Public key update services for MOT test devices

Written by: C.S.R.P.A.D.

Description: Web-service manual

Document: 2010-02-25_MU-001_EN

Rev 001 del 25/02/2010

Rev 002 del 30/06/2010

Rev 003 del 18/06/2015

Table of contents

Purpose and scope

The C.S.R.P.A.D. the website provides a service for downloading public keys for devices used in the vehicle MOT process. The service is accessible only to manufacturers who have previously registered with the website through the appropriate procedure.

Description of the service

The service is accessible via Web Services (in SOAP format).

This provides:

- a list of all valid keys (for all devices)
- a list of all valid keys (for all devices) with the latest amendment date within a time interval
- a list of all valid keys (for all devices) with the latest amendment date prior to a given date
- a list of valid keys for one type approval
- a list of all valid keys for one type approval with the latest amendment date within a time interval
- a list of all valid keys for one type approval with the latest amendment date prior to a given date
- a list of valid keys relating to a given type of encryption
- a list of all valid keys for a given type of encryption with the latest amendment date within a time interval
- a list of all valid keys for a given type of encryption with the latest amendment date prior to a given date
- a list of all keys (including inactive or expired keys)
- a list of all keys (including inactive or expired keys) with the latest amendment date within a time interval

- a list of all keys (including inactive or expired keys) with the latest amendment date prior to a given date
- a list of inactive keys
- a list of inactive keys relating to a given type of encryption
- an individual key

The service also provides additional functions for partial downloading of keys, to allow page layout for example.

Type of key encryption

Each type approval may contain keys with RSA 1024 or RC4 encryption.

Both or only one of these formats may be present for one type approval.

Key expiry dates

When a manufacturer adds a new key for a type approval, the previous key entered for the same type approval with the same type of encryption begins a validity period lasting 30 days.

For this reason, when valid keys are required for a specific device, more than one may be provided. The service in fact provides the latest key added by the manufacturer, in addition to all other keys that have not yet expired. Keys with inactive type approval are excluded.

Other methods are available for obtaining a record of keys for a given device, even if the associated type approval has been deactivated. These methods may be used for the purposes of testing the validity of a test conducted in the past.

Description of keys

Data relating to an individual key are embedded in the key object, which is returned singly or contained in a list (array) returned by the methods used for downloading the keys.

The key object displays the following properties:

1. *key*: the key (in text format)
2. *code*: code (in 5-figure numerical format)
3. *encryptionType*: type of encryption
4. *linkType*: type of link
5. *approvalNumber*: device type approval number
6. *creationDate*: date of creation
7. *expirationDate*: date of expiration
8. *inhibitionDate*: date of inhibition
9. *updateDate*: date of latest update

Detail of fields

Key text (key)
Contains the key.

Format depends on the encryption type:

- RSA 1024 keys are in xml format
- RC4 are in 8-character hexadecimal text format.

Xml example of RSA 1024 key:

```
<RSAKeyValue>
    <Modulus>
```

yPIZrZHeMPqVoRnclEwSUdRFkO2k4eEvjBfi6y9aVVpfzgJ34fRL6txizafmrKu8L
4rAGklNX95ueMTGmgR84uo1F4sWHtgL4SIb2EE/SJWKMQeSKcEbHfCu+a17Xma40e
Td23Ld/LoV5uhs5FSRlVBDlld1APZllcDdAeUKu/k=
</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

Code (code)
Identifies a 5-figure numerical string representing the progressive number allocated by the system to the key.

Encryption type (encryptionType)
Identifies a text identifying the encryption type used for the key. Permitted values are RSA or RC4.

Type of link (linkType)
Identifies a text identifying the type of link used for the key. Permitted values are DIR, RETE, RSSE, RSCE.

Device type approval number (approvalNumber)
Identifies a text (maximum 50 characters) identifying the type approval with which the keys are linked.

Date of creation (creationDate)
Identifies the key creation date

Date of expiration (expirationDate)
Identifies the key expiration date. This coincides with the input date (postponed by 30 days) of a new key for the same type of encryption.

Date of inhibition (inhibitionDate)
Identifies the date on which type approval was deactivated (inhibited). An inhibited key should be considered no longer valid to all intents and purposes as of the inhibition date.

Date of latest update (updateDate)
Identifies the date on which key status was last updated. For example, a key changes status when a new key is linked to the same type approval, or also when the key itself is inhibited.

Authentication service specifications

Introductory notes

In order to gain access to public key download services, users must first authenticate themselves in the system. For this purpose, the appropriate authentication service methods must be followed by providing the username and password for the relevant manufacturer/user.

Service access

The authentication service is published at the following address:

https://www.csrpad.it/csrpad-csrpad/AuthenticatorService?wsdl

Methods implemented

boolean loginPlain(String username, String password)

Allows an authenticated working session to be set up, by means of username and password. Returns *true* if successful (authentication carried out).

public String login(String username, String password)

Allows an authenticated working session to be set up, by means of username and password. Also activates management of conversation (described below), returning *conversationId* if successful (authentication carried out), otherwise an empty string.

boolean logout()

Allows an authenticated working session to be closed, terminating any open conversation. Returns a boolean logout.

Both *login* methods set a session ID (*JSESSIONID*) in the SOAP response header. To use the authenticated session in subsequent calls to the service, the session ID must be set in SOAP request headers.

*Example of response generated by the method of **login:***

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
Set-Cookie: JSESSIONID=FF013E1775EC01CD786E0D4AC3C4C35A; Path=/csrpad-csrpad
Content-Type: text/xml;charset=UTF-8
Transfer-Encoding: chunked
Date: Wed, 17 Feb 2010 14:40:54 GMT

<env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/envelope/'>
  <env:Header>
    <seam:conversationId xmlns:seam='http://www.jboss.org/seam/webservice'>
    4
    </seam:conversationId>
  </env:Header>
  <env:Body>
    <ws:loginResponse xmlns:ws='http://www.csrpad.it/ws'>
      <return>4</return>
    </ws:loginResponse>
  </env:Body>
</env:Envelope>
```

Conversation

The conversation is a context, defined within a working session, that allows the status assumed by objects used to be stored from one call to the next.

The conversation may be opened and closed without this involving a log-out from the authenticated session.

The starting new conversation, it is necessary to invoke the *login* method. subsequently, to use it and maintain it active, set *conversationId* in the SOAP request header.

The *conversationId* may be obtained as a value returned by the *login* method, or extracted from the SOAP response.

Examples of using conversationId:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:seam="http://seambay.example.seam.jboss.org/">
  <soapenv:Header>
    <seam:conversationId xmlns:seam='http://www.jboss.org/seam/webservice'>
      2
    </seam:conversationId>
  </soapenv:Header>
  <soapenv:Body>
    <seam:findValidKeys/>
  </soapenv:Body>
</soapenv:Envelope>
```

Key download service specifications

Introductory notes

The term ***ValidKeys*** in the methods indicates that the keys returned are active and have not expired.

The term ***Updated*** in the methods indicates that the keys returned are all those updated from date **startUpdateDate** to date **endUpdateDate.**

The term ***UpdatedUntil*** in the methods indicates that the keys returned are all those amended prior to the date **endUpdateDate.**

The three items ***approvalNumber, creationDate, code*** represent a unique identifier for the key.

In order to allow page layout of the results, all methods returning a list of keys have corresponding methods that return only a portion of the list and a method that returns the total number of results.

Service access

The service for managing key download is published at the address:

https://www.csrpad.it/csrpad-csrpad/KeyService?wsdl

Methods implemented

Key[] findValidKeys()

Returns an array of key objects containing keys linked with all type approvals currently registered in the C.S.R.P.A.D. website.

Key[] findUpdatedValidKeys(Date startUpdateDate, Date endUpdateDate)

Returns an array of key objects containing keys linked with all type approvals currently registered in the C.S.R.P.A.D. website, filtered by *updateDate* (latest update date of key included in the interval between startUpdateDate and endUpdateDate inclusive).

Key[] findUpdatedUntilValidKeys(Date endUpdateDate)

Returns an array of key objects containing keys linked with all type approvals currently registered in the C.S.R.P.A.D. website, filtered by *updateDate* (latest update date of key equal to or prior to endUpdateDate).

Key[] findValidKeysByApprovalNumber(String approvalNumber)

Returns an array of key objects, filtered by*approvalNumber* (type approval number with which the keys are linked).

Key[] findUpdatedValidKeysByApprovalNumber(String approvalNumber, Date startUpdateDate, Date endUpdateDate)

Returns an array of key objects, filtered by*approvalNumber* (type approval number with which the keys are linked) and *updateDate* (latest key update date included in the interval between startUpdateDate and endUpdateDate inclusive).

Key[] findUpdatedUntilValidKeysByApprovalNumber(String approvalNumber, Date endUpdateDate)

Returns an array of key objects, filtered by*approvalNumber* (type approval number with which the keys are linked) and *updateDate* (latest key update date equal to or prior to endUpdateDate).

Key[] findValidKeysByType(String encryptionType)

Returns an array of key objects, filtered by*encryptionType* (encryption type, permitted values being RSA or RC4).

Key[] findUpdatedValidKeysByType(String encryptionType, Date startUpdateDate, Date endUpdateDate)

Returns an array of key objects, filtered by*encryptionType* (encryption type, permitted values being RSA or RC4) and *updateDate* (latest key update date included in the interval between startUpdateDate and endUpdateDate inclusive).

Key[] findUpdatedUntilValidKeysByType(String encryptionType, Date endUpdateDate)

Returns an array of key objects, filtered by*encryptionType* (encryption type, permitted values being RSA or RC4) and *updateDate* (latest key update date equal to or prior to endUpdateDate).

Key[] findKeys()

Returns an array of key objects containing all keys, even if inhibited or expired, currently registered in the C.S.R.P.A.D. website.

Key[] findUpdatedKeys(Date startUpdateDate, Date endUpdateDate)

Returns an array of key objects containing all keys, even if inhibited or expired, currently registered in the C.S.R.P.A.D. website, filtered by *updateDate* (latest update date of key included in the interval between startUpdateDate and endUpdateDate inclusive).

Key[] findUpdatedUntilKeys(Date endUpdateDate)

Returns an array of key objects containing all keys, even if inhibited or expired, currently registered in the C.S.R.P.A.D. website, filtered by *updateDate* (latest update date of key equal to or prior to endUpdateDate).

Key[] findInhibitedKeys(Date inhibitionDate)

Returns an array of key objects, containing keys for which the inhibition date is equal to or later than *inhibitionDate*.

Key[] findInhibitedKeysByType (Date inhibitionDate, String encryptionType)

Returns an array of key objects containing keys for which the inhibition date is equal to or later than *inhibitionDate* and whose type corresponds to *encryptionType* (encryption type, for which permitted values are RSA or RC4).

Key findKeyById(String approvalNumber, Date creationDate, String code)

Returns a single key, searching by key ID (see introductory notes).

Date getSynchroDate()

Returns the current server date.

Examples

Authentication

Examples of authentication with login, login plain and logout.

PseudoCodice

```
String conversationId = AuthenticatorService.login("wsuser@csrpad.it", "xxxxxxxx");
Print(conversationId);
```

Response

```
4
```

PseudoCodice

```
boolean loggedIn = AuthenticatorService.loginPlain("wsuser@csrpad.it", "xxxxxxxx");
if(loggedIn){
	Print(“true”);
}
```

Response

```
true
```

PseudoCodice

```
boolean loggedOut = AuthenticatorService.logout();
if(loggedOut){
	Print(“true”);
}
```

Response

```
true
```

Synchronisation date

Example of server data reception.

PseudoCodice

```
Date now = KeyService.getSynchroDate();
Print(now);
```

Response

```
Thu Mar 11 10:05:17 CET 2010
```

Downloading keys for a type approval

Example of downloading keys for a specific type approval, showing the relevant details.

PseudoCodice

```
Key[] keys = KeyService.findValidKeysByApprovalNumber("OMTESTRSA0001");

for (Key k : keys) {
        Print(k.getKey());
        Print(k.getCode());
        Print(k.getLinkType());
        Print(k.getEncryptionType());
        Print(k.getApprovalNumber());
        Print(k.getCreationDate());
        Print(k.getExpirationDate());
        if (k.getExpirationDate() > getCurrentDate()) {
                Print("Key Expired");
        }
        Print(k.getCreationDate());
        if (k.getInhibitionDate() != null) {
                Print("Key inhibited");
        }
        Print(k.getUpdateDate());
}
```

Response

<RSAKeyValue><Modulus>yPIZrZHeMPqVoRnclEwSUdRFkO2k4eEvjBfi6y9aVVpfzgJ34fRL6txizafmrKu8L4rAGklNX95ueMTGmgR84uo1F4sWHtgL4SIb2EE/SJWKMQeSKcEbHfCu+a17Xma40eTd23Ld/LoV5uhs5FSRlVBDlld1APZllcDdAeUKu/k=</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>
00001
RSCE
RSA
om1234T
Mon Feb 08 00:00:00 CET 2010
Fri Mar 12 00:00:00 CET 2010
null
Thu Feb 11 09:46:32 CET 2010

1287abcd
00002
RSCE
RC4
om1234T
Mon Feb 08 00:00:00 CET 2010
Fri Mar 12 00:00:00 CET 2010
null
Thu Feb 11 09:57:53 CET 2010

Downloading inhibited keys

Example of downloading keys for which the inhibition date is equal to or later than 02-07-2010, showing relevant detail.

PseudoCodice

```
Key[] keys = KeyService.findInhibitedKeys(createDate(7, 2, 2010));

for (Key k : keys) {
        Print(k.getKey());
        Print(k.getCode());
        Print(k.getLinkType());
        Print(k.getEncryptionType());
        Print(k.getApprovalNumber());
        Print(k.getCreationDate());
        if (k.getExpirationDate() > getCurrentDate()) {
                Print("Key Expired");
        }
        Print(k.getCreationDate());
        if (k.getInhibitionDate() != null) {
                Print("Key inhibited");
        }
        Print(k.getUpdateDate());
}
```

Response

<RSAKeyValue><Modulus>yPIZrZHeMPqVoRnclEwSUdRFkO2k4eEvjBfi6y9aVVpfzgJ34fRL6txizafmrKu8L4rAGklNX95ueMTGmgR84uo1F4sWHtgL4SIb2EE/SJWKMQeSKcEbHfCu+a17Xma40eTd23Ld/LoV5uhs5FSRlVBDlld1APZllcDdAeUKu/k=</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>
00003
RSCE
RSA
om1234T
Thu Feb 04 00:00:00 CET 2010
Fri Mar 05 00:00:00 CET 2010
Mon Feb 08 00:00:00 CET 2010
Key inhibited
Mon Feb 08 18:00:00 CET 2010

1245abcd
00004
RSSE
RC4
om1234T
Thu Feb 04 00:00:00 CET 2010
Fri Mar 05 00:00:00 CET 2010
Mon Feb 08 00:00:00 CET 2010
Key inhibited
Mon Feb 08 18:00:00 CET 2010

Downloading of valid keys with filter on latest update date

The following example shows the downloading of all valid keys using the latest update date as a filter. The first time it is run, all currently valid keys will be downloaded while only keys that have been updated since the last download date will subsequently be downloaded.

To download keys in paginated mode (this mode is advisable to avoid any timeouts when receiving the response from Web services), see example below.

PseudoCodice

```
function Date getLastUpdateDate(){
	// Return key last download date (stored locally in database, configuration file or on similar
system), void if first download
}

function Date setLastUpdateDate(Date data){
	// Save key last download date (stored locally in database, configuration file or on similar
system)
}

function void updateKeys(Key[] keys){
	// Update downloaded keys (stored locally in database, configuration file or on similar system)
}


function void main(){
	boolean loggedIn = AuthenticatorService.loginPlain("wsuser@csrpad.it", "xxxxxxxx");
	if(loggedIn){
		Date startUpdateDate = getLastUpdateDate();
		Date endUpdateDate = KeyService.getSynchroDate();

		if (startUpdateDate == null){
			Key[] keys = KeyService.findUpdatedUntilValidKeys(endUpdateDate);
			updateKeys(keys);
			setLastUpdateDate(endUpdateDate);
		}
		else {
			Key[] keys = KeyService.findUpdatedValidKeys(startUpdateDate, endUpdateDate);
			updateKeys(keys);
			setLastUpdateDate(endUpdateDate);
		}

		AuthenticatorService.logout();
	}

}
```

Downloading paginated valid keys with filter on latest update date

The following example shows the downloading of all valid keys using the latest update date as a filter, managing the download in paginated mode. The first time it is run, all currently valid keys will be downloaded while only keys that have been updated since the last download date will subsequently be downloaded.

Each individual download will take place in paginated mode: In the first instance, the number of keys to be downloaded will be counted and all the necessary keys will be downloaded in blocks on the basis of this number.

It is important to note that if a key is updated during paginated download, the synchronisation result may not be consistent: it is therefore essential to check whether the number of keys to be downloaded is the same as those effectively downloaded and if not, it will be necessary to carry out a fresh synchronisation.

Paginated mode is recommendable, particularly to avoid timeouts in receiving a response from Web services

PseudoCodice

```
function Date getLastUpdateDate(){
	// Return key last download date (stored locally in database, configuration file or on similar
system), void if first download
}

function Date setLastUpdateDate(Date data){
	// Save key last download date (stored locally in database, configuration file or on similar
system)
}

function void updateKeys(Key[] keys){
	// Update downloaded keys (stored locally in database, configuration file or on similar system)
}

function key[] downloadUpdatedUntilValidKeysPaginated(startCounter,endUpdateDate){
	List listKeys = new List();

	long endCounter = 0;
	long pageSize = 100;
	while (startCounter != endCounter){

		startCounter = KeyService.countUpdatedUntilValidKeys(endUpdateDate);
		long numPage = (startCounter / pageSize) +1;
		if (startCounter % pageSize == 0) numPage--;

		for (int i=0; i< numPage ; i++){

	listKeys.addAll(KeyService.findUpdatedUntilValidKeysPaginated(endUpdateDate,i*pageSize,
pageSize));
		}

		endCounter = listKeys.size();
	}

	return listKeys.toArray();
}

function key[] downloadUpdatedValidKeysPaginated(startCounter,startUpdateDate,endUpdateDate){
	List listKeys = new List();

	long endCounter = 0;
	long pageSize = 100;
	while (startCounter != endCounter){

		startCounter = KeyService.countUpdatedValidKeys(startUpdateDate,endUpdateDate);
		long numPage = (startCounter / pageSize) +1;
		if (startCounter % pageSize == 0) numPage--;

		for (int i=0; i< numPage ; i++){
			listKeys.addAll(KeyService.findUpdatedValidKeysPaginated(startUpdateDate
,endUpdateDate,i*pageSize, pageSize));
		}

		endCounter = listKeys.size();

	}

	return listKeys.toArray();
}
```

```
function void main(){
	boolean loggedIn = AuthenticatorService.loginPlain("wsuser@csrpad.it", "xxxxxxxx");
	if(loggedIn){
		Date startUpdateDate = getLastUpdateDate();
		Date endUpdateDate = KeyService.getSynchroDate();

		if (startUpdateDate == null){

			long count = KeyService.countUpdatedUntilValidKeys(endUpdateDate);
			if (count > 0){
				Key[] keys = downloadUpdatedUntilValidKeysPaginated(count,endUpdateDate);
				updateKeys(keys);
				setLastUpdateDate(endUpdateDate);
			}
		}
		else {
			long count = KeyService.countUpdatedValidKeys(startUpdateDate,endUpdateDate);
			if (count > 0){
				Key[] keys = downloadUpdatedValidKeysPaginated(count,startUpdateDate,
	endUpdateDate);
				updateKeys(keys);
				setLastUpdateDate(endUpdateDate);
			}


		}

		AuthenticatorService.logout();
	}

}
```

Test account

The C.S.R.P.A.D. the website provides a service for downloading public keys for devices used in the vehicle MOT process. In order to verify the download of keys via web services, a test account has been set up:

USERNAME : utcsrpadws

PASSWORD: gf00g278

which makes it possible to use the service and download all the fictitious keys created exclusively for test purposes. As mentioned previously, in order to use the service and download the official public keys, it will be necessary to log onto the website using the procedure for manufacturers and use the account generated to log into the web services authentication service.

A list of fictitious keys may be downloaded from the following address:

http://www.csrpad.it/csrpad/pages/public/index.seam?page=chiavi_test

This list will be updated if further test keys are added, updated or removed.

For any technical problems, send an e-mail to the following address: *supporto@csrpad.it*