sha256');
            $aesDecryptionKey = $rsa->decrypt(decodeBase64Url($jweEncryptedKeyBase64));
            break;
        }
        default:
        {
            // Alg nicht unterstützt
            return '*** Alg nicht unterstützt ***';
        }
    }
    switch ($encryptionAlg) {
        case 'A128GCM' || 'A192GCM' || 'A256GCM':
        {
            $nonce = decodeBase64Url($nonceBase64);
            $ciphertext = decodeBase64Url($encryptedDataBase64);
            $gcmTag = decodeBase64Url($gcmTagBase64);
            $cipher = new AES('gcm');
            $cipher->setKey($aesDecryptionKey);
            $cipher->setNonce($nonce);
            $cipher->setAAD($jweHeaderBase64);
            $cipher->setTag($gcmTag);
            return $cipher->decrypt($ciphertext);
        }
        default:
```

© 2022 Deutsche Post AG

```
            {
                // Alg nicht unterstützt
                return '*** Alg nicht unterstützt ***';
            }
     }
}


function getJweHeaderKeyAlg($jweHeaderBase64)
{
    $array = json_decode(decodeBase64Url($jweHeaderBase64), true);
    return $array['alg'];
}

function getJweHeaderEncryptionAlg($jweHeaderBase64)
{
    $array = json_decode(decodeBase64Url($jweHeaderBase64), true);
    return $array['enc'];
}


function encodeBase64Url($data)
{
    return rtrim(strtr(base64_encode($data), '+/', '-_'), '=');
}


function decodeBase64Url($data)
{
    return base64_decode(str_pad(strtr($data, '-_', '+/'), strlen($data) % 4, '=', STR_PAD_RIGHT));
}


?>
```

Python Client Sample

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

The python modules cryptography and jwcrypto are required.

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

SCR Python Client

```
'''

Postident SCR Python Reference Client

Created on 26.06.2022

@author: @author Deutsche Post AG

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


run pip install cryptography
run pip install jwcrypto
```

© 2022 Deutsche Post AG

```
'''

import base64
import http.client

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


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


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


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

rsa_private_key_loaded3k_base64=\
    "MIIG4gIBAAKCAYEApmc13IbTFltQJyH7CRipWfK4ibMikIxeDVO1XjpndbBNHDfQk0C+jZLnzNGpy3BvaRr1TqWbV"\
    "06paOazw6v2iocaNEGYA+6bxwf7rKhrmAYzce91E8bzIgbmyLAHU/d0yPI3lcxqL01QxaF6gi+uSA88Q5loSK7VCp"\
    "SObb9sXG02iJL4Op0An9s1n6/8NXxyu41tA/GxMTmbSWjH8VlRcO9xr7/ja88aIHIMFjgG9+yr/IPyr6i0kWtOciF"\
    "vT+scEseVTxVU7adSOrQPW0OD/yjspIqwHr9WWqv0DGY8/ha+mY2A0zaYUWYoWS7MnfFaBsDHl01z4vwdl0BwE3Y2"\
    "lCkrIK6IohkV2RqB5pB7nUrUAzEiDy23Cv31DdF1fJ0R2mW/GeTQW6oqL7n5pRUsdmsiwVy7pGGlSTUJ6qmZgsQsN"\
    "vYkxWrkbn+C6qSK+xjAs3eqJsSrmIvb0PLYVMWmXXbVvRvonm4KgumCpJGZ4f+1KeI92MSMKekxdy4ACLhBAgMBAA"\
    "ECggGAJYW8dX/JdY8dodVdwA1ZkrjSt+o27xVX1/V6Pkc5MSaSHLzRQYMl9NKdhmf4u7EczjyP+C4Lu0I3nTn4azU"\
    "JoO8aD5KC7tANPHImZCaOLHepfhWa9tyz0oZv3+0hmeOA6BcGRbcUcpoijo39DAIocINO+YdMJqeVzo71FvZFPwo5"\
    "c86APudMTr+ldd6/JmhrjrMzU+JaLFlrFMCV5WxYmvjDT/gbFiJbJ9nPs+rqV1l85rHyNX37SZdC1yWG7iR7kO6wg"\
    "lMii9oI1KM2o40QsTMI+zyxDushE3w6lZPPxxNaawqOJdpJfzsEhRGJz0IS2UZZ50t02x1zImh0o5uVwXyHeMMnNC"\
    "vowFo+BvTeAbFlDn5cXpEWx+mX7i2Q8qPLT5/rvYxjSjRvAUhhErvuF15/TFzhXDe/J0gFuXVzktAJeCOVfgrkk9Q"\
    "vJ9HkzF/q7WltAKE0EHy5bB1J1e0d0fnDg5hJvsQCsT+89LbNFQ35MeyPX54mvyY3oWw8I7XpAoHBANczIViDxLmB"\
    "ej2OGmVTG+1Cl7h7xyeQDetWwARYBx4G+e6jxAXAzvRRXqSPNhWv7b3lVLUszWU5W4cAMEPlhhRceyhkXfkEDBDiG"\
    "CVEORjUCoJ7E+l6AAB+gSbUbpUqjRBlATinnwmnuY/2X+0OFUEl30nsXkaFhr2WfQLLehuTa6GUyAK0ieYmEMDcl0"\
    "C+C5E5swMuSpmuszleuGxgwaqY29JpDSBvc3FxnMYyaMVMi/VxeEsHvCJ6+wHyB9pvTwKBwQDF87RsBmjUIijOeq3"\
    "aigXptwljjuwXKOh9POpF9hLDbKDn8U3tkcq5IICri4wf506AD/OrQlu/7fUZgjKBbzKchCgQxg/JbB9cskHAsYOA"\
    "+aHne0fRI+kNeI+Wpx6q8UFGHlcp0gSMHaabjEO1PRbxbF4PUwY2csO/kW/P30hihRxSxbQt3bNsB8Se75xzJOoMy"\
    "j2BF/pYM9g7uDEs1psyBoVg4UKLybHlqxEQgkq306eYd9EoMrT7h7u/afPQ+28CgcAGJMiL7V8daKvju/3W7LN8Z4"\
    "1LUAVUhNFQ6a4bsaOqYMqb3IMJIwMFiJkqG4iQv/AKntR3Q6ste6C4TvIRziiwxh8h/RONu2bYyIul7LewlMUkCn1"\
    "8CeacHQB06lWp3ogecrPBOU/aB7bNfFwquV8z9+/S/XOHkeJR4Uz6WnLG/MNy3nuDUEIrluSdjO6og4RzBfGtYpHw"\
    "p0MonSKovW5p/2kvLZ6ZUXU7ROIT6naBQ3kvbajgg30EgvxNhupgexMCgcBEHITaHqJ3S4Gq+j9T0OYT8u0CGq1Vk"\
    "rfcT9d0VyVBpOEAreDYe6+7X1mmrQ978sTp/5MiPGUBd0k8i7YBH1fUzwxSKTRJZDCuXo+NVAchm8N/uMWPSsHn5r"\
    "HRbpN40iZZkBsBwsfZxmILZ4nslKaOT3FV6IVcusRr6AkHB5cKfy6ttGU42u3foBShc2Troan+2J+tCakULFKcydX"\
    "xza382o2ONjQFkVLq6Z+nhI1dDzC9n4ySPlBTs/J//F7ua3UCgcA/ipe/YMbP8+N79+w0tWDJWMnwJ+tJ0tG9F3Bj"\
    "Mh2Se4KDQZbhgacReEo3jb6WQ2NxkELsl3ciozZ996VwHTQ6LJHoRJZeN/3ugUQkxItsMetHt23EbU1BFF6CPtel4"\
    "MoVi85FxF/mhGvvy/Bc7ri1BaaKVHMriRpxqVJ+diXdfwK/zLe+dwLhBd70vP38YhtTdbJ2dPsm2gW2h+0+g04NE4"\
    "prWDKhjNr6EBmjw/3kQEhXF4MJ3FT5j13W0KKx3xQ="

rsa_private_key_loaded_bytes = base64.b64decode(rsa_private_key_loaded_base64)
rsa_private_key_loaded = serialization.load_der_private_key(
    data=rsa_private_key_loaded_bytes ,
    password=None
    )

''' OPTION:
to calculate rsa keypair use this
rsa_private_key_generated = rsa.generate_private_key(
    public_exponent=65537,
```

```
    key_size=3096,
)
'''
# work with loaded private key - it will be more secure often to change the private key
rsa_private_key = rsa_private_key_loaded  # alternatively: rsa_private_key_generated
rsa_public_key = rsa_private_key.public_key()


# bring the private key in pem form
rsa_private_key_pem = rsa_private_key.private_bytes(
   encoding=serialization.Encoding.PEM,
   format=serialization.PrivateFormat.TraditionalOpenSSL,
   encryption_algorithm=serialization.NoEncryption()
)

publicKeyDerBytes = rsa_public_key.public_bytes( encoding=serialization.Encoding.DER, format =
serialization.PublicFormat.SubjectPublicKeyInfo)
public_key_der_base64 = base64.b64encode( publicKeyDerBytes ).decode("ascii")

print( "p4.3 get public  key in byte[] form " , publicKeyDerBytes.hex())
print( "p4.4 get public  key in Base64 String Form " , public_key_der_base64)
print( "p4.5 get private key in byte[] form " , rsa_private_key_pem.hex())
print( "p4.6 get private key  " , rsa_private_key_pem)



# ALL STEPS
'''
1.1 convert datapassword to byte[]
'''
dataPasswordAsBytes = str.encode(DATAPASSWORD_STR, "utf-8")
datapassword_as_hexstring = dataPasswordAsBytes.hex()
print( "1.1 dataPasswordBytes                " , DATAPASSWORD_STR, " ", datapassword_as_hexstring)


'''
1.2 convert RSA public key to byte[] and base64
'''
print( "1.2 public key in DER encoding in byte[] form" , publicKeyDerBytes.hex())
print( "1.2 public key in DER encoding in base64" , public_key_der_base64  )
'''
1.3 create and initialize javax.crypto.Mac

SecretKeySpec signingKey = new SecretKeySpec(dataPasswordBytes, "HmacSHA256");
Mac mac = Mac.getInstance("HmacSHA256");
mac.init(signingKey);
'''
hmacHashser = hmac.HMAC(dataPasswordAsBytes, hashes.SHA256(), backend=default_backend())

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

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

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


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

© 2022 Deutsche Post AG

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

"""2.2    convert the result of step 2.1 into base64 form    userAndPass = Base64.getEncoder().
encodeToString(userAndPass.getBytes("UTF-8"));    userAndPass =
            U0NSREVNTzozciM0TXUjR0JSbVA=
"""
userAndPass = base64.b64encode(str.encode(userAndPass, "utf-8")).decode("ascii") #decode
transforms bytearray to string
print( "2.2 userAndPass step 2 base64" , userAndPass  )

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


'''3.1

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

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


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

http_connection = http.client.HTTPSConnection(POSTIDENT_HOSTNAME_STR)

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



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

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

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

payload = ""

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

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

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


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

    6.1 parse encrypted payload into JWEObject (implicit in 6.4)
    6.2 instantiate RSAPrivateKey '''
jwk_private_key = jwk.JWK.from_pem(rsa_private_key_pem)
''' 6.3 instantiate Decrypter '''
jwe_decrypt_token = jwe.JWE()
''' 6.4. decrypt JWEObject'''
jwe_decrypt_token.deserialize(encrypted_response.decode("utf-8"), jwk_private_key)
''' 6.5. get decrypted Response'''
payload = jwe_decrypt_token.payload
print(payload.decode('utf-8'))
```

© 2022 Deutsche Post AG