**KeyTalk - API**

|  |  |
| --- | --- |
| Author | MR vd Sman |
| Creation date | 14-March-2017 |
| Last updated | 17-December-2018 |
| Document version | 2.3.2 |
| Document status | Qualified |
| Product | KeyTalk certificate and key management & enrolment virtual appliance |
| Data classification | Public |

TABLE OF CONTENTS

1. InTROduction 2

1.1 Purpose 2

1.2 Scope 2

2. CERTIFICATE RETRIEVAL API (RCDP v2) 3

2.1 RCDPv2 versions 3

2.2 KeyTalk config file 3

2.3 RCDPv2 overview 4

2.4 RCDPv2 communication phases 5

2.5 Messages sent in all phases 6

2.5.1 End Of communication 6

2.5.2 Error 6

2.6 Phase 1 (handshake) 8

2.6.1 Hello 8

2.6.2 Handshake 8

2.7 Phase 2 (authentication) 10

2.7.1 Request authentication requirements 10

2.7.2 Authentication 12

2.7.3 Change password 18

2.8 Phase 3 (service provision) 19

2.8.1 Check for the last messages 19

2.8.2 Generate certificate on the server 20

2.8.3 *[as of v2.2.0]* Query CSR requirements 22

2.8.4 *[as of v2.2.0]* Generate certificate from the client CSR 23

3. PUBLIC API 25

3.1 Public API versions 25

3.2 API overview 25

3.2.1 Retrieve self-service availability 25

3.2.2 Retrieve address book URLs 26

3.2.3 *[as of v1.1.0]* Retrieve availability of S/MIME certificate enrollment to external parties for self-service 27

4. SELF-SERVICE API 28

4.1 API versions 28

4.2 API overview 28

4.2.1 Enroll S/MIME certificate for external parties 28

5. Certificate AUTHORITY RETRIEVAL API (CA API) 30

5.1 CA API versions 30

5.2 CA API overview 30

5.2.1 Retrieve internal signing CA 30

# InTROduction

## Purpose

The purpose of this document is to describe the API used by the KeyTalk system.

## Scope

This document is intended for KeyTalk and its hired 3rd parties for continuous development of the KeyTalk product and related services.

More importantly this document is intended for release to the public so they may use it for their own KeyTalk related development purposes.

# CERTIFICATE RETRIEVAL API (RCDP v2)

This section describes certificate retrieval API called RCDP version 2. The motivation to develop a new API over the existing legacy RCDPv1 was as follows:

* Offload handcrafted security to the standard SSL/TLS stack implemented by HTTPS protocol
* Use RESTful way of communication based on simple HTTP GET requests and JSON responses
* Simplify the API to make it easier to develop KeyTalk clients and related services

## RCDPv2 versions

|  |  |  |
| --- | --- | --- |
| **RCDP version** | **Supported KeyTalk server** | **Changes wrt the previous RCDP version** |
| 2.0.0 | 4.6.0 and up |  |
| 2.1.0 | 5.3.0 and up | Allow caller to request a certificate download URL in the phase 3 cert request instead of a certificate body. |
| 2.2.0 | 5.3.1 and up | - Allow submitting CSR for signing  - Include TPM Virtual Smart Card requirement flag as a part of auth-requirements response |
| 2.3.0 | 5.3.3 and up | - Allow for integrated Kerberos authentication  - Return “LOCKED <time>” when the user is still locked iso “DELAY <time>”  - Use HTTP POST iso HTTP GET for authentication and password change requests |

## KeyTalk config file

In order to make use of the KeyTalk API, several details are required from the KeyTalk Real Client Communication Data file (RCCD).

This configuration file is used to feed a KeyTalk app with minimal required information to setup a proper secure connection to any KeyTalk instance.

The RCCD file is effectively a zip container and can thus easily be extracted.

As such a developer incorporating the KeyTalk API in their app, can choose to statically make use of individual files in an RCCD file, or choose to import the entire RCCD into their app or simply make use of some of the components within this RCCD file.

The content folder within the RCCD contains several files, the most important ones being:

▪ **RCA.der Root CA** typically only included when KeyTak’s internal private CA is generated under an already existing CA.

▪ **PCA.der Primary CA** with a KeyTalk self-signed private CA its usually the top of the KeyTalk internal private CA, but when RCA is included its generated under the RCA.

▪ **UCA.der User CA** signed under the PCA. It is the trust under which the end-point client and/or server certificates are signed and issued only in case of using the internal KeyTalk CAfor issuance.

When issuing end-point client and/or server certificates under for example a connected Microsoft CA or Trusted Certificate Service Provider, ensure that their intermediate certificates are included in your app or present on the target OS as well as these are by default not part of the current KeyTalk RCCD

▪ **SCA.der Server CA** signed under the PCA, it is the trust under which the KeyTalk virtual appliance certificates are generated and used.

▪ **user.ini** Generic configuration settings which includes the KeyTalk server URL/IP as well as the KeyTalk tenant name/SERVICE used to communicate with.

▪ **user.yaml** Generic configuration settings which includes the KeyTalk server URL/IP as well as the KeyTalk tenant name/SERVICE used to communicate with. Similar to user.ini just another format

## RCDPv2 overview

Communication in RCDPv2 is encapsulated in RESTful calls over HTTPS using standard port 443. Optional out-of-band certificate downloads are made possible over HTTP using port 8000.

Below is a set of client HTTP headers that the client needs to send to the server.

|  |  |  |
| --- | --- | --- |
| **HTTP Header** | **Required** | **Description** |
| GET | YES | /rcdp/2.X.Y/<action>?<request-params> |
| Host | YES | Should contain the FQDN or IP (v4 or v6) of the KeyTalk virtual appliance. |
| Cookie | YES except for hello | Session identifier received from KeyTalk server. |

**action** is a request action

**request-params** is URL-encoded string of request parameters. Complex request parameters (arrays, dictionaries) should be JSON-encoded. All JSON objects should escape forward slashes ‘/’ as ‘\/’.

A typical set of client HTTP headers:

|  |
| --- |
| **GET** /rcdp/2.3.0/authentication?service=DEMO\_SERVICE&PASSWD=change%21&HWSIG=123456&USERID=DemoUser &ips=%5B%2281.175.103.107%22%5D&caller-hw-description=Windows+7%2C+BIOS+s%2Fn+1234567890 HTTP/1.1  **Host:** keytalkdemo.keytalk.com  **Cookie:** keytalkcookie=a622bb821bec1f5315668c8f9a8e780f |

A typical set of HTTP response headers:

|  |
| --- |
| **HTTP/1.1 200 OK**  **Content-type:** application/json  **Cache-Control:** no-cache  **Set-Cookie:** keytalkcookie=a622bb821bec1f5315668c8f9a8e780f  {'status': 'auth-result', 'auth-status': 'OK'} |

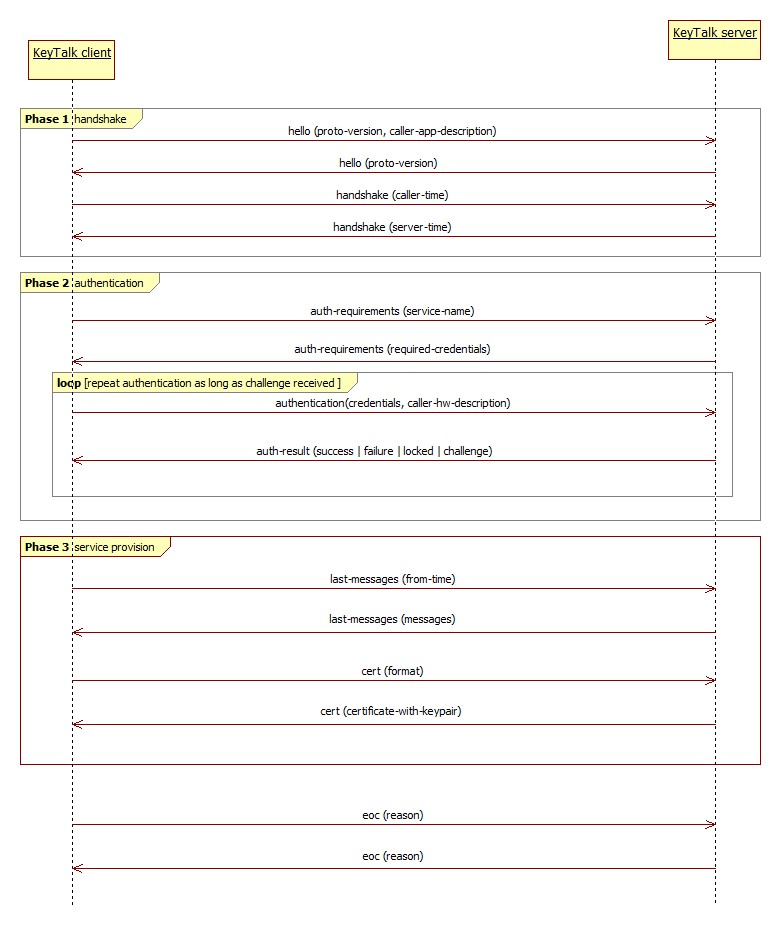
## RCDPv2 communication phases

The complete RCDPv2 communication circle consists of 3 phases:

**Phase1**: handshake

**Phase 2**: authentication

**Phase 3**: service provision



Further we describe message semantics on each phase in detail.

## Messages sent in all phases

### End Of communication

#### Request

GET /rcdp/<version>/eoc

**Example:**

/rcdp/2.3.0/eoc

/rcdp/2.3.0/eoc?reason=bye%2C+server

**Query parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **parameter** | **type** | **required** | **description** |
| reason | *string* | no | optional reason for ending communication |

**Response**

HTTP 200 – application/json

|  |
| --- |
| {  'status': 'eoc',  *[optional]* 'reason': optional reason for ending communication  } |

End of communication can be sent at any time, initiated by any communication side.

### Error

#### Errors are typically sent by the server to notify the caller on error processing its request. The client can also send errors to the server when it can’t handle the server’s response.

#### Request

GET /rcdp/<version>/error

**Example:**

/rcdp/2.3.0/error?code=1066&description=invalid+response

**Query parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **parameter** | **type** | **required** | **description** |
| code | *number* | yes | numeric error code |
| reason | *string* | no | optional error description. Might be required for certain error codes. See the error code table below. |

**Response**

HTTP 200 – application/json

|  |
| --- |
| {  'status': 'error',  'code': numeric error code,  *[optional]* 'description': error description. Might be required for certain error codes. See the error code table below.  } |

#### Error codes

|  |  |  |  |
| --- | --- | --- | --- |
| **code** | **description** | **direction** | **remarks** |
| 1001 (ErrResolvedIpInvalid) | optional | server -> client | Sent by the server when none of IPs resolved by the client and by the server match. |
| 1002 (ErrDigestInvalid) | optional | server -> client | Sent by the server when the client’s calculated executable digest does not much the digest stored on the server. |
| 1003 (ErrTimeOutOfSync) | difference in seconds between caller UTC and the server UTC | server -> client | Sent by the server when the client time is out of sync with the server’s time. |
| 1004 (ErrMaxLicensedUsersReached) | optional | server -> client | Sent by the server when no certificate can be supplied because the max number of licensed users has been reached |
| 1005 (ErrPasswordExpired) | optional | server -> client | Sent by the server when the password of the user trying to authenticate is expired and the caller is not supposed to change it. |

## Phase 1 (handshake)

### Hello

Agree on RCDP API version and establish session ID.

#### Request

GET /rcdp/<version>/hello

**Example:**

/rcdp/2.3.0/hello

/rcdp/2.3.0/hello?caller-app-description=Demo+KeyTalk+client

**Query parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **parameter** | **type** | **required** | **description** |
| caller-app-description | *string* | no | optional description of the caller application |

RCDP API version proposed by a caller is sent as a part HTTP GET path.

**Response**

HTTP 200 – application/json

|  |
| --- |
| {  “status”: “hello”,  “version”: proposed API version  } |

Session ID is returned in HTTP cookie keytalkcookie in Set-Cookie header.

### Handshake

Confirm version handshake and exchange time information.

#### Request

GET /rcdp/<version>/handshake

**Example:**

/rcdp/2.3.0/handshake?caller-utc=2016-04-22T10%3A44%3A35.746255Z

**Query parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **parameter** | **type** | **required** | **description** |
| caller-utc | *UTC string in ISO 8601 format including date and time* | yes | caller UTC |

If the caller supports API version proposed by the server on the previous step, it proceeds with this version in HTTP GET path. Otherwise the caller ends communication.

**Response**

HTTP 200 – application/json

|  |
| --- |
| {  “status”: “handshake”,  “server-utc”: server UTC in ISO 8601 format including date and time  } |

## Phase 2 (authentication)

### Request authentication requirements

Request authentication requirements from the server.

#### Request

GET /rcdp/<version>/auth-requirements

**Example:**

/rcdp/2.3.0/auth-requirements?service=DEMO\_SERVICE

**Query parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **parameter** | **type** | **required** | **description** |
| service | *string* | yes | KeyTalk service name |

**Response**

HTTP 200 – application/json

|  |
| --- |
| {  “status”: “auth-requirements”,  “credential-types”: credential types,  *[optional]* “hwsig\_formula”: HWSIG formula,  *[optional]* “password-prompt”: password-prompt,  *[optional]* “service-uris”: service URIs,  *[optional]* “resolve-service-uris”: if service URIs need to be resolved,  *[optional]* “calc-service-uris-digest”: if service URIs digest needs to be calculated,  ***[as of v2.2.0]****[optional]* “use-tpm-vsc-authentication”: if TPM Virtual Smart Card authentication should be used,  ***[as of v2.3.0]****[optional]* “use-kerberos-authentication”: if Kerberos authentication should be used,  } |

*credential-types*

JSON array of credential types required to authenticate against the given service. Supported credential types are: “USERID”, “HWSIG”, “PASSWD”, “PIN” and “RESPONSE”.

Example: [“USERID”, “HWSIG”, “PASSWD”]

*hwsig\_formula*

formula to calculate caller’s hardware signature.

Example: “1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16”. Sent when credential-types parameter contains HWSIG.

*password-prompt*

prompt to display to a user when a password is requested interactively e.g. “password” or “tokencode”. Sent when credential-types parameter contains PASSWD.

*service-uris*

JSON array of RFC 3986-compliant URIs of the given service

Example:

[“https://demo1.keytalk.com”, “https://demo2.keytalk.com”]

or

[“file://%ProgramFiles%\vpn\vpn.exe”]

*resolve-service-uris*

Boolean flag (“true” or “false”)requesting a caller to resolve IP addresses of each supplied service-uris identifying web resources. Defaults to “false”.

*calc-service-uris-digest*

Boolean flag (“true” or “false”)requesting a caller to calculate sha-256 hexadecimal digests of each supplied service-uris identifying file resources. Defaults to “false”.

*use-tpm-vcs-authentication*

Boolean flag (“true” or “false”)requesting a caller to make use of PM Virtual Smart Card to generate a certificate signing request (CSR). The CSR will be then sent KeyTalk server to create a certificate. Defaults to “false”.

use-kerberos-authentication

Boolean flag (“true” or “false”)requesting a caller to make use of Kerberos authentication. If Kerberos authentication happens to be not possible, the caller should fall back to regular KeyTalk authentication specified in *credential-types.*

**Example:**

|  |
| --- |
| {  “status”: “auth-requirements”,  “credential-types”: [“HWSIG”, “PASSWD”, “USERID”],  “hwsig\_formula”: “1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16”,  “password-prompt”: “Password”,  “service-uris”: [“https://demo.keytalk.com”],  “resolve-service-uris” : “true”  } |

### Authentication

Authenticate the caller against the selected service using the supplied set of credentials. Multiple authentication rounds might be needed e.g. for RADIUS SecurID or RADIUS EAP AKA/SIM authentication.

#### Request

***[as of v2.3.0]******HTTP POST is used for authentication***

POST /rcdp/<version>/authentication

Content-type: application/x-www-form-urlencoded

**Example:**

**$ curl** -H "Content-Type: application/x-www-form-urlencoded" \

-H "Cookie: keytalkcookie=a77c33e55a1f411396031ce91ee48d9d" \

–H "Expect:" \

-d "service=DEMO\_SERVICE&caller-hw-description=Windows+7%2C+BIOS+s%2Fn+1234567890&USERID=DemoUser&HWSIG=123456&PASSWD=change%21&resolved=%5B%7B%22ips%22%3A+%5B%2281.175.103.107%22%5D%2C+%22uri%22%3A+%22https%3A%2F%2Fdemo.keytalk.com%2F%22%7D%5D" \

-X POST <https://test.keytalk.com/rcdp/2.3.0/authentication>

**Query parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **parameter** | **type** | **required** | **description** |
| service | *string* | yes | KeyTalk service name |
| caller-hw-description | *string* | yes | Caller HW description which should be unique for the given device. For uniqueness e.g. BIOS serial number or iOS device UDID can be used. Examples:   * Windows 10, BIOS s/n 1234567890 * iPAD: Jan’s iPAD 234567890abcdef1234567890abcdef |
| USERID | *string* | if requested | ID of the user. Required if USERID was previously set by the server in auth-requirements response. |
| HWSIG | *string* | if requested | Hardware Signature of the caller’s device calculated with the formula specified in the previous auth-requirements server response. Required if HWSIG was previously set by the server in auth-requirements response.. |
| PASSWD | *string* | if requested | User password. Required if PASSWD was previously set by the server in auth-requirements response. |
| PIN | *string* | if requested | User pincode. Required if PIN was previously set by the server in auth-requirements response. |
| resolved | *JSON array* | if requested | JSON array of objects containing service URIs accompanied with [RFC 3986](http://tools.ietf.org/html/rfc3986)-compliant IPv4 or IPv6 address resolved from the URI hostname.  Required if resolve-service-uris was previously set in auth-requirements response.  Example:  [  {  "uri":"https://demo1.keytalk.com",  "ips":["81.175.10.107","81.175.103.109"]  },  {  "uri":"https://demo2.keytalk.com",  "ips":["81.175.10.108","[2001:db8:a0b:12f0::1]"]  }  ] |
| digests | *JSON array* | if requested | JSON array of objects containing service URIs accompanied with SHA-256 hexadecimal digest of the underlying file.  Required if calc-service-uris-digest was previously set in auth-requirements response.  Example:  [  {  "uri":"file://%Program Files%\vpn\vpn.exe",  "digest":"01c7198fb614bf8746b46062aa551dff4506dd553ad96817622c76dafe8dc354"  },  {  "uri":"file://%Program Files%\vpn\vpn2.exe",  "digest":"01c7198fb614bf8746b46062aa551dff4506dd553ad96817622c76dafe8dc355"  }  ] |
| ***[as of v2.3.0]***kerberos-ticket | *JSON object* | if requested | JSON object containing Kerberos Ticket Granting Ticket (TGT).  Should present if use-kerberos-authentication was previously requested by the server in auth-requirements. If the caller does not supply Kerberos ticket despite requested by the server, the remaining credentials provided by the caller will be used for authentication.  The ticket should obey the following schema:  {  "client-principal": client principal,  "session-key": base64 encoded session key,  "session-key-encoding": session key encoding,  "tgt": base64 encoded ASN.1 TGT,  "tgt-flags": TGT flags,  "start-time": TGT validity start UTC in ISO 8601,  "end-time": TGT validity end UTC in ISO 8601,  "renew-till": TGT renewal due UTC in ISO 8601  } |

**Response**

HTTP 200 – application/json

|  |
| --- |
| {  “status”: “auth-result”,  “auth-status”: authentication-status,  *[optional]* “delay”: number of seconds the user is disallowed to authenticate as a result of the (previous) failed authentication or because the user’s password is expired,  *[optional]* “password-validity”: password validity on success,  *[optional]* “challenges”: requested challenges,  *[optional]* “response-names”: response names for the given challenges  } |

*auth-status*

authentication status. Can be one of:

“OK” - authentication successful

“DELAY” - authentication was not successful and delay parameter is set

“LOCKED” - cannot login because the user is locked on the server

“EXPIRED” - authentication not successful because the user password is expired

“CHALLENGE”- challenge is supplied by the server and challenges parameter is set

***[as of v2.3.0]*** ”KERBEROS-AUTH-NOK” – validation of Kerberos ticket failed (e.g. expired), a caller can try again with the remaining credentials (no user lock gets applied on failed Kerberos validations)

*delay*

when DELAY is received in auth-status, indicates the time in seconds the caller is suspended from repeating its authentication attempt. Can be 0 which means a caller can try re-authenticating immediately.

***[as of v2.3.0]***  when LOCKED is received in auth-status, indicates the time in seconds the caller is *still* suspended from repeating its authentication attempt. Before v2.3.0 this was communicated as a part of DELAY auth-status.

*password-validity*

when authentication succeeds (“OK” received), indicates the number of seconds until the password expires or -1 if the password never expires. Password validity is supplied only when provided by an authentication backend.

*challenges*

when CHALLENGE is received, contains JSON array of challenges. Challenge names are meant to be displayed to a user during interactive challenge prompt. Challenge values is the value of the challenge to use for response calculation.

Example:

[

{

“name”: “enter first pincode”,

“value”: “981fa356”

},

{

“name”: “enter second pincode”,

value”: “981fa357”

}

]

*response-names*

when CHALLENGE is received, contains JSON array of response names. When multiple responses are required by the server, response name allow identifying each response sent by the caller, thus serving as response keys. Response names can be omitted when only one response is expected by the server.

Example: [“response 1”, “response 2”, “response 3”]

**Example:**

Successful authentication:

|  |
| --- |
| {  “status”: “auth-result”,  “auth-status”: “OK”  } |

Unsuccessful authentication, the caller is suspended for 10 seconds

|  |
| --- |
| {  “status”: “auth-result”,  “auth-status”: “DELAY”,  “delay”: 10,  } |

Extra challenge is requested (RADIUS SecurID authentication)

|  |
| --- |
| {  “status”: “auth-result”,  “auth-status”: “CHALLENGE”,  “challenges”: [{“name”:“Password challenge”, “value”:“Enter your new PIN of 4 to 8 digits, or <Ctrl-D> to cancel the New PIN procedure:”}],  } |

Extra challenge is requested (RADIUS EAP-AKA UMTS challenge-response authentication)

|  |
| --- |
| {  “status”: “auth-result”,  “auth-status”: “CHALLENGE”,  “challenges”: [{“name”:“UMTS AUTN”, ”value”:“01010101010101010101010101010101”},  {“name”:“UMTS RANDOM”, “value”:“101112131415161718191a1b1c1d1e1f”}],  “response-names”: [“RES”, “IK”, “CK”]  } |

When a caller receives CHALLENGE in auth-status from the server, it should proceed as follows:

* provided the set of required credentials does not include RESPONSE, the caller should re-submit all the credentials required by the server, filling PASSWD credential with the response to the received challenge. This is called multi-phase password authentication. Example: RADIUS SecurID authentication.
* provided the set of required credentials includes RESPONSE, the caller should respond with RESPONSE credential only as described below in 2.6.2.1. This is called Challenge-Response authentication. Example: RADIUS EAP AKA/SIM authentication.

#### Challenge-response authentication

#### Request

***[as of v2.3.0]******HTTP POST is used for authentication***

POST /rcdp/<version>/authentication

Content-type: application/x-www-form-urlencoded

**Example:**

**$ curl** -H "Content-Type: application/x-www-form-urlencoded" \

-H "Cookie: keytalkcookie=a77c33e55a1f411396031ce91ee48d9d" \

–H "Expect:" \

-d "responses=%7B%22CK%22%3A+%22123%22%2C+%22RES%22%3A+%22456%22%2C+%22IK%22%3A+%22789%22%7D" \

-X POST <https://test.keytalk.com/rcdp/2.3.0/authentication>

**Query parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **parameter** | **type** | **required** | **description** |
| responses | *JSON object* | yes | JSON array of responses. Response names should be the same as returned by the server on the previous authentication request.  Example:  [  {“name”:“RES”, “value”:“123”},  {“name”:“IK”, “value”:“456”},  {“name”:“CK”, “value”:“789”}  ] |

**Response**

HTTP 200 – application/json

|  |
| --- |
| {  “status”: “auth-result”,  “auth-status”: authentication-status,  *[optional]* “delay”: authentication delay for failed authentication,  *[optional]* “password-validity”: password validity on success,  *[optional]* “challenges”: requested challenges,  *[optional]* “response-names”: response names for the given challenges  } |

*auth-status*

authentication status. Can be one of:

“OK” - authentication successful

“DELAY” - authentication was not successful and delay parameter is set

“LOCKED” - cannot login because the user is locked on the server

“EXPIRED” - authentication not successful because the user password is expired

“CHALLENGE”- challenge is supplied by the server and challenges parameter is set

*delay*

when DELAY is received in auth-status, indicates the time in seconds the caller is suspended from repeating its authentication attempt. Can be 0 which means a caller can try re-authenticating immediately.

*password-validity*

when authentication succeeds (“OK” received), indicates the number of seconds until the password expires or -1 if the password never expires. Password validity is supplied only when provided by an authentication backend.

*challenges*

when CHALLENGE is received, contains JSON array of challenges. Challenge names are meant to be displayed to a user during interactive challenge prompt. Challenge values is the value of the challenge to use for response calculation.

Example:

[

{

“name”: “enter first pincode”,

“value”: “981fa356”

},

{

“name”: “enter second pincode”,

value”: “981fa357”

}

]

*response-names*

when CHALLENGE is received, contains JSON array of response names. When multiple responses are required by the server, response name allow identifying each response sent by the caller, thus serving as response keys. Response names can be omitted when only one response is expected by the server.

Example: [“response 1”, “response 2”, “response 3”]

**Example:**

Successful authentication:

|  |
| --- |
| {  “status”: “auth-result”,  “auth-status”: “OK”  } |

Unsuccessful authentication, the caller is suspended for 10 seconds

|  |
| --- |
| {  “status”: “auth-result”,  “auth-status”: “DELAY”,  “delay”: 10,  } |

Extra challenge is requested (RADIUS SecurID authentication)

|  |
| --- |
| {  “status”: “auth-result”,  “auth-status”: “CHALLENGE”,  “challenges”: [{“name”: “Password challenge”, “value”:“Enter your new PIN of 4 to 8 digits, or <Ctrl-D> to cancel the New PIN procedure:”}],  } |

### Change password

Change user password. Password change facility has to be supported by the server backend such as Active Directory. A caller should normally change his password after EXPIRED authentication result is received from the server. A caller may also choose to change his password on successful authentication when *password-validity* parameter gives a hint that the password is about to expire.

#### Request

***[as of v2.3.0]******HTTP POST is used for changing password***

POST /rcdp/<version>/change-password

Content-type: application/x-www-form-urlencoded

**Example:**

**$ curl** -H "Content-Type: application/x-www-form-urlencoded" \

-H "Cookie: keytalkcookie=a77c33e55a1f411396031ce91ee48d9d" \

–H "Expect:" \

-d " old-password=changeme&new-password=changed" \

-X POST [https://test.keytalk.com/rcdp/2.3.0/change-password](https://test.keytalk.com/rcdp/2.3.0/change-password%20)

**Query parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **parameter** | **type** | **required** | **description** |
| old-password | *string* | yes | Current (old) user password. |
| new-password | *string* | yes | New user password. |

**Response**

See 2.6.2 with authentication status limited to “OK”, “DELAY” or “LOCKED”

“OK” means the password has been successfully changed and the user has to re-authenticate with his new password.

“DELAY” means the password change did not succeed (e.g. incorrect old password or too short new password) and the caller may try again after the given amount of seconds.

## Phase 3 (service provision)

### Check for the last messages

Check for the last server messages. Server messages are meant for KeyTalk users e.g. to indicate planned server maintenance.

#### Request

GET /rcdp/<version>/last-messages

**Example:**

/rcdp/2.3.0/last-messages

/rcdp/2.3.0/last-messages?from-utc=2018-04-26T06%3A49%3A55.614010Z

**Query parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **parameter** | **type** | **required** | **description** |
| from-utc | *UTC string in ISO 8601 including date and time* | no | UTC to request the messages from. Defaults to requesting all server messages. |

**Response**

HTTP 200 – application/json

|  |
| --- |
| {  “status”: “last-messages”,  “messages”: [  {  “text”: message text string,  “utc”: message UTC in ISO 8601 including date and time  },  ....  ]  } |

Example:

|  |
| --- |
| {  “status”: “last-messages”,  {“messages”: [{“text”: “This is user message number 1”,  “utc”: “2017-04-06T04:15:15+0000”},  {“text”: “This is user message number 2”,  “utc”: “2018-03-04T02:10:10+0000”},  {“text”: “This is user message number 3”,  “utc”: “2018-05-02T00:05:05+0000”}]  } |

### Generate certificate on the server

Retrieve a server-generated certificate in the desired format along with a private key.

#### Request

GET /rcdp/<version>/cert

**Example:**

/rcdp/2.3.0/cert?format=P12

/rcdp/2.3.0/cert?format=PEM&include-chain=True

/rcdp/2.3.0/cert?format=P12&out-of-band=True

**Query parameters**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **parameter** | **type** | **required** | **default value** | **description** |
| format | *”P12 or ”PEM”* | yes | n/a | “PEM” to request PEM-encoded X.509 certificate and private key  “P12” to request PKCS#12-encoded X.509 certificate and private key |
| include-chain | *boolean* | no | false | Request the entire certificate chain including suburdinate and root CAs. |
| out-of-band | *boolean* | no | false | ***[as of v2.1.0]*** When set, the server will send back URL to download the certificate instead of the certificate itself. |

**Response**

HTTP 200 – application/json

|  |
| --- |
| {  “status”: “cert”,  “cert”: certificate in the desired format returned when out-of-band is not set.  PEM-encoded certificate has its private key encrypted with the first 30 characters of the session ID sent by the server in keytalkcookie.  When the certificate is delivered in PKCS#12 package, the package gets encrypted with with the first 30 characters of the session ID sent by the server in keytalkcookie and subsequently base64 encoded to be transported with JSON,  “cert-url-templ”: certificate download URL template returned when out-of-band is set.  The template conatins $(KEYTALK\_SVR\_HOST) placeholder that needs to be instantiated with a hostname or IP address of the KeyTalk server used by the caller to make up a valid URL. The download URL is valid for a limited amount of time (normally 5 minutes) and gets invalidated after the first use.  PEM-encoded certificate has its private key encrypted with the first 30 characters of the session ID sent by the server in keytalkcookie.  When the certificate is delivered in PKCS#12 package , the package gets encrypted with with the first 30 characters of the session ID sent by the server in keytalkcookie,  “execute-sync”: boolean flag indicating whether the caller should invoke the service URIs synchronously (true) or asynchronously (false). Defaults to false.  } |

Example regular usage (certificate is returned in the response body):

|  |
| --- |
| {  “status”: “cert”,  “cert”: "-----BEGIN CERTIFICATE-----\nMIIFGTCCAwGgAwIBAgIIWurOaAAAABYwDQYJKoZIhvcNAQELBQAwgYgxHzAdBgkq\nhkiG9w0BCQEWEGluZm9Aa2V5dGFsay5jb20xCzAJBgNVBAYTAk5MMRwwGgYDVQQK\nDBNLZXlUYWxrIElUIFNlY3VyaXR5MRgwFgYDVQQLDA9GYWN0b3J5IERlZmF1bHQx\nIDAeBgNVBAMMF0tleVRhbGsgRGVtbyBTaWduaW5nIENBMB4XDTE4MDUwMzA3NTUw\nNFoXDTE4MDUwMzA5NTUwNFowgZAxETAPBgNVBAMMCERlbW9Vc2VyMQswCQYDVQQG\nEwJOTDEWMBQGA1UECAwNTm9vcmQtQmFyYmFudDESMBAGA1UEBwwJRWluZGhvdmVu\nMRQwEgYDVQQKDAtTaW91eCBHcm91cDEMMAoGA1UECwwDU0VTMR4wHAYJKoZIhvcN\nAQkBFg90ZXN0dWlAc2lvdXguZXUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK\nAoIBAQDJGKTHSL16vsgxIjXvDOTKLk2q518JaIF9Q9ews88NmpVV9cDbOPRxwsns\nSd1kNAXEYi05ScmIc5pGpIV8hyyNjtZ17tiolVO0ALkXgk7hG7wO2Rz+bAQzCdvS\noJjtzo6gZPYcQVlfq+ENMt39ibLqfuAnMLjVpn44fwfqxQFeEsd4doO74E1bUXh7\n7KzaoxsiDAyIITYZe5Azz9Ol47ffg3pRDtq\/6IDYmr7xlBMOq+7QObKBU0pgwNkn\n3JTgkBspXGEXok6S1qNBqJ199NJjdYjiWjHa\/9vSpHSN8RF2s9xrBanLM3S+fnr6\nBx34P6cBoTccllZ9Dpr8IYNJWkanAgMBAAGjfTB7MAkGA1UdEwQCMAAwHQYDVR0l\nBBYwFAYIKwYBBQUHAwIGCCsGAQUFBwMBMAsGA1UdDwQEAwID+DAqBglghkgBhvhC\nAQIEHRYbQ1VTVF9QQVNTV0RfSU5URVJOQUxfVEVTVFVJMBYGA1UdEQQPMA2CC25z\nLnNpb3V4LmV1MA0GCSqGSIb3DQEBCwUAA4ICAQCYKF1OTJqL3eg1JgJdbLPzDo74\nfqZbEBpNkeBFe6nQ6calHJrZNG857WGdfVKfXSOrkwGHmdSNl\/0XM+ySIpcNOWQf\nM9o9rxKQigk4n\/tvjNCiVXRa125t5pUR1ZSyu11SWQAJYc2nPjzasl5B8SwJOIet\nJV80z1pgLFh2GU7hGNiWVqJLF\/U0\/t+xZ1lW1sZ64iih49owTsLt9CL06pD6KPN6\nWvmzLNoK\/ouEeRnYgkyWXvlahGY5N2bPwlq+7+s3BOYRo3APL4N6iVEOUfYDE78K\n05g5zdhVbn717CMx1sQpXggyF5X\/ztQLkrUB5kLT9D7eCBnLDVdjELzl12KJar\/b\ny9eumkCg+Y9PCZN2513o1zUlDLGaH9\/9KdCf6yEca3D3NvnbfCmrDvx10AN+Ht3L\n4XU2L5Rx2rqwB9tj3rZy8i6BK7\/A+ARfg6Tqki5FQ9k667q2hBRPtr69bLeML5at\nyn\/beKjnYnzCRcfXDgnJIKZdfKt2PBM7lh508HNn6aaRZUfHBKHxjMxwuXNMdq9m\nHk6+H8rbRipV\/4xCzEFYvaqlpYO3lOzLIrw8AohRlUzX7UFGm1Dbpn3G2qeikD1Z\nhySYTxjmjXE0DVnPLX05+MR08Eq3hC6QDYs3gBZgP3nILvfEZliOax4fqbT3ijJ9\noxMI+OJsawZMG0uO0w==\n-----END CERTIFICATE-----\n-----BEGIN ENCRYPTED PRIVATE KEY-----\nMIIFDjBABgkqhkiG9w0BBQ0wMzAbBgkqhkiG9w0BBQwwDgQIq9o+wzvbXQQCAggA\nMBQGCCqGSIb3DQMHBAipsAoCJT4gVgSCBMhTb\/8ws1tw9uhH12t9mozccMJQeSAe\nIDxu86RaxgbaMcHj2GnfQjFPou1Ik28eU4Pbi6OEpdlGSBAtrRTK9ZsIOCv+26vN\njrh4gFsLqa9LC\/RB6T7gQFK6nSj+9332d+jCr4tKBIJvSu6hmTGTOraePHb8ic8B\niSHphmz9lN91M311qYKMzhW\/MZg043u2TBJzx1LdsFicIH\/KJ8LXkYQNyM0G663y\nqWpngyjWvzIL7oL5rZh5pv7ygFTuUTy\/1akDW3inuC8fN3\/Zy1374IHeAk4V\/hGQ\nC7FmpF15FTZAYICuKQQsTzUKOd+9Oqlq8YrbcPbHrcMH43UTeaJzjklc3R5K\/mQk\n6a2ggjPc2z4LoFOYEtoPUointBLnRetk7QEHWQdWWW5WfFGRrjbK2t0jZLLVzXuS\nZ0QYBoHeGzFYH0AeYB01DAcT8OC9PAB4r\/vEFdKyXD85OdYdIp4cAbYm5IBB8bYd\nnf9JIV8iifIHy38of6FpHI3AwPZqZTTDaR+arLTjpmpN6d9bRfMNyWUWnJsv0WOo\nd1YuWU\/\/OE0tdvVQKnU1T9FdhbjyW6nQpR8uwhYLi\/BIjpvCUK6ZAe\/+llik0Z2+\nCXnlbU225MOaY2YLS3BizXUkkMcQAo4JE5tEj9vMsEa4VHvt9zcsfpT4vZIGmG2h\nU9UoY2XGhZ4jIEVtqO2ihz7V1ow+kO7eD6HlHMhws9CPZkKh03Z94FK1V\/Sf53U6\ndnR1sAmuuI5HJroXYyX6N5cLguSnwyyvOWRPrUUjqWPZrfvLzndpro6IFPilS7L4\n2fR1DEHwe\/VV0StF31CV6N88KRyGN+gBWrvkGKJ8EozhEz2qToqLBU0CLQ+FVOlE\nuYS30hejXc8wYKFupwSOlhpJUp2B4zC4EbsmTnn7sS55Yk+9NCetE\/k0VMf\/PVVN\nWG0kFhq5CCmtkx8fvvq0nnnNuZS4Hy+tBlEeqMvRvQQ62eRCR94msYG2LCVxRUiB\nNrKQvBM3\/RbxjQFVULr6Wjw9I8dLenjfjou47JLSMShaxlDeAG5iBb0GzLZP6Wlh\nOyXIyusRePxv40GPZsCBRqD2c6fdk52U3Bgk7asctp1L9Y1qP71lbJwnuFtygt+7\nZ+7b38PLltxMRYMCoLD78kugFAP2St0iGGdzdUEWoIP\/IZT2SmMo578CPum3RSHt\nu3lCtHfzzrMIq2o1uTGv+HDswTrLwZt\/VDcaZZUP9a6Vyfzd83jqRXCkFeBk2udM\nHDo5TC6EvLAv9cXqGRW8VSxkJ1WdyxhIdjNSCN+CrECX\/PTbmv5MP9gydnqDSJDq\npCHXZr6dca6vAUGYn5ouQuhrTjsSRsk4M5ZhwgYt9xwCcfNE+juVeweWEJM1GnxP\nmEW3fFSE+NNDfYoPWEA5XEGpR3xF7g9Bj5lT4Yk0XVk\/ED3hTx0VI8g2IZGrvt40\nyh+\/OxyxB9zUzsleQVDitmzQnqti3nXReHwyenO0p9frC5J\/o4ibYKkPF9lH9\/UK\nh8SCSLpWBil\/8RBQ8kD0Pms5G\/Z2TNS6dnwrXZU+so1pl+Kk+T+TTjKkDp8U1xkv\nWClAUsbs8gO0289SjGjhPge0c4UWRiKLElj6jDx0g3yHoJU8bi6pMnJzVeg7IhLF\nxK8=\n-----END ENCRYPTED PRIVATE KEY-----\n“  } |

*Notice again that JSON-serialization of PEM certificates requires forward slashes ‘/’ to be escaped as ‘\/’*

Example when certificate download URL is returned:

|  |
| --- |
| {  “status”: “cert”,  “cert-url-templ”: “ http://$(KEYTALK\_SVR\_HOST):8000/cert/?cbf498dc683c4e0499fd7e2d27640917“  } |

### *[as of v2.2.0]* Query CSR requirements

Client might want to generate a key pair itself and submit the CSR to KeyTalk server for signing. Before generating a key pair the client should ask the server for the initial parameters for the CSR such as key size, signing algorithm and certificate subject.

#### Request

GET /rcdp/<version>/csr-requirements

**Example:**

/rcdp/2.2.2/csr-requirements

**Response**

HTTP 200 – application/json

|  |
| --- |
| {  “status”: “csr-requirements”,  “key-size”: key size in bits,  “signing-algo”: algorithm to use for CSR signing,  “subject”: dictionary of subject fields to use in CSR  } |

Example:

|  |
| --- |
| {“status”: “csr-requirements”,  “key-size”: “2048”,  “signing-algo”: “sha256”,  “subject”: {“cn”: “TestUser”,  “c”: “NL”,  “st”: “Utrecht”  “l”: “Amsersfort”,  “o”: “KeyTalk”,  “ou”: “Development”,  “e”: “test@keytalk.com”,  }  } |

### *[as of v2.2.0]* Generate certificate from the client CSR

Retrieve a PEM-encoded certificate from the CSR supplied by the client. The CSR should be created from the parameters retrieved from csr-requirements call described in 2.7.3.

#### Request

POST /rcdp/<version>/cert

Content-type: application/x-www-form-urlencoded

**Example:**

**$ curl** -H "Content-Type: application/x-www-form-urlencoded" \

-H "Cookie: keytalkcookie=a77c33e55a1f411396031ce91ee48d9d" \

-H "Expect:" \

-d "csr=-----BEGIN+CERTIFICATE+REQUEST-----%0AMIIC1jCCAb4CAQAwgZAxCzAJBgNVBAYTAk5MMRIwEAYDVQQHDAlFaW5kaG92ZW4x%0ADDAKBgNVBAsMA1NFUzEUMBIGA1UECgwLU2lvdXggR3JvdXAxFjAUBgNVBAgMDU5v%0Ab3JkLUJhcmJhbnQxETAPBgNVBAMMCERlbW9Vc2VyMR4wHAYJKoZIhvcNAQkBFg90%0AZXN0dWlAc2lvdXguZXUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDG%0AfyCCkM7cbVhpBCSx1Nf%2BFDqa9banKf9sPRW5VwBFYP5siLdsywNkNqrFYcV0w6ss%0Ath21qK9bkjZoyiKpbzvzgQw08NlbBmJfj700Ol8HUn2xLvp2z6J6q3Z4rAR4d8jx%0ApwcdRlPeJO5b3OtBaURKILaJTjtsUVyCXr%2B6u%2FgiuaD0DGBKsIQccyAWGy%2B1zNer%0AsmUib%2FsnWHEaAPJtvg7T2amaWACKcqIOppR%2BHDJUUNSYYju9xZqCLjx6Y2%2B2ZXHK%0AMpFcFsP%2F8GCYGZ2%2FAIlWtsVzKSaRWmTVJfBsy50gW3YmwI0QYghl52NIDQuBJeoT%0AmQFxsKXpqcWjpP3KTOS5AgMBAAGgADANBgkqhkiG9w0BAQsFAAOCAQEAbUVCaYm%2F%0Aw1otZaLgtCP2mIVVH%2FgHvTeVFs1436Lz%2FaKT5q1QRee81C2us1z9G7h3PG%2BM6w1N%0AUJauwqQ2mR2c1VAidROdT52syNPR4jXeR11%2F7a%2FmsZFqaw3%2FLlwVtBJHEfOA6apU%0AjSVWi6%2F3kUjD0FhYHAufKm2nJ10qGnwC5xpzuvYOQsUFFobLZoyGq5NNEgnSpK8X%0A9A9j5kKGBOm9eQOrWxw%2F0UlwRqLpt6l76Gt5%2BlMp5BtTCPK2uboHvJiPu4aJUuHh%0Afx9ZjKox73V%2BleOEmNSYfesuQPE5AwiFkE988NFixGXOHw7uQdWc9SFsYFRFZG2p%0AYb%2Bm9iFyUY8AHw%3D%3D%0A-----END+CERTIFICATE+REQUEST-----%0A" \

-X POST https://test.keytalk.com/rcdp/2.3.0/cert

**Request POST parameters:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **parameter** | **type** | **required** | **default value** | **description** |
| csr | *string* | yes | n/a | Base64 encoded PKCS#10 certificate signing request |
| include-chain | *boolean* | no | false | Request the entire certificate chain including subordinate and root CAs. |
| out-of-band | *boolean* | no | false | When set, the server will send back URL to download the certificate instead of the certificate itself. |

**Response**

HTTP 200 – application/json

|  |
| --- |
| {  “status”: “cert”,  “cert”: PEM-encoded certificate returned when out-of-band is not set,  “cert-url-templ”: certificate download URL template returned when out-of-band is set.  The template contains $(KEYTALK\_SVR\_HOST) placeholder that needs to be instantiated with a hostname or IP address of the KeyTalk server used by the caller to make up a valid URL. The download URL is valid for a limited amount of time (normally 5 minutes) and gets invalidated after the first use,  “execute-sync”: boolean flag indicating whether the caller should invoke the service URIs synchronously (true) or asynchronously (false). Defaults to false.  } |

Example regular usage (certificate is returned in the response):

|  |
| --- |
| {  “status”: “cert”,  “cert”: “-----BEGIN CERTIFICATE-----\nMIIFGTCCAwGgAwIBAgIIWurNEwAAABUwDQYJKoZIhvcNAQELBQAwgYgxHzAdBgkq\nhkiG9w0BCQEWEGluZm9Aa2V5dGFsay5jb20xCzAJBgNVBAYTAk5MMRwwGgYDVQQK\nDBNLZXlUYWxrIElUIFNlY3VyaXR5MRgwFgYDVQQLDA9GYWN0b3J5IERlZmF1bHQx\nIDAeBgNVBAMMF0tleVRhbGsgRGVtbyBTaWduaW5nIENBMB4XDTE4MDUwMzA3NDky\nM1oXDTE4MDUwMzA5NDkyM1owgZAxCzAJBgNVBAYTAk5MMRIwEAYDVQQHDAlFaW5k\naG92ZW4xDDAKBgNVBAsMA1NFUzEUMBIGA1UECgwLU2lvdXggR3JvdXAxFjAUBgNV\nBAgMDU5vb3JkLUJhcmJhbnQxETAPBgNVBAMMCERlbW9Vc2VyMR4wHAYJKoZIhvcN\nAQkBFg90ZXN0dWlAc2lvdXguZXUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK\nAoIBAQDGfyCCkM7cbVhpBCSx1Nf+FDqa9banKf9sPRW5VwBFYP5siLdsywNkNqrF\nYcV0w6ssth21qK9bkjZoyiKpbzvzgQw08NlbBmJfj700Ol8HUn2xLvp2z6J6q3Z4\nrAR4d8jxpwcdRlPeJO5b3OtBaURKILaJTjtsUVyCXr+6u\/giuaD0DGBKsIQccyAW\nGy+1zNersmUib\/snWHEaAPJtvg7T2amaWACKcqIOppR+HDJUUNSYYju9xZqCLjx6\nY2+2ZXHKMpFcFsP\/8GCYGZ2\/AIlWtsVzKSaRWmTVJfBsy50gW3YmwI0QYghl52NI\nDQuBJeoTmQFxsKXpqcWjpP3KTOS5AgMBAAGjfTB7MAkGA1UdEwQCMAAwHQYDVR0l\nBBYwFAYIKwYBBQUHAwIGCCsGAQUFBwMBMAsGA1UdDwQEAwID+DAqBglghkgBhvhC\nAQIEHRYbQ1VTVF9QQVNTV0RfSU5URVJOQUxfVEVTVFVJMBYGA1UdEQQPMA2CC25z\nLnNpb3V4LmV1MA0GCSqGSIb3DQEBCwUAA4ICAQCCca0ClI9Dw+iO7IIqMZ8UKzhq\n8MWcbpthcgFHPHdxqFYIfTWYOzXCN8FVq96oHH2e09anBYopGyHW+a5oMbY8bKbP\nvGD6\/CslC8nFFqkQfRTH6nanDSq18S\/4uc3bMaIQvWzv5mEYpiTKtKCSUMfV7FLN\nS64I\/UQNglEhHMul1UyL0NM3xU8QYmz+k6qnkw2C3M5Y9eprUT9iZxXCm4XGJo7j\nUPBIRBXUCsaPz+UdK0Syq2Hl\/IsREt5iPRJIU\/B4FjduJlD1R68ZAyNnyOeDQI7f\nEJWUeBYC2QwdlXW3FqKdwki928wksRpY4x3Fyz9\/f32chZQOihee378HP9PDiTZQ\nFCIWSsrO+WUUjToehK2ErgqwCrH0Ydw5ZuIV1vVivGzlgmDHmIQY6uPnYasa1kQw\nspY2JyvlZA\/9mhCvfupwB6L4QIA8yjwNoM3MAsZgq4fvk1kxm\/k1pRMPB2bSGy4u\nFLyMoodTAYJfpzH\/gCwWnrYowqw2T67HsPqBBiOnsuaA0h4k\/m88i4ypcv5f48wJ\nzcxaXqRqWqxzw\/efkYg5m4HdncAPU05NxwJmP17n77l88MZvKc0wVbA+22vCBgCi\nMaOYWhnkTuBN90AoaYAJwe1bkLlbTFMZJjsNPvvS5sAk1l9NihCrXS8ZWtZRfGYz\ngPkm+UPWboYdQbKCRg==\n-----END CERTIFICATE-----\n“  } |

*Notice again that JSON-serialization of PEM certificates requires forward slashes ‘/’ to be escaped as ‘\/’*

Example when certificate download URL is returned:

|  |
| --- |
| {  “status”: “cert”,  “cert-url-templ”: “ http://$(KEYTALK\_SVR\_HOST):8000/cert/?cbf498dc683c4e0499fd7e2d27640917“  } |

# PUBLIC API

There is a set of API to query various information from KeyTalk server without the need to authenticate.

## Public API versions

|  |  |  |
| --- | --- | --- |
| **REST API version** | **Supported KeyTalk server** | **Changes wrt the previous API version** |
| 1.0.0 | 5.3.2 and up | n/a |
| 1.1.0 | 5.5.0 and up | Add enrolment of S/MIME certs for external parties |

## API overview

The communication goes over HTTPS and uses port 443.

### Retrieve self-service availability

Retrieves whether self-service is available for the given account.

#### Request

POST /public/1.0.0/self-service-availability

Content-type: application/x-www-form-urlencoded

**Request POST parameters:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **parameter** | **type** | **required** | **default value** | **description** |
| cert | *string* | yes | n/a | PEM-encoded X.509 user certificate previously received from KeyTalk identifying the caller |

**Response**

HTTP 200 - application/json – successful invocation

|  |
| --- |
| {  “status”: “self-service-availablility”,  “available”: boolean  } |

HTTP 400 – application/json – invalid request

|  |
| --- |
| {  “status”: “error”,  “error”: error message (optional)  } |

### Retrieve address book URLs

Retrieves URLs of address books used by back-end LDAP/AD servers.

#### Request

GET /public/1.0.0/address-book-list

**Example:**

/public/1.0.0/address-book-list?service=DEMO\_SERVICE

**Query parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **parameter** | **type** | **required** | **description** |
| service | *string* | yes | KeyTalk service name |

**Response**

HTTP 200 - application/json – successful invocation

|  |
| --- |
| {  “status”: “address-book-list”,  “address-books”: [  {“ldap\_svr\_url”: LDAP server URL,  “search\_base”: LDAP server search base DN (e.g. “ou=people,dc=example,dc=com”,  “verification\_ca”: PEM-encoded X.509 verification CA(s) of the LDAPs server(optional for LDAPs URL),  },  …  ]  } |
|  |

Example response when no address books configured for the service

|  |
| --- |
| {  “status”: “address-book-list”,  “address-books”: []  } |

HTTP 400 – application/json – invalid request

|  |
| --- |
| {  “status”: “error”,  “error”: error message (optional)  } |

### *[as of v1.1.0]* Retrieve availability of S/MIME certificate enrollment to external parties for self-service

Check the availability and requirements for S/MIME certificate enrollment to external parties for the given self-service account.

#### Request

POST /public/1.1.0/smime-cert-enrollment-availability

Content-type: application/x-www-form-urlencoded

**Request POST parameters:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **parameter** | **type** | **required** | **default value** | **description** |
| cert | *string* | yes | n/a | PEM-encoded X.509 S/MIME certificate previously received from KeyTalk identifying the caller as self-service–eligible user |

**Response**

HTTP 200 - application/json – enrollment available for the given self-service user

|  |
| --- |
| {  “status”: “smime-cert-enrollment-availability”,  “available”: true,  “mobile-required”: boolean  } |

HTTP 200 - application/json – enrollment not available for the given self-service user

|  |
| --- |
| {  “status”: “smime-cert-enrollment-availability”,  “available”: false,  “reason”: text  } |

HTTP 400 – application/json – invalid request (e.g. user certificate is not S/MIME)

|  |
| --- |
| {  “status”: “error”,  “error”: error message (optional)  } |

# SELF-SERVICE API

There is a set of API to be used by KeyTalk self-service user. The API requires client certificate and key as an authentication means. This API should not be confused with the Public API which does not require authentication yet might require a client certificate (without a key) to identify a self-service user. In this case the user certificate is communicated as a part of REST API call, whereas the Self-Service API requires the certificate and the key during TLS connection establishment phase.

## API versions

|  |  |  |
| --- | --- | --- |
| **REST API version** | **Supported KeyTalk server** | **Changes wrt the previous API version** |
| 1.0.0 | 5.5.0 and up | n/a |

## API overview

The communication goes over HTTPS and uses port 3000. All API calls should be authenticated with a certificate and key identifying the caller as KeyTalk self-service user. KeyTalk server should be configured to require certificate-based logins.

### Enroll S/MIME certificate for external parties

Enroll a S/MIME certificate to an external party i.e. to a user not (likely) known by KeyTalk. The certificate will be communicated to the indicated email address. It is strongly recommended to call smime-cert-enrollment-availability from the Public API before enrolling a certificate to minimize the chance of enrollment errors.

#### Request

POST /ssapi/1.0.0/smime-cert-enrollment

Content-type: application/x-www-form-urlencoded

**Request POST parameters:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **parameter** | **type** | **required** | **default value** | **description** |
| recipient-email | *string* | yes | n/a | Email of the S/MIME recipient. The link to download the S/MIME certificate will be sent to this email address. |
| recipient-mobile | *string* | only if enforced by the server | empty | Mobile phone number to receive the password for the S/MIME certificate and key. Required if enforced by the server configuration. If not enforced and not supplied by the caller the password will be sent in the email. |
| svr-host-name | *string* | no | hostname extracted from the KeyTalk server’s web management certificate otherwise IP address | KeyTalk server hostname to make up a download link to for the generated S/MIME certificate. This link is communicated to the S/MIME recipient hence the hostname should be routable for this person. |

**Response**

HTTP 200 - application/json – enrollment succeeded

|  |
| --- |
| {  “status”: “success”,  “address-books”: [  {“ldap-svr-url”: LDAP server URL,  “search-base”: LDAP server search base DN (e.g. “ou=people,dc=example,dc=com”,  “verification-ca”: PEM-encoded X.509 verification CA(s) of the LDAPs server(optional for LDAPs URL),  },  …  ]  } |

HTTP 400 – application/json – invalid request (e.g. invalid mobile phone number supplied or SMTP not configured for the given service)

|  |
| --- |
| {  “status”: “error”,  “error”: error message (optional)  } |

HTTP 500 – application/json – server-side enrolment error

|  |
| --- |
| {  “status”: “error”,  “error”: error message (optional)  } |

# Certificate AUTHORITY RETRIEVAL API (CA API)

Besides strongly authenticated TLS-secured RCDP API, KeyTalk server also supports unauthenticated plain-HTTP REST API to retrieve trusted and intermediate signing certificate authorities. CA API is meant to be called by KeyTalk clients in order to roll out the initial trust CAs on the system before RCDP API comes into play. The same effect can be achieved by deploying RCCD files, though parsing RCCD is far more complex task compared to downloading a single file over HTTP.

The calls go over plain HTTP iso HTTPS because at the stage of calling CA API a KeyTalk client is not yet supposed to possess a trusted KeyTalk communication CA to establish secure TLS connection to the server. In any case the retrieved certificates contain only public information hence no secrets are leaked by using plain HTTP.

## CA API versions

|  |  |  |
| --- | --- | --- |
| **REST API version** | **Supported KeyTalk server** | **Changes wrt the previous API version** |
| 1.0.0 | 5.3.1 and up | n/a |

## CA API overview

The communication goes over HTTP and uses port 8000.

### Retrieve internal signing CA

Retrieve KeyTalk Signing CA or KeyTalk Primary CA or KeyTalk root CA for a user certificate that will be eventually received via RCDP call. Each subsequent CA is an issuer of the previous one.

The received CAs are KeyTalk internal CAs (i.e. not from GlobalSign or Microsoft CA tree) corresponding to “Signing CA” “Primary CA” and “Root CA”, on the KeyTak admin web panel. A typical KeyTalk internal CA tree is 2 level deep with self-signed Primary CA and no Root CA.

#### Request

GET /ca/1.0.0/signing

GET /ca/1.0.0/primary

GET /ca/1.0.0/root

**Response**

HTTP 200 - application/octet-stream - PEM-encoded CA certificate is returned in HTTP response body

HTTP 404 – CA does not exist (e.g. for Root CA)