

KeyTalk - Protocols

Author	MR vd Sman
Creation date	14-March-2017
Last updated	23-July-2021
Document version	2.5.4
Document status	Qualified
Product	KeyTalk certificate and key management & enrolment virtual appliance
Data classification	Public

TABLE OF CONTENTS

TABLE OF CONTENTS.....	2
1. INTRODUCTION.....	5
1.1 Purpose.....	5
1.2 Scope.....	5
2. CERTIFICATE RETRIEVAL API (RCDPV2).....	6
2.1 KeyTalk config file.....	6
2.2 RCDPV2 overview.....	7
2.3 RCDPV2 communication phases.....	9
2.4 Messages sent in all phases.....	10
2.4.1 End Of communication.....	10
2.4.2 Error.....	10
2.5 Phase 1 (handshake).....	12
2.5.1 Hello.....	12
2.5.2 Handshake.....	12
2.6 Phase 2 (authentication).....	14
2.6.1 Request authentication requirements.....	14
2.6.2 Authentication.....	16
2.6.3 Change password.....	24
2.7 Phase 3 (service provision).....	25
2.7.1 Check for the last messages.....	25
2.7.2 Generate certificate on the server.....	26
2.7.3 [as of v2.2.0] Query CSR requirements.....	30
2.7.4 [as of v2.2.0] Generate certificate from the client CSR.....	31
3. PUBLIC API.....	34
3.1 Public API versions.....	34
3.2 API overview.....	34
3.2.1 Retrieve self-service availability.....	34
3.2.2 Retrieve address book URLs.....	36
3.2.3 [as of v1.1.0] Retrieve availability of S/MIME certificate enrollment to external parties for self-service.....	38

3.2.4 [as of v1.3.0] Query the certificate store to place the resulted certificate.....	40
3.2.5 [as of v1.4.0] Request a disclaimer to install to the mail client.....	41
3.2.6 [as of v1.5.0] Query certificate approver emails.....	42
3.2.7 [as of v1.6.0] Query KeyTalk server version.....	43
3.2.8 [as of v1.6.1] Query Common Name customization policy.....	44
4. ADMINISTRATOR API.....	45
4.1 API versions.....	45
4.2 API overview.....	45
4.2.1 [as of v1.1] Enroll user from a server-generated keypair.....	45
4.2.2	46
4.2.3 [as of v1.1] Enroll user with client CSR.....	47
4.2.4 Revoke user certificates.....	50
4.2.5 [as of v1.2] Download KeyTalk settings.....	52
4.2.6 [as of v1.3] Retrieve SCEP configuration.....	54
4.2.7 [as of v1.4] Copy KeyTalk TEMPLATE.....	56
5. SELF-SERVICE API.....	58
5.1 API versions.....	58
5.2 API overview.....	58
5.2.1 Enroll S/MIME certificates for external parties.....	58
6. CERTIFICATE AUTHORITY RETRIEVAL API (CA API).....	62
6.1 CA API versions.....	62
6.2 CA API overview.....	62
6.2.1 Fetch internal signing CA.....	62
7. OUTBOUND CERTIFICATE MANAGEMENT API.....	64
7.1 Obtaining API credentials.....	64
7.2 Certificate Provisioning API.....	65
7.2.1 Request a new certificate.....	65
7.2.2 Renew a certificate.....	69
7.2.3 Query a certificate.....	71
7.2.4 Revoke a certificate.....	73
7.2.5 Query an account information.....	75

1. INTRODUCTION

1.1 Purpose

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

1.2 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.

2. CERTIFICATE RETRIEVAL API (RCDPV2)

This section describes certificate retrieval REST API called RCDPV2.

RCDP version	Minimal KeyTalk server version	Changes wrt the previous RCDP version
2.0.0	4.6.0	
2.1.0	5.3.0	Allow caller to request a certificate download URL in the the <code>cert</code> response instead of a certificate body.
2.2.0	5.3.1	- 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	- 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
2.4.0	5.5.0	Add flag instructing the caller to store certificate to the System (Machine) Store instead of the User Store
2.4.1	5.5.3	Add <code>apply-address-books</code> flag and <code>address-books</code> to the <code>cert</code> response
2.4.2	5.5.10	Add <code>apply-smime-settings</code> flag to the <code>cert</code> request
2.4.3	5.6.1	Add <code>supply-computer-name</code> flag to the <code>auth-requirements</code> response and <code>computer-name</code> to the authentication request
2.4.4	5.6.5	Add <code>retired-computer-name</code> to the <code>authenticate</code> request
2.4.5	5.7.2	Add <code>signature</code> to the <code>cert</code> request
2.5.0	5.8.7	- Add a possibility for the server to request an extra info from the caller during the phase 3 before proceeding with an issuance of a certificate - The <code>cert</code> request should be sent as HTTP POST - Change <code>signature</code> object to <code>set-disclaimer</code> flag in the <code>cert</code> response
2.5.1	6.1.1	- Add a possibility to authenticate using OTP (One Time Password)
2.5.2	6.1.6	Added <code>common-name</code> parameter to the <code>cert</code> request.

2.1 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 CA for 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

2.2 RCDPv2 overview

Communication in RCDPv2 is encapsulated in RESTful calls over HTTPS port 443 secured by KeyTalk internal CAs or, when enabled server-side, over HTTPS ports 443 and 4443 secured by a public trusted CA such as GlobalSign. Optional out-of-band certificate downloads are made possible over HTTP using port 8000 and, when enabled server-side, also over HTTPS port 8443 secured by known trusted CA.

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

HTTP Header	Required	Description
GET	YES	/rdcp/<version>/<action>?<request-params>
Host	YES	Should contain the FQDN or IP of KeyTalk server.
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:

```
POST /rdcp/2.5.2/authentication 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'}
```

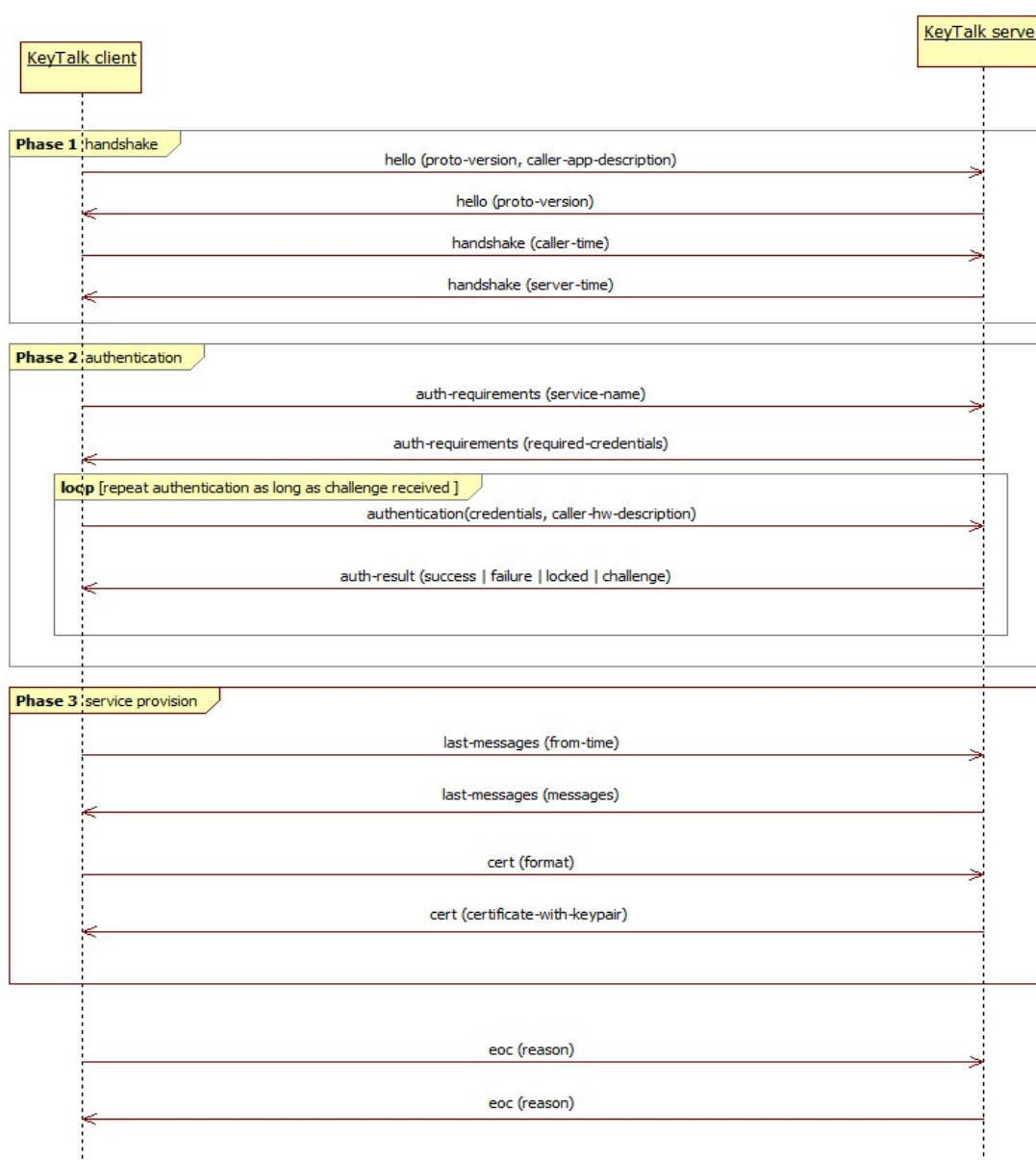
2.3 RCDPv2 communication phases

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.

2.4 Messages sent in all phases

2.4.1 End Of communication

Request

GET /rcdp/<version>/eoc

Example:

/rcdp/2.5.2/eoc
/rcdp/2.5.2/eoc?reason=bye%2C+server

Query parameters

parameter	type	required	description
reason	<i>string</i>	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.

2.4.2 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/2.5.2/error?code=1066&description=invalid+response

Query parameters

parameter	type	required	description
code	<i>number</i>	yes	numeric error code
reason	<i>string</i>	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.
<i>[as of v2.5.0]</i> 1006 (ErrIncompatibleProtocolVersion)	optional	server -> client	Sent by the server when further communication is not possible because the client's API version does not support it (normally too old)

2.5 Phase 1 (handshake)

2.5.1 Hello

Agree on RCDP API version and establish session ID.

Request

GET /rcdp/<version>/hello

Example:

```
/rcdp/2.5.2/hello
/rcdp/2.5.2/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.

2.5.2 Handshake

Confirm version handshake and exchange time information.

Request

GET /rcdp/<version>/handshake

Example:

```
/rcdp/2.5.2/handshake?caller-utc=2016-04-22T10%3A44%3A35Z
```

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  
}
```

2.6 Phase 2 (authentication)

2.6.1 Request authentication requirements

Request authentication requirements from the server.

Request

GET /rdcp/<version>/auth-requirements

Example:

/rdcp/2.5.2/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,
  [as of v2.4.3][optional] "supply-computer-name": if the machine/device name
  should be supplied to the server,
}
```

credential-types

JSON array of credential types required to authenticate against the given service. Supported credential types are: "USERID", "HWSIG", "PASSWD", "PIN", "RESPONSE" and, **as of v2.5.1**, "OTP".

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 to 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"
}
```

2.6.2 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&P
ASSWD=change%21&resolved=%5B%7B%22ips%22%3A+%5B%2281.175.103.107%22%5D%2C+%22
uri%22%3A+%22https%3A%2F%2Fdemo.keytalk.com%2F%22%7D%5D" \
-X POST https://test.keytalk.com/rcdp/2.5.2/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.
<i>[as of v2.5.1]</i> OTP	string	if requested	After receiving WAIT-FOR-OTP (see 2.6.2.2) from the KeyTalk server the caller received a One Time Password (OTP) via an alternative channel. Required if OTP was previously set by the server in auth-requirements response. Can only be used once per positive authentication.
resolved	JSON array	if requested	JSON array of objects containing service URIs accompanied with RFC 3986-compliant IPv4 or IPv6 address resolved from the URI hostname. Required if resolve-service-uris was previously set in auth-requirements response.

Example:			
<pre>[{ "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]"] }]</pre>			
digests	JSON array	if requested	JSON array of objects containing service URIs accompanied with SHA-256 hexadecimal digest of the underlying file. Required if <code>calc-service-uris-digest</code> was previously set in <code>auth-requirements</code> response. Example: <pre>[{ "uri": "file://%Program Files%\\vpn\\vpn.exe", "digest": "01c7198fb614bf8746b46062aa551dff4506dd553ad96817622c76dafa8dc354" }, { "uri": "file://%Program Files%\\vpn\\vpn2.exe", "digest": "01c7198fb614bf8746b46062aa551dff4506dd553ad96817622c76dafa8dc355" }]</pre>
[as of v2.3.0] kerberos-ticket	JSON object	if requested	JSON object containing Kerberos Ticket Granting Ticket (TGT). Should present if <code>use-kerberos-authentication</code> was previously requested by the server in <code>auth-requirements</code>. 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: <pre>{ "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 }</pre>
[as of v2.4.3] computer-name	string	if requested	Caller's machine/device name. Required if <code>supply-computer-name</code> was previously set by the server in <code>auth-requirements</code> response.
[as of v2.4.4] retired-computer-name	string	conditional	Only pass if all the following scenarios are true: - <code>supply-computer-name</code> was previously set by the server in <code>auth-requirements</code> response. - The current machine/device name is different from the

machine/device in the latest valid certificate.

Machine/device name can be found in a certificate using the following guidelines:

SAN contains one or more DNS values

& the certificate CN should equal a DNS value in SAN

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)

[\[as of v2.5.1\]](#) "WAIT-FOR-OTP" - caller's identity successfully checked, the caller should wait for the OTP to be sent (e.g. to his email represented by USERID) and re-submit authentication request with this OTP as a credential.

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.

2.6.2.1 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/2.5.2/authentication
```

Query parameters

parameter	type	required	description
responses	<i>JSON object</i>	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:"}],
}
```

2.6.2.2 [\[as of v2.5.1\]](#) Start OTP (One Time Password) authentication

Start OTP authentication after if OTP was previously received by the server in `auth-requirements` response and the caller does not yet requested a One-Time Password from the server. Once successful the caller should receive an OTP for USERID supplied in the request (normally email). This OTP should later be used to authenticate against the server. The OTP should only be used once.

Request

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=demo%40keytalk.com&HWSI
G=123456" \
-X POST https://test.keytalk.com/rcdp/2.5.2/authentication
```

Query parameters

parameter	type	required	description
service	<i>string</i>	yes	KeyTalk service name
caller-hw-description	<i>string</i>	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	<i>string</i>	if requested	ID of the user, normally a e-mail

HWSIG	<i>string</i>	if requested	Hardware Signature of the caller's device calculated with the formula specified in the previous <code>auth-requirements</code> server response. Required if HWSIG was previously set by the server in <code>auth-requirements</code> response.
computer-name	<i>string</i>	if requested	Caller's machine/device name. Required if supply-computer-name was previously set by the server in <code>auth-requirements</code> response.

Response

HTTP 200 - application/json

```
{
  "status": "auth-result",
  "auth-status": authentication-status,
  [optional] "delay": authentication delay for failed authentication,
}
```

auth-status

authentication status. Can be one of:

"WAIT-FOR-OTP" - the caller's identity successfully verified, and the caller should wait for the OTP to be sent (e.g. to his email represented by USERID) and re-submit the authentication request with this OTP as a credential.

"DELAY" - authentication was not successful and `delay` parameter is set

"LOCKED" - cannot login because the user is locked on the server

"OK" - authentication successful. Normally should not be sent in this flow because it means that OTP is not required on the server.

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.

Example:

Successful OTP initiation:

```
{
  "status": "auth-result",
  "auth-status": "WAIT-FOR-OTP"
}
```

Unsuccessful OTP initiation, the caller is suspended for 10 seconds

```
{
  "status": "auth-result",
  "auth-status": "DELAY",
  "delay": 10,
}
```

2.6.3 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" \
-d "old-password=changeme&new-password=changed" \
-X POST https://test.keytalk.com/rcdp/2.5.2/change-password
```

Query parameters

parameter	type	required	description
old-password	<i>string</i>	yes	Current (old) user password.
new-password	<i>string</i>	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.

2.7 Phase 3 (service provision)

2.7.1 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/2.5.2/last-messages
/rcdp/2.5.2/last-messages?from-utc=2018-04-26T06%3A49%3A55.614010Z
```

Query parameters

parameter	type	required	description
from-utc	<i>UTC string in ISO 8601 including date and time</i>	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"}
  ]
}
```

2.7.2 Generate certificate on the server

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

Request

[\[as of v2.5.0\]](#) HTTP POST is used for certificate requests

POST /rcdp/<version>/cert

Request POST 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 subordinate and root CAs, when available
[as of v2.1.0] out-of-band	boolean	no	false	When set, the server will send back URL to download the certificate instead of the certificate itself.
[as of v2.5.0] response	JSON object	if requested	n/a	Response sent by the client after the challenge status received in the previous "cert" request. Format: { "type": type of the previously received challenge, "data": Base64-encoded response }
[as of v2.5.0] cookie	JSON object	if requested	n/a	Cookie received in the previous response, typically of type "submitted" or "in-progress" or "challenge".
[as of v2.5.2] common-name	string	no	empty string	Common Name to be used in the certificate. Only honored if allowed by the server. Use cn-customization-policy Public API call to check.

Response

HTTP 200 - application/json

Certificate is successfully issued by the server

```
{
```

```
"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 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 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.

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 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.

[as of v2.4.0] "store-to-system": if set, instruct the caller to store the generated certificate to the System Store (Machine Store) instead of the User Store. Defaults to false.

[as of v2.4.1] "apply-address-books": boolean flag indicating whether the caller should apply the address books, typically to be used by an email client. Applying the address books should allow the user to send encrypted emails and verify email signatures to other users registered to the address books

[as of v2.4.1] "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),
 },
 ...]

[as of v2.4.2] "apply-smime-settings": boolean flag indicating whether the caller should apply S/MIME settings to an email client

[as of v2.5.0] "set-disclaimer": boolean flag indicating whether the caller should apply a disclaimer to an email client signatures. The disclaimer(s) can be fetched from the Public API
 }

Example typical usage, the certificate is returned in the response body:

```
{
  "status": "cert",
  "cert": "-----BEGIN CERTIFICATE-----
\nMIIFGTCCAwwGgAwIBAgIIWurOaAAAABywDQYJKoZIhvcNAQELBQAwYgYxH2AdBgkq\n\nhkiG9w0B\nCQEWEGluZm9Aa2V5dGFsay5jb20xCzAJBgNVBAYTAk5MMRwwGgYDVQQK\n\nnDBNLZl1UYWxrIEl1IF\nNlY3VyaXR5MRgwFgYDVQQLDA9GYWN0b3J5IERlZmF1bHhQx\n\nnIDAEBgNVBAMMF0tleVRhbGsgRGVt\nbyBTaWduaW5nIENBMB4XDTE4MDUwMzA3NTUw\n\nnNFoXDTE4MDUwMzA5NTUwNFowZGZAxETAPBgNVBA
```

```

MMCERlbW9Vc2VyMQswCQYDVQGG\nEwJOTDEWMBQGA1UECAwNTm9vcmtQmFyYmFudDESMBAGA1UE
BwwJRWluZGhvdnVu\nMRQwEgYDVQKDAwTaW9leCBHcm9lcDEMAA0GA1UECwwDU0VTMR4wHAYJKo
ZIhvcN\nAQkBFg90ZXN0dWlAc2lvdXguZXUwggEiMA0GCSqGSIB3DQEBAQUAA4IBDwAwggEK\nAo
IBAQDJGKTHSL16vsgxIjXvDOTKLk2q518JaIF9Q9ews88NmpVV9cDbOPRxsns\nSdlkNAXEYi05
ScmIc5pGpIV8hyyNjtZ17tiolVO0ALkXgk7hG7wO2Rz+bAQzCdvS\nnoJjtzo6gZPYcQVlfq+ENMt
39ibLqfuAnMLjVpn44fwfQxQFeEsd4do074E1bUXh7\nn7KzaoxsiDayIITYZe5Azz90147ffg3pR
Dtq\ /6IDYmr7xlBMOq+7QObKBU0pgwNkn\n3JTgkBspXGEXok6S1qNBqJ199NJjdYjiWjHa\ /9vS
pHSN8RF2s9xrBanLM3S+fnr6\nBx34P6cBoTcc1lZ9Dpr8IYNJWkanAgMBAAGj fTB7MAkGA1UdEw
QCMAAwHQYDV01\nnBBYwFAYIKwYBBQUHAWIGCCsGAQUFBwMBAsGA1UdDQEAwID+DAqBg1ghkgB
hvhC\nnAQIEHRYbQ1VTVF9QQVNTV0RfSU5URVJOQUxvFEVTVFVJMBYGA1UdEQQPMMA2CC25z\nnLnNp
b3V4LmVlMA0GCSqGSIB3DQEBCwUAA4ICAQCkYF1OTJqL3eg1JgJdbLPzDo74\nnfqZbEBPnKbFe6
nQ6calHJRZNG857WGdfVKfXSorkwGHmdSN1\ /0XM+ySIpcNOWQf\nnm9o9rxKQigk4n\ /tvjNCiVX
Ra125t5pUR1ZSyu11SWQAJYc2nPjzas15B8SwJOIet\nnJV80z1pgLFh2GU7hGN1WVqJLF\ /U0\ /t
+xZ1lW1sZ64iih49owTsLt9CL06pD6KPN6\nnWvmzLNOk\ /ouEeRnYgkyWXv1ahGY5N2bPwlq+7+s
3BOYRo3APL4N6iVEOUFYDE78K\nn05g5zdhVbn717CMx1sQpXggyF5X\ /ztQLkrUB5kLT9D7eCBnL
DVdjELz112KJar\ /b\nny9eumkCg+Y9PCZN2513o1zU1DLGaH9\ /9KdCf6yEca3D3NvnbfCmrDvx1
0AN+Ht3L\nn4XU2L5R2rQwB9tj3rZy8i6BK7\ /A+ARfg6Tqki5FQ9k667q2hBRPtr69bLeML5at\
nyn\ /beKjnYnzCRcfXDgnJIKZdfKt2PBM7lh508HNN6aaRZUfHBKXjMxwuXNMdq9m\nnHk6+H8rb
RipV\ /4xCzEFYvaqlpY03lOzLIrw8AohR1UzX7UFgm1Dbpn3G2geikD1Z\nnhySYTxjmjXE0DVnPL
X05+MR08Eq3hC6QDYs3gBZgP3nILvFEZliOax4fqbT3ijJ9\nnoxMI+OJsawZMG0u00w==\n-----
END CERTIFICATE-----\n-----BEGIN ENCRYPTED PRIVATE KEY-----
\nMIIFDjBABGkqhkiG9w0BBQ0wMzAbBgkqhkiG9w0BBQwwDgQIQ9o+wzvbXQQCAggA\nnMBQGCCqG
SIB3DQMHBAipsAoCJT4gVgSCBMhTb\ /8ws1tw9uhH12t9mozccMJQeSAe\nnIDxu86RaxgBaMchJ2
GnfQjFPou1lk28eU4Pbi60Epd1GSBAttrRTK9ZsIOcv+26vN\nnjrh4gFsLqa9LC\ /RB6T7gQFK6nS
j+9332d+jCr4tKBIJvSu6hmTGTOraePhb8ic8B\nniSHphmz91N91M311qYKMzhW\ /MZg043u2TBJ
zx1LdsFicIH\ /KJ8LXkYQNYM0G663y\nnqWpnygyWvzIL7oL5rZh5pv7ygFTuTy\ /1akDW3inuC8
fn3\ /Zy1374IHeAk4V\ /hGQ\nnC7FmpF15FTZAYICuKQQsTzUKOd+90qlq8YrbcPbHrcMH43UTeaJ
zjklc3R5K\ /mQk\nn6a2ggjPc2z4LoFOYetoPUointBLnRetk7QEHWQdWWW5WfFGRrjbK2t0jZLLV
zXuS\nnZ0QYBoHeGzFYH0AeYB01DACT8OC9PAB4r\ /vEFdKyXD85OdYdIp4cAbYm5IBB8bYd\nnnf9
JIV8iifIHy38of6FpHI3AwPZqZTTDaR+arLTjpmPN6d9bRfMNYWUWnJsv0W0o\nnd1YuWU\ /\ /OE0
tdvVQKnu1T9FdhbjyW6nQpR8uwYLi\ /BIjpvCUK6ZAe\ /+1lik0Z2+\nCXnlbu225MOay2YLS3B
izXUkkMcQAo4JE5tEj9vMsEa4VHvt9zcsfpt4vZIGmG2h\nnU9UoY2XGhZ4jIEvtq02ihz7V1ow+k
O7eD6H1HMHws9CPZkKh03Z94FK1V\ /Sf53U6\ndnRlsAmuuI5HJroXYyX6N5cLguSnwyvOwRPrU
UjgWPZrfrvLzndpro6IFPils7L4\nn2fR1DEHwe\ /VV0StF31CV6N88KRYGN+gBWrvkGKJ8EozhEz2
qToqLBU0CLQ+FVO1E\nnuYS30hejXc8wYKFupwSolhpJUp2B4zC4EbsmTnn7sS55Yk+9NCetE\ /k0
VMf\ /PVVN\nnWG0kFhq5CCmtkx8fvvqOonnNuZS4Hy+tBlEeqMvRvQQ62eRRCR94msYG2LCVxRuiB\
nNrKQvBM3\ /RbxjQFVULr6Wjw9I8dLenjffjou47JLSMShaxlDeAG5iBb0GzLZP6Wlh\nnOyXIYusR
ePxv40GPZsCBRQd2c6fdk52U3Bgk7asctplL9Y1qP71lbJwnuFtygt+7\nnZ+7b38PLltxMRyMCoL
D78kugFAP2St0iGGdzdUEWoIP\ /IZT2SmMo578CPum3RSHt\nnu3lCtHfzrMIq2o1uTGv+HDswTr
LwZt\ /VDcaZZUP9a6Vyfdz83jqRXCkFeBk2udM\nnHDo5TC6EvLAV9cXqGRW8VSxkJ1WdyxhIdjNS
CN+CrECX\ /PTbmV5MP9gydnqDSJDq\nnpCHXZr6dca6vAUGYn5ouQuhrtjsSRsk4M5ZhwgYt9xwCc
fNE+juVewEWEJM1GnxP\nnmEW3fFSE+NNDfYoPWEA5XEGpR3xF7g9Bj51T4Yk0XV\ /ED3hTx0VI8
g2IZGrvt40\nnyh+\ /OxyxB9zUzsleQVDitmqnqti3nXReHwyen00p9frC5J\ /o4ibYKkPF91H9\
/UK\nnh8SCSLpWBil\ /8RBQ8kD0Pms5G\ /Z2TNS6dnwrXZU+solpl+Kk+T+TTjKkDp8U1xkv\nnWC1
AUsbs8gO0289SjGjhPge0c4UWRiKLElj6jDx0g3yHoJU8bi6pMnJzVeg7IhLF\nnxK8=\n-----
END ENCRYPTED PRIVATE KEY-----\n"
}

```

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

Example when the certificate download URL is returned iso a certificate itself:

```

{
  "status": "cert",
  "cert-url-templ":
  " http://$(KEYTALK_SVR_HOST):8000/cert/?cbf498dc683c4e0499fd7e2d27640917"
}

```

[as of v2.5.0] Response

HTTP 200 - application/json

An extra info is requested from the client before issuing a certificate

```
{
  "status": "cert-challenge",
  "challenge-type": (string) type of the challenge. See below for the details
  "challenge-data": base64-encoded challenge contents interpreted by the client based on the
  challenge type. The client is supposed to reply to this challenge by resending the cert request
  augmented with the response parameter.
  "cookie":
  {
    "name": name,
    "value": value
  }, cookie to return to the server in the subsequent cert request
}
```

Currently the only supported challenge type is "select-globalsign-domainssl-approver-email" accompanied with the list of emails in challenge-data indicating emails eligible for approval the issuance of GlobalSign DomainSSL certificates. The caller is supposed to repeat the previous cert request by adding the selected email address in the response parameter.

[as of v2.5.0] Response

HTTP 200 - application/json

The certificate request is successfully submitted ("submitted") or was previously submitted ("in-progress") to a CA signer (normally to a 3rd party one). The caller is supposed to periodically check the certificate issuance status and retrieve the certificate using the cert request.

```
{
  "status": "submitted"|"in-progress",
  "cookie":
  {
    "name": name,
    "value": value
  }, cookie to return to the server in the subsequent cert request

  "details": (string) extra details to present to the caller
}
```

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

A client might want to generate a key pair by himself and submit the resulted CSR to KeyTalk server for signing. Before doing that the client should query the server for the initial parameters for the CSR such as key size, signing algorithm, certificate subject and a subject alternative name (SAN).

Request

GET /rcdp/<version>/csr-requirements

Example:

/rcdp/2.5.2/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,
  "san": if set, holds an array of Subject Alternative Names to use in CSR
}
```

Example:

```
{
  "status": "csr-requirements",
  "key-size": "2048",
  "signing-algo": "sha256",
  "subject": {
    "cn": "TestUser",
    "c": "NL",
    "st": "Utrecht",
    "l": "Amersfoort",
    "o": "KeyTalk",
    "ou": "Development",
    "e": "test@keytalk.com",
  },
  "san": ["email:test@keytalk.com", "DNS:test.keytalk.com"]
}
```

2.7.4 [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-----
%0AMIIC1jCCAb4CAQAwgZAxCAJBGNVBAYTAk5MMRIwEAYDVQQHDA1FaW5kaG92ZW4x%0ADDAKBgNVBAsMA1NFUzEUMBIGA1UECgwLU21vdXggR3JvdXAxFjAUBGNVBAGMDU5v%0Ab3JkLUJhcmJhbnQxE
TAPBgNVBAMMCERlbW9Vc2VyMR4wHAYJKoZIhvcNAQkBFg90%0AZXN0dWlAc21vdXguZXUwgGEiMA0GCSqGSIb3DQEB
AQUAA4IBDwAwggEKAoIBAQDQG%0AfyCCkM7cbVhpBCSx1Nf%2BFDqa9banKf9sPRW5VwBFYP5siLdsywNkNqrFYcV0w6ss%0Ath21qK9bkjZoyiKpbzvzgQw08N1bBmJfj700O18HUn2xL
vp2z6J6q3Z4rAR4d8jx%0ApwcdRlPeJO5b3OtBaURKILaJTjtsUVyCXr%2B6u%2FgiuaD0DGBKsIQccyAWGy%2B1zNer%0AsmUib%2FsnWHEaAPJtvg7T2amaWACKcqIOppR%2BHDJUUNSYyju9xZqCLjx
6Y2%2B2ZXHK%0AmpFcFsP%2F8GCYGGZ2%2FAI1WtsVzKSaRWmTVJfBsy50gW3YmwI0QYgh152NIDQuBJeoT%0AmQFxsKXpqcWjpP3KTOS5AgMBAAGgADANBgkqhkiG9w0BAQsFAAOCAQEAbUVCaYm%2F%0A
w1otZaLgtCP2mIVVH%2FgHvTeVFs1436Lz%2FAkt5q1QRee81C2us1z9G7h3PG%2BM6w1N%0AUJauwqQ2mR2c1VAidROdT52syNPR4jXeR11%2F7a%2FmsZFqaw3%2FLLwVtBJHEfOa6apU%0AjSVWi6%2
F3kUjD0FhYHAufKm2nJ10qGnwC5xpzuVYOQsUFFobLZoyGq5NNEgnSpK8X%0A9A9j5kKGBom9eQOrWwx%2F0UlwRqLpt6176Gt5%2B1Mp5BtTCPK2uboHvJiPu4aJUuHh%0Afx9ZjKox73V%2BleOEmNSY
fesuQPE5AwifKE988NFixGXOHw7uQdWc9SFSYFRFZG2p%0AYb%2Bm9iFyUY8AHw%3D%3D%0A-----
END+CERTIFICATE+REQUEST-----%0A" \
-X POST https://test.keytalk.com/rcdp/2.5.2/cert
```

Request POST parameters:

parameter	type	required	default value	description
csr	<i>string</i>	yes	n/a	Base64 encoded PKCS#10 certificate signing request
include-chain	<i>boolean</i>	no	false	Request the entire certificate chain including subordinate and root CAs.
out-of-band	<i>boolean</i>	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.

  [as of v2.4.1] "apply-address-books": boolean flag indicating whether the caller
  should apply the address books, typically to be used by an email client. Applying the address books
  should allow the user to send encrypted emails and verify email signatures to other users registered to
  the address books,
  [as of v2.4.1] "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),
    },
    ... ],
  [as of v2.4.2] "apply-smime-settings": boolean flag indicating whether the caller
  should apply smime settings to an email client. ,
  [as of v2.5.0] "set-disclaimer": boolean flag indicating whether the caller should apply
  a disclaimer to an email client signatures. The disclaimer(s) can be fetched from the Public API
}
```

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

```
{
  "status": "cert",
  "cert": "-----BEGIN CERTIFICATE-----
\nMIIFGTCCAwwGAgIBAgIIWurNEwAAABUwDQYJKoZIhvcNAQELBQAwYgYxHAdBgkq\n\nhkiG9w0B\nCQEWEGluZm9Aa2V5dGFsay5jb20xCzAJBgNVBAYTAk5MMRwwGgYDVQQK\n\nnDBNLZlXlUYWxrIElUIF\nNlY3VyaXR5MRgwFgYDVQQLDA9GYWN0b3J5IERlZmFlbHJx\n\nnIDAEgNVBAMMF0tleVRhbGsgRGVt\nbyBTAWduaW5nIENBMB4XDTE4MDUwMzA3NDky\n\nnM1oXDTE4MDUwMzA5NDkyM1owGgAx\nCzAJBgNVBA\nYTAk5MMRiEAYDVQQHDA1FaW5k\n\nnaG92ZW4xDDAKBgNVBAsMA1NFUzEUMBIGA1UECgwLU2lvdXgg\nR3JvdXAxFjAUBgNV\n\nnBAGMDU5vb3JkLUJhcmJhbnQxETAPBgNVBAMMCERlbW9Vc2VyMR4wHAYJKo\nZiHvCN\n\nnaQkBFg90ZXN0dWlAc2lvdXguZXUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK\n\nnao\nIBAQDGfyCCKM7cbVhpBCSx1Nf+FDqa9banKf9sPRW5VwBFYP5siLdsyWnKqRf\n\nnYcV0w6sssth21\nqK9bkjZoyiKpbzvzgQw08NlbBmJfj700018HUN2xLvp2z6J6q3Z4\n\nnrAR4d8jxpwcDRlPeJO5b30\n\nBaURKILaJTjtsUVyCXr+6u\n\n/giuaD0DGBKsIQccyAW\n\nnGy+1zNersmUib\n\n/snWHEaAPJtvg7T2a\nmaWACKcqIoppR+HDJUUNSYyju9xZqCLjx6\n\nny2+2ZXHKMpFcFsp\n\n/8GCYgz2\n\n/A1lWtsVzKSaRWm\nTVJfBsy50gW3YmwIOQYghl52NI\n\nnDQuBJeoTmQFxsKXpqcWjpp3KTOS5AgMBAAAGj\n\nfTB7MAkGA1Ud\nEwQCMAAwHQYDVOR01\n\nnBBYwFAYIKwYBBQUHAWIGCCsGAQUFBwMBMA\n\nsGA1UdDwQEAwID\n\nDAqBg1ghk\nqBhvhC\n\nnaQIEHRYbQ1VTVF9QQVNTVORfSU5URVJOQUx\n\nfVEVTVFVJMBYGA1UdEQQPMA2CC25z\n\nnLn\nNpb3V4LmV1MA0GCSqGSIb3DQEBCwUAA4ICAQCCca0C\n\nlI9Dw+io7IIqMZ8UKzhq\n\nn8MWcbpthcgFH
```

```
PHdxqFYIfTWYOzXCN8FVq96oHH2e09anBYopGyHW+a5oMbY8bKbP\nvGD6\ /Cs1C8nFFqkQfRTH6
nanDSq18S\ /4uc3bMaIQvWzv5mEYpiTKtKCSUMfV7FLN\ns64I\ /UQNglEhHMu1UyL0NM3xU8QY
mz+k6qnkw2C3M5Y9eprUT9iZxXCm4XGJo7j\nUPBIRBXUCsaPz+UdK0Syq2H1\ /IsREt5iPRJIU\
/B4FjduJlD1R68ZAYNnyOeDQI7f\nEJWUeBYC2QwdlXW3FqKdwi928wksRpY4x3Fyz9\ /f32chZ
QOihee378HP9PDiTZQ\nFCIWSsrO+WUUjToehK2ErgqwCrH0Ydw5ZuIV1vVivGzlgmDHmIQY6uPn
Yasa1kQw\nspY2JyvlZA\ /9mhCvfupwB6L4QIA8yJwNoM3MASZgg4fvk1kxm\ /klpRMPB2bSGy4u
\nFLyMoodTAYJfpzH\ /gCwWnrYowqw2T67HsPqBBiOnsuaA0h4k\ /m88i4ypcv5f48wJ\nzcxaXq
RqWqxzw\ /efkYg5m4HdncAPU05NxxJmP17n77188MzvKc0wVbA+22vCBgCi\nMaOYWhnkTuBN90A
oaYAJwelbkLlbTFMZJjsNPvvS5sAk119NihCrXS8ZWtZRfGyz\nngPkm+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"
}
```

3. PUBLIC API

API to query various information from KeyTalk server without the need to authenticate.

3.1 Public API versions

REST API version	Minimal KeyTalk server version	Changes wrt the previous API version
1.0.0	5.3.2	n/a
1.1.0	5.5.0	added enrolment of S/MIME certs for external parties
1.2.0	5.5.1	added synchronous flag to smime-cert-enrollment-availability call
1.2.1	5.5.3	added apply-address-books flag to address-book-list call
1.3.0	5.5.9	allow querying which certificate store (User or System/Machine) the resulted certificate should be placed in
1.4.0	5.7.5	added request for a signature file based on a supplied email address
1.5.0	5.8.7	allow querying certificate approver emails
1.6.0	5.8.13	allow querying KeyTalk version
1.6.1	6.1.5	allow querying Common Name customization policy

3.2 API overview

The communication goes over HTTPS and uses port 443 secured by KeyTalk internal CA or, when enabled server-side, over HTTPS ports 443 and 4443 secured by a public trusted CA such as GlobalSign.

3.2.1 Retrieve self-service availability

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

Request

```
POST /public/1.6.1/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

Example:

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

```
-H "Expect:" \
-d "cert=-----BEGIN%20CERTIFICATE-----
%0AMIIFFDDCCAvSgAwIBAgIIWlSC7gAAAYowDQYJKoZIhvcNAQELBQAwwGxHZAAdBgkq%0AhkiG9w0
BCQEWEGluZm9Aa2V5dGFsay5jb20xCzAJBgNVBAYTAk5MMRwwGgYDVQQK%0ADBNLZXlUYWxrIElUI
FNlY3VyaXR5MRgwFgYDVQQLDA9GYWN0b3J5IERlZmF1bHhQx%0AIDAeBgNVBAMMF0tleVRhbGsgRGV
tbyBTAWduaW5nIENBMB4XDTE4MDEwOTA3NTMw%0AMloXDTI4MDEwNzA4NTMwMlowgZAxETAPBgNVB
AMMCERlbW9Vc2VyMQswCQYDVQQG%0AEwJOTDEWMBQGA1UECAwNTm9vcmQtQmFyYmFudDESMBAGA1U
EBwwJRWluZGhvdnVu%0AMRQwEgYDVQQKDAwTaW9leCBHcm91cDEMMAAoGA1UECwwDU0VTMR4wHAYJK
oZIhvcN%0AAQkBFg90ZXN0dWlAc2lvdXguZGUwGgEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK%0A
AoIBAQDI98ytSED47IFmjT0PUezLlyHULxXgYGF83G27J5%2BLkfn1xcWa2cCVH%0AotFG%2BK
ZNfspd5HAXUFRgJI%2BciypdMf3DS9ZH3PIScgEzwNXG0WnNUagHjUfsYQ6r%0A1I6MBKE86F8sjR
hyF%2FrlUpogsc24ALypBdIhQ6Ham3ni4NFsYEcBk80nKK%2BpATDJ%0APqzK1IIGsd3XhOmxjVUZ
PR7OfaEbHwnbOakWfeLibwJAWtttdIL7KcTSMSBLg3U2o%0AogYmm5fJqfLwZZEmwNslkuzGcHB6M1
WBYPQHyp966k1YnItBDFEMYKvaW2ec8tZEA%0AljGAWmIrrqAMR9obUWoUuGVwLoOXzAgMBAAGjCDBu
MAKGA1UdEwQCMAAwEwYDVR01%0ABAwwCgYIKwYBBQUHAIwCwYDVR0PBAQDAgP4MCCGCWCsSAGG%2
BEIBAgQAFhhDVVNU%0AX0FOT19JTLRFUK5BTF9URVNUVUkwFgYDVR0RBA8wDYILbnMuc2lvdXguZGU
UwDQYJ%0AKoZIhvcNAQELBQADggIBAIXrf8FkL2Bh2rW%2FgJQOpHADELqk25%2F8UmMNbpauneVB
%0AOaJFHCXpsSUDQ4beJo3gyM5JvZRV5nfC8Na3v4aNB0loVJHDHA8%2BGsl1UXy3fN1v%0AU4YAF
yiJULxELrYyuA5g0rdxXjLSSvSTWGWPKIJEGLYC2xv08kh%2BnwOijVciHFCp%0AGATpCbEO4MPwj
QiLpJowx5W5FhcyaRvp%2FjH10T1IsPWRuD23oG8gcg4uRAF9CKvU%0AYE8%2BV9RrYG%2B6VyeNP
2Va6DGTxVsH2%2Fi3vP7Ida08cnVOCYcHKzOBU%2Bfr60muNpuo%0AhE%2FoGzZ7uj%2F3wja9q49
OGJaNXHjadUjk5k799e%2Fv2isBaSuNmXVobOExzjasIyF%2B%0A2VWGrxSJcfhUXSv3%2B0xnm0o
UigN7uxVl%2BriPcpvlzyNLQqzNxSgerD0iW6dIAPHj%0AFuchgAYYuRXVMHbwxfqUVUWqyBg2Pr5
ESSYB6PTDXzigChLjWJRLuEZxWxmi6ztn%0AfgSMOG9XRFFR7yxQKDEuGZKfbAc0gXtCDbH3T%2B
UUESjAc7bUz40M5Zxfea%2F8F%2FE%0AoBrzb6XROZEWAPxmbQOWRXRu01ddl%2Fi77irHAsH%2FL
Gq6RGv7qsDmic%2B3WQXZtdkX%0As1TcBslGLZkgbnWevNtA8UWHL1JdfJjKmnOv5RdYxRc6kCXEO
WQilCBYQxpQXR5L%0A-----END%20CERTIFICATE-----" \
-X POST https://test.keytalk.com/public/1.6.1/self-service-availability
```

Response

HTTP 200 - application/json - successful invocation

```
{
  "status": "self-service-availability",
  "available": boolean
}
```

HTTP 400 - application/json - invalid request

```
{
  "status": "error",
  "error": error message (optional)
}
```

3.2.2 Retrieve address book URLs

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

Request

GET /public/1.6.1/address-book-list

Example:

/public/1.6.1/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",
  [as of v1.2.1] "apply-address-books": boolean flag indicating whether the caller
  should apply the address books, typically to be used by an email client. Applying the address books
  should allow the user to send encrypted emails and verify email signatures to other users registered to
  the address books,
  "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",
  [as of v1.2.1] "apply-address-books": see successful invocation,
  "address-books": ""
}
```

HTTP 400 - application/json - invalid request

```
{  
  "status": "error",  
  "error": error message (optional)  
}
```

3.2.3 *[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/1.6.1/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
<i>[as of v1.2.0]</i> synchronous	boolean	no	true	When set to true, checks whether an immediate enrolment is possible. When set to false checks whether it is possible to place a certificate order without yielding a certificate. The order will be handed in to a configured CA, which will get responsible for communicating the certificate back to the users via e-mail. At the moment S/MIME asynchronous orders are only supported by KeyTalk services bound to GlobalSign PersonalSign products.

Example:

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
-H "Expect:" \
-d "cert=-----BEGIN%20CERTIFICATE-----
%0AMIIFFDDCCAvSgAwIBAgIIWlSC7gAAAYowDQYJKoZIhvcNAQELBQAwwGgYDVRQQK%0ADBNLZXlUYWxrIElUI
BCQEWEGluZm9Aa2V5dGFsay5jb20xCzAJBgNVBAYTAk5MMRwwGgYDVRQQK%0ADBNLZXlUYWxrIElUI
FNlY3VyaXR5MRgwFgYDVRQLDA9GYWN0b3J5IERlZmF1bHJhZGQx%0AIDAEgNVBAMMF0tleVRhbGsgRGV
tbyBTAWduaW5nIENBMB4XDTE4MDEwOTA3NTMw%0AMl0XDTI4MDEwNzA4NTMwMlowZGZAxETAPBgNVB
AMMCERlbW9Vc2VyMQswCQYDVRQQG%0AEWJOTDEWMBQGA1UECAwNTm9vcmtQmFyYmFudDESMBAGA1U
EBwwJRWluZGhvdnVu%0AMRQwEgYDVRQQKDAAtaW91eCBHcm91cDEMAoGA1UECwwDU0VTMR4wHAYJK
oZIhvcN%0AAQkBFg90ZXN0dWlAc2lvdXguZXUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK%0A
AoIBAQDI98ytSED47IFmjT0PUezLlyHULxXUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK%0A
ZNFpsd5HAXUFRgJI%2BciypdMf3DS9ZH3PIScgEzwNXG0WnNUagHjUfsYQ6r%0A1I6MBKE86F8sjR
hyF%2FrlUpogsc24ALypBdIhQ6Ham3ni4NFsYEcBk80nKK%2BpATDJ%0APqzK1IIIGsd3XhOmXjVUZ
PR7OfaEbHwnbOakWfeLibwJAWtttIL7KctSMsBLg3U2o%0AogYmm5fJqfLwZZEmwNslkuzGcHB6M1
WBYPQHyp966k1YnItBDFEMYKvaW2ec8tZEA%0AljGAWmIrrqAMR9obUWoUuGVVLoOXzAgMBAAGjCDBu
MAkGA1UdEwQCMAAwEwYDVR01%0ABAwwCgYIKwYBBQUHAIwCwYDVR0PBAQDAgP4MCCGCWCSAGG%2
BEIBAgQaFhhDVVNU%0AXOFOT19JTlRFUk5BTF9URVNUVUkwFgYDVR0RBA8wDYILbnMuc2lvdXguZX
UwDQYJ%0AKoZIhvcNAQELBQADggIBAJXrf8FkL2Bh2rW%2FgJQOpHADELqk25%2F8UmMNbpauneVB
%0AOaJFHXCpsSUDQ4beJo3gyM5JvZRV5nfc8Na3v4aNB0loVJHDHA8%2BGs1lUxy3fN1v%0AU4YAF
yiJULxELrYyuA5g0rdxXjLSSvSTWGWPKIJEGLYC2xvO8kh%2BnwOijVciHFCp%0AGATpCbEO4MPwj
```

```
QiLpJowx5W5FhcyaRvp%2FjHl0T1IsPWRuD23oG8gCg4uRAF9CKvU%0AYE8%2BV9RrYG%2B6VYeNP
2Va6DGTXVsH2%2Fi3vP7Ida08cnVOcYcHKzOBU%2Bfr60muNpuo%0AhE%2FoGzZ7uj%2F3wja9q49
OGJaNxHjadUjk5k799e%2Fv2isBaSuNmXVobOExzjasIyF%2B%0A2VWGrxSJcfhUXSv3%2B0xnm0o
UigN7uxVl%2BriPcpvlzyNLQqzNxSgerD0iW6dIAPHj%0AFuchgAYYuRXVMHbwxfqUVUWqyBg2Pr5
ESSYB6PTDXzigChLjWJRLuEZXwXmi6ztn%0AfgSMOG9XRFFR7yxQKDEuGZKfbAc0gXtCDbH3T%2B
UUEsjAc7bUz4OM5Zxfea%2F8F%2FE%0AoBrzb6XROZEWAPxmbQOWRXRu01ddl%2Fi77irHAsH%2FL
Gq6RGv7qsDmic%2B3WQXZtdkX%0AslTcBslGLZkgnbWevNtA8UWHL1JdfJjKmnOv5RdYxRc6kCXEO
WQilCBYQxpQXR5L%0A-----END%20CERTIFICATE-----" \
-X POST https://test.keytalk.com/public/1.6.1/smime-cert-enrollment-
availability
```

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)
}
```

3.2.4 *[as of v1.3.0]* Query the certificate store to place the resulted certificate

Query the type of the certificate store to place the certificate received using the certificate retrieval API. The result will be identical as `store-to-system` flag returned from `/cert` call of the certificate retrieval API, however received without going through the entire handshake-authentication procedure.

Request

GET `/public/1.6.1/should-cert-go-to-system-store`

Example:

`/public/1.6.1/should-cert-go-to-system-store?service=DEMO_SERVICE`

Query parameters

parameter	type	required	description
service	string	yes	KeyTalk service name

Response

HTTP 200 - application/json - successful invocation

```
{
  "status": "should-cert-go-to-system-store",
  "system-store": boolean flag indicating whether the certificate should be
  placed in the caller's System (Machine) store ("true") or User store
  ("false")
}
```

HTTP 400 - application/json - invalid request

```
{
  "status": "error",
  "error": error message (optional)
}
```

3.2.5 *[as of v1.4.0]* Request a disclaimer to install to the mail client

Request an email disclaimer based on the provided email address for the provided service. If the service is setup to supply a disclaimer the response will contain a disclaimer to be applied to the mail client.

Request

GET /public/1.6.1/set-disclaimer-for-smime-cert-email

Example:

/public/1.6.1/set-disclaimer-for-smime-cert-email?service=DEMO_SERVICE&email=demo%40keytalk.com

Query parameters

parameter	type	required	description
service	string	yes	KeyTalk service name
email	string	yes	Email address for which to request a disclaimer

Response

HTTP 200 - application/json - successful invocation

```
{
  "status": "set-disclaimer-for-smime-cert-email",
  "set-disclaimer": flag determining whether a Disclaimer should be set to the mail client
  "disclaimer": {
    "name": String with the disclaimer name.
    "zip": Base64-encoded string containing the content of a mail client
signature compressed into a zip file.
[as of v1.5.0] "check-interval": Interval at which to check disclaimers in minutes.
  } : optional, must be set if set-disclaimer is true. Extend all mail client signatures with the
contents contained in zip
}}
```

HTTP 400 - application/json - invalid request e.g. the given service does not exist

```
{
  "status": "error",
  "error": error message (optional)
}
```

3.2.6 *[as of v1.5.0]* Query certificate approver emails

Query approver emails for the service. Typically used by services configured with 3rd CA, for example GlobalSign DomainSSL.

Request

GET /public/1.6.1/cert-approver-emails

Example:

/public/1.6.1/cert-approver-emails?service=DEMO_SERVICE&user=test-user&computer-name=test.keytalk.com

Query parameters

parameter	type	required	description
service	string	yes	KeyTalk service name
user	string	yes	User name. Should be the same value as the USERID used in RCDP authentication request.
computer-name	string	yes	Name of the caller's machine/device name. Should be the same as the computer name as used in RCDP authentication request.

Response

HTTP 200 - application/json - successful invocation

```
{
  "status": "cert-approver-emails",
  "emails": JSON array of emails
}
```

HTTP 400 - application/json - invalid request e.g. the given service does not exist is not configured with a CA implementing approval of cert requests via emails

```
{
  "status": "error",
  "error": error message (optional)
}
```

3.2.7 *[as of v1.6.0]* Query KeyTalk server version

Query KeyTalk server version

Request

GET /public/1.6.1/version

Query parameters

None

Response

HTTP 200 - application/json - successful invocation

```
{
  "status": "version",
  "version": KeyTalk server version string
}
```

3.2.8 *[as of v1.6.1]* Query Common Name customization policy

Query customization policy for Common Name for the given user

Request

GET /public/1.6.1/cn-customization-policy

Example:

/public/1.6.1/cn-customization-policy?service=DEMO_SERVICE&user=test-user&computer-name=test.keytalk.com

Query parameters

parameter	type	required	description
service	string	yes	KeyTalk service name
user	string	yes	User name. Should be the same value as the USERID used in RCDP authentication request.
computer-name	string	yes	Name of the caller's machine/device name. Should be the same as the computer name as used in RCDP authentication request.

Response

HTTP 200 - application/json - successful invocation

```
{
  "status": "cn-customization-policy",
  "policy": "ALLOWED"|"DISALLOWED-NOT-SUPPORTED-BY-SERVICE"|"DISALLOWED-
CERT-STILL-VALID"
}
```

HTTP 400 - application/json - invalid request e.g. the given service and user combination does not exist

```
{
  "status": "error",
  "error": error message (optional)
}
```

4. ADMINISTRATOR API

There is a set of API to be used by KeyTalk admins to perform the same (subset of) tasks as using KeyTalk Web Admin Interface.

4.1 API versions

REST API version	Minimal KeyTalk server version	Changes wrt the previous API version
1.0	5.6.13	Initial version having revocation API only.
1.1	5.7.0	Added certificate enrolment API.
1.2	5.8.13	Allow requesting KeyTalk settings
1.3	6.1.4	Allow requesting SCEP configuration
1.4	6.2.4	Allow copying KeyTalk TEMPLATE

4.2 API overview

The communication goes over HTTPS and uses port 3000 secured using KeyTalk internal CA or, when, enabled server-side, using a public trusted CA such as GlobalSign. All the API calls should be authenticated using credentials of the Web Admin Interface (username/password or a client certificate). Admin API requires a valid system admin/cluster admin/manager privileges to execute.

4.2.1 *[as of v1.1]* Enroll user from a server-generated keypair

Enroll a certificate for the given user, creating the user if not exist. The keypair is created server-side.

Request

```
POST /admapl/1.4.0/cert-enrollment
Content-type: application/x-www-form-urlencoded
```

Request POST parameters:

parameter	type	required	default value	description
auth-username	<i>string</i>	if required by the webserver	n/a	Caller's username. Required if the webserver is configured with username/password authentication.
auth-password	<i>string</i>	if required by the webserver	n/a	Caller's password. Required if the webserver is configured with username/password authentication.
service	<i>string</i>	yes	n/a	Name of KeyTalk service of the user to enroll
deviduser	<i>string</i>	yes	n/a	Name of KeyTalk DevID user to enroll
san	<i>JSON</i>	no	empty	Array of Subject Alternative Names to be used

array

in the certificate e.g.
["DNS:test.server.com",
"IP:192.168.1.2"]

Example (for username/password authentication):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
-d "auth-username=admin&auth-password=secret" \
-d "service=DEMO_SERVICE&deviduser=DemoUser" \
-X POST https://test.keytalk.com:3000/admapi/1.4.0/cert-enrollment
```

Example (for client certificate authentication):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
--cert ./client-cert.pem \
--key ./client-cert-key.pem \
-d "service=DEMO_SERVICE&deviduser=DemoUser" \
-X POST https://test.keytalk.com:3000/admapi/1.4.0/cert-enrollment
```

Response

HTTP 200 - application/json

```
{
  "status": "cert-enrollment",
  "cert": enrolled certificate and key in PEM format
  "created-user-auth-password": optional, authentication password of a newly created
  KeyTalk user provided the user being enrolled did not exist in an authentication back-end (only users
  of an internal db authentication backend can be automatically created this way)
}
```

HTTP 400 - application/json invalid request

HTTP 401 - application/json invalid credentials

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

```
{
  "status": "error",
  "error": error message (optional)
}
```

4.2.3 *[as of v1.1]* Enroll user with client CSR

An admin might want to generate a key pair by himself and submit the resulted CSR to KeyTalk server for signing. Enrolling a user with a client-side CSR consists of 3 steps: querying the KeyTalk server for parameters to use in the CSR, generation of the CSR and submitting the CSR to the KeyTalk server for signing.

4.2.3.1 Query CSR requirements for enrollment

Prior submitting a CSR to the KeyTalk server, an admin should query the server for the initial parameters for the CSR such as key size, signing algorithm, certificate subject and a subject alternative name (SAN).

Request

POST /admapi/1.4.0/csr-enrolment-requirements

Request POST parameters:

parameter	type	required	default value	description
auth-username	<i>string</i>	if required by the webserver	n/a	Caller's username. Required if the webserver is configured with username/password authentication.
auth-password	<i>string</i>	if required by the webserver	n/a	Caller's password. Required if the webserver is configured with username/password authentication.
service	<i>string</i>	yes	n/a	Name of KeyTalk service of the user to create a CSR for
deviduser	<i>string</i>	yes	n/a	Name of KeyTalk DevID user to create a CSR for.

Response

HTTP 200 - application/json

```
{
  "status": "csr-enrolment-requirements",
  "key-size": key size in bits,
  "signing-algo": algorithm to use for CSR signing,
  "subject": dictionary of subject fields to use in CSR,
  "san": if set, holds an array of Subject Alternative Names to use in CSR
}
```

Example:

```
{
  "status": "csr-requirements",
  "key-size": "2048",
  "signing-algo": "sha256",
  "subject": { "cn": "TestUser",
    "c": "NL",
    "st": "Utrecht",
    "l": "Amersfoort",
    "o": "KeyTalk",
    "ou": "Development",
    "e": "test@keytalk.com",
  },
  "san": ["email:test@keytalk.com","DNS:test.keytalk.com"]
}
```

Example (for username/password authentication):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
-d "auth-username=admin&auth-password=secret" \
-d "service=DEMO_SERVICE&deviduser=DemoUser" \
-X POST https://test.keytalk.com:3000/admapi/1.4.0/csr-enrolment-requirements
```

Example (for client certificate authentication):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
--cert ./client-cert.pem \
--key ./client-cert-key.pem \
-d "service=DEMO_SERVICE&deviduser=DemoUser" \
-X POST https://test.keytalk.com:3000/admapi/1.4.0/csr-enrolment-requirements
```

4.2.3.2 Enroll user with a client-generated CSR

Enroll a certificate for the given user, creating the user if not exist. The CSR is supplied by an admin and should obey the requirements retrieved with `csr-enrolment-requirements` call.

Request

```
POST /admapi/1.4.0/cert-enrollment-for-csr
Content-type: application/x-www-form-urlencoded
```

Request POST parameters:

parameter	type	required	default value	description
auth-username	<i>string</i>	if required by the webserver	n/a	Caller's username. Required if the webserver is configured with username/password authentication.

auth-password	<i>string</i>	if required by the webserver	n/a	Caller's password. Required if the webserver is configured with username/password authentication.
service	<i>string</i>	yes	n/a	Name of KeyTalk service of the user to enroll
deviduser	<i>string</i>	yes	n/a	Name of KeyTalk DevID user to enroll
csr	<i>string</i>	yes	n/a	Base64 encoded PKCS#10 certificate signing request

Example (for username/password authentication):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
-H "Expect:" \
-d "auth-username=admin&auth-password=secret" \
-d "service=DEMO_SERVICE&deviduser=DemoUser" \
-d "csr=-----BEGIN+CERTIFICATE+REQUEST-----
%0AMIIC1jCCAb4CAQAwgZAxCzAJBgNVBAYTAk5MMRlW EAYDVQQHDA1FaW5kaG92ZW4x%0ADDAKBgNV
VBAsMA1NFUzEUMBIGA1UECgwLU21vdXggR3JvdXAxFjAUBgNVBAGMDU5v%0Ab3JkLUJhcmJhbnQxE
TAPBgNVBAMMCERlbW9Vc2VyMR4wHAYJKoZIhvcNAQkBFg90%0AZXN0dW1Ac21vdXguZXUwggEiMA0
GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDG%0AfyCCkM7cbVhpBCSx1Nf%2BFDqa9banKf9sPRW
5VwBFYP5siLdsywnNqrFYcV0w6ss%0Ath21qK9bkjZoyiKpbzvzqQw08NlbBmJfj700O18HUn2xL
vp2z6J6q3Z4rAR4d8jx%0ApwcdRlPeJO5b3OtBaURKILaJTjtsUVyCXr%2B6u%2FgiuaD0DGBKsIQ
ccyAWGy%2BlzNer%0AsmUib%2FsnWHEaAPJtvg7T2amaWACKcqIOPPr%2BHDJUUNSYyju9xZqCLjx
6Y2%2B2ZXHK%0AMPfCfSP%2F8GCYgZ2%2FAIlWtsVzKSaRWmTVJfBsy50gW3YmwI0QYghl52NIDQu
BJeoT%0AmQFxsKXpqcWjpP3KTOS5AgMBAAgGADANBgkqhkiG9w0BAQsFAAOCAQEAbUVCaYm%2F%0A
w1otZaLgtCP2mIVVH%2FgHvTeVFs1436Lz%2FAKT5q1QRee81C2us1z9G7h3PG%2BM6w1N%0AUJau
wqQ2mR2c1VAidROdT52syNPR4jXeR11%2F7a%2FmsZFqaw3%2FLlWtBJHEfOA6apU%0AJSVWi6%2
F3kUjD0FhYHAufKm2nJl0qGnwC5xpzuvYOQsUFFobLZoyGq5NNEgnSpK8X%0A9A9j5kKGBom9eQOr
Wwx%2F0UlwRqLpt6l76Gt5%2BlMp5BtTCPK2uboHvJiPu4aJUuHh%0Afx9ZjKox73V%2BleOEmNSY
fesuQPE5AwifKe988NfixGXOHw7uQdWc9SfSYFRFZG2p%0AYb%2Bm9iFyUY8AHw%3D%3D%0A-----
END+CERTIFICATE+REQUEST-----%0A" \
-X POST https://test.keytalk.com:3000/admapi/1.4.0/cert-enrollment-for-csr
```

Example (for client certificate authentication):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
-H "Expect:" \
--cert ./client-cert.pem \
--key ./client-cert-key.pem \
-d "service=DEMO_SERVICE&deviduser=DemoUser" \
-d "csr=-----BEGIN+CERTIFICATE+REQUEST-----
%0AMIIC1jCCAb4CAQAwgZAxCzAJBgNVBAYTAk5MMRlW EAYDVQQHDA1FaW5kaG92ZW4x%0ADDAKBgNV
VBAsMA1NFUzEUMBIGA1UECgwLU21vdXggR3JvdXAxFjAUBgNVBAGMDU5v%0Ab3JkLUJhcmJhbnQxE
TAPBgNVBAMMCERlbW9Vc2VyMR4wHAYJKoZIhvcNAQkBFg90%0AZXN0dW1Ac21vdXguZXUwggEiMA0
GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDG%0AfyCCkM7cbVhpBCSx1Nf%2BFDqa9banKf9sPRW
5VwBFYP5siLdsywnNqrFYcV0w6ss%0Ath21qK9bkjZoyiKpbzvzqQw08NlbBmJfj700O18HUn2xL
vp2z6J6q3Z4rAR4d8jx%0ApwcdRlPeJO5b3OtBaURKILaJTjtsUVyCXr%2B6u%2FgiuaD0DGBKsIQ
ccyAWGy%2BlzNer%0AsmUib%2FsnWHEaAPJtvg7T2amaWACKcqIOPPr%2BHDJUUNSYyju9xZqCLjx
6Y2%2B2ZXHK%0AMPfCfSP%2F8GCYgZ2%2FAIlWtsVzKSaRWmTVJfBsy50gW3YmwI0QYghl52NIDQu
BJeoT%0AmQFxsKXpqcWjpP3KTOS5AgMBAAgGADANBgkqhkiG9w0BAQsFAAOCAQEAbUVCaYm%2F%0A
w1otZaLgtCP2mIVVH%2FgHvTeVFs1436Lz%2FAKT5q1QRee81C2us1z9G7h3PG%2BM6w1N%0AUJau
wqQ2mR2c1VAidROdT52syNPR4jXeR11%2F7a%2FmsZFqaw3%2FLlWtBJHEfOA6apU%0AJSVWi6%2
F3kUjD0FhYHAufKm2nJl0qGnwC5xpzuvYOQsUFFobLZoyGq5NNEgnSpK8X%0A9A9j5kKGBom9eQOr
Wwx%2F0UlwRqLpt6l76Gt5%2BlMp5BtTCPK2uboHvJiPu4aJUuHh%0Afx9ZjKox73V%2BleOEmNSY
fesuQPE5AwifKe988NfixGXOHw7uQdWc9SfSYFRFZG2p%0AYb%2Bm9iFyUY8AHw%3D%3D%0A-----
END+CERTIFICATE+REQUEST-----%0A" \
```

-X POST <https://test.keytalk.com:3000/admapi/1.4.0/cert-enrollment-for-csr>

Response

HTTP 200 - application/json

```
{
  "status": "cert-enrollment",
  "cert": enrolled certificate and key in PEM format
  "created-user-auth-password": optional, authentication password of a newly created
  KeyTalk user provided the user being enrolled did not exist in an authentication back-end (only users
  of an internal db authentication backend can be automatically created this way)
}
```

HTTP 400 - application/json invalid request

HTTP 401 - application/json invalid credentials

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

```
{
  "status": "error",
  "error": error message (optional)
}
```

4.2.4 Revoke user certificates

Revoke certificates of the given user. Requires a valid system/cluster/service admin or a service operator privileges to execute.

Request

POST /admapi/1.4.0/cert-revocation

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

Request POST parameters:

parameter	type	required	default value	description
auth-username	string	if required by the webserver	n/a	Caller's username. Required if the webserver is configured with username/password authentication.
auth-password	string	if required by the webserver	n/a	Caller's password. Required if the webserver is configured with username/password authentication.

service	string	yes	n/a	Name of KeyTalk service of the user whose certificates are to be revoked
deviduser	string	yes	n/a	Name of KeyTalk DevID user whose certificates are to be revoked

Example (for username/password authentication):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
-d "auth-username=admin&auth-password=secret" \
-d "service=DEMO_SERVICE&deviduser=DemoUser" \
-X POST https://test.keytalk.com:3000/admapi/1.4.0/cert-revocation
```

Example (for client certificate authentication):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
--cert ./client-cert.pem \
--key ./client-cert-key.pem \
-d "service=DEMO_SERVICE&deviduser=DemoUser" \
-X POST https://test.keytalk.com:3000/admapi/1.4.0/cert-revocation
```

Response

HTTP 200 - application/json

```
{
  "status": "cert-revocation",
  "num-revoked-certs": number of revoked certificates,
  "warning": warning message if applicable
}
```

HTTP 400 - application/json invalid request

HTTP 401 - application/json invalid credentials

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

```
{
  "status": "error",
  "error": error message (optional)
}
```

4.2.5 *[as of v1.2]* Download KeyTalk settings

Download KeyTalk settings. This is the counterpart of the saving settings under the System -> Settings page of KeyTalk Web Admin interface.

Request

POST /admapl/1.4.0/settings
Content-type: application/x-www-form-urlencoded

Request POST parameters:

parameter	type	required	default value	description
auth-username	<i>string</i>	if required by the server	n/a	Caller's username. Required if the webserver is configured with username/password authentication.
auth-password	<i>string</i>	if required by the server	n/a	Caller's password. Required if the webserver is configured with username/password authentication.
include-hsm-connection-settings	<i>boolean</i>	no	false	include HSM Connection Settings
include-keytalk-cert-tree	<i>boolean</i>	no	false	include KeyTalk Certificate Tree

Example (for username/password authentication):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
-d "auth-username=admin&auth-password=secret" \
-d "include-shared-settings=true&include-db-connection-settings=true" \
-X POST https://test.keytalk.com:3000/admapl/1.4.0/settings
```

Example (for client certificate authentication):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
--cert ./client-cert.pem \
--key ./client-cert-key.pem \
-d "include-shared-settings=true&include-db-connection-settings=true" \
-X POST https://test.keytalk.com:3000/admapl/1.4.0/settings
```

Response

HTTP 200 - application/octet-stream - settings is returned in HTTP response body

HTTP 400 - application/json invalid request

HTTP 401 - application/json invalid credentials

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

```
{  
  "status": "error",  
  "error": error message (optional)  
}
```

4.2.6 *[as of v1.3]* Retrieve SCEP configuration

Retrieve SCEP configuration. This is the counterpart of the Certificate and Keys -> SCEP page of KeyTalk Web Admin interface.

Request

POST /admapi/1.4.0/scep-config
Content-type: application/x-www-form-urlencoded

Request POST parameters:

parameter	type	required	default value	description
auth-username	<i>string</i>	if required by the server	n/a	Caller's username. Required if the webserver is configured with username/password authentication.
auth-password	<i>string</i>	if required by the server	n/a	Caller's password. Required if the webserver is configured with username/password authentication.

Example (for username/password authentication):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
-d "auth-username=admin&auth-password=secret" \
-X POST https://test.keytalk.com:3000/admapi/1.4.0/scep-config
```

Example (for client certificate authentication):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
--cert ./client-cert.pem \
--key ./client-cert-key.pem \
-X POST https://test.keytalk.com:3000/admapi/1.4.0/scep-config
```

Response

HTTP 200 - application/json

```
{
  "status": "scep-config",
  "enabled": Boolean flag indicating whether SCEP configuration is enabled. The properties below
are returned if "enabled" is true,
  "service-name": service name
  "recipient-cert": recipient certificate (PEM),
  "recipient-key": recipient key ( PKCS#8 PEM),
  "signing-cert": signing certificate (PEM),
  "signing-key": signing key (PKCS#8 PEM),
  "issuer-certs": concatenated issuer CA chain of the signing and recipient cert above (PEM),
}
```

HTTP 400 - application/json invalid request
HTTP 401 - application/json invalid credentials
HTTP 500 - application/json - generic server-side enrolment error

```
{  
  "status": "error",  
  "error": error message (optional)  
}
```

4.2.7 *[as of v1.4]* Copy KeyTalk TEMPLATE

Copy KeyTalk TEMPLATE (previously, SERVICE0 along with all their properties. The caller should supply a name of the new TEMPLATE and, optional, some extra TEMPLATE properties to be altered.

Request

POST /admapl/1.4.0/copy-template

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

Request POST parameters:

parameter	type	required	default value	description
auth-username	<i>string</i>	if required by the server	n/a	Caller's username. Required if the webserver is configured with username/password authentication.
auth-password	<i>string</i>	if required by the server	n/a	Caller's password. Required if the webserver is configured with username/password authentication.
src-template-name	<i>string</i>	yes	n/a	Name of the TEMPLATE to copy
new-template-name	<i>string</i>	yes	n/a	Name of the new TEMPLATE
digicert-central-settings	<i>JSON-serialized string</i>	if applicable	n/a	If the source TEMPLATE is configured with DigiCert Central CA source, an admin can alter some of these settings in the new TEMPLATE. <pre>{ "product": "ssl_basic" "ssl_ev_basic" "classl_smime" "client_premium" "ssl_securesite_flex" "ssl_ev_securesite_flex" "ssl_geotrust_truebizid" "ssl_thawte_webserver" "ssl_ev_thawte_webserver", "api-key": new API-key, "cert-validity-months": new certificate validity in months, "organization-id": new organization ID, if applicable, "approver-user-id": approver user ID, if applicable }</pre>

Example (for username/password authentication):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
-d "auth-username=admin&auth-password=secret&src-template-name=TEMPL&new-
template-name=TEMPL-NEW" \
-X POST https://test.keytalk.com:3000/admapi/1.4.0/copy-template
```

Example (for client certificate authentication):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
--cert ./client-cert.pem \
--key ./client-cert-key.pem \
-d "src-template-name=TEMPL&new-template-name=TEMPL-NEW" \
-X POST https://test.keytalk.com:3000/admapi/1.4.0/copy-template
```

Example (with custom DigiCert Central settings):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
-d "auth-username=admin&auth-password=secret" \
-d "src-template-name=TEMPL&new-template-name=TEMPL-NEW" \
-d "digicert-central-settings=%7B%22product%22%3A%22ssl_basic%22%2C%20%22api-
key%22%3A%22123456%22%2C%20%22cert-validity-
months%22%3A%2212%2C%20%22organization-id%22%3A%206543%7D" \
-X POST https://test.keytalk.com:3000/admapi/1.4.0/copy-template
```

Response

HTTP 200 - application/json

```
{
  "status": "success",
}
```

HTTP 400 - application/json invalid request, e.g. a new template with the given name already exists

HTTP 401 - application/json invalid credentials

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

```
{
  "status": "error",
  "error": error message (optional)
}
```

5. SELF-SERVICE API

There is a set of the API to be used by KeyTalk self-service user. The API requires client certificate and key as an authentication means. The Self-Service 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.

5.1 API versions

REST API version	Minimal KeyTalk server version	Changes wrt the previous API version
1.0.0	5.5.0	n/a
1.1.0	5.5.1	added <code>synchronous</code> flag to <code>smime-cert-enrollment</code> call
1.1.1	5.5.3	added <code>apply-address-books</code> flag to <code>smime-cert-enrollment</code> call

5.2 API overview

The communication goes over HTTPS and uses port 3000. All the Self-Service 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.

5.2.1 Enroll S/MIME certificates for external parties

Enroll or place orders for S/MIME certificates for external parties i.e. to users that are generally not registered at KeyTalk. The enrolled certificates will be communicated to the indicated email addresses. It is strongly recommended to call `smime-cert-enrollment-availability` from the [Public API](#) before enrolling a certificate to get more detailed diagnostics when the enrolment is not possible and minimize the chance of errors during enrolment.

Request

```
POST /ssapi/1.1.1/smime-cert-enrollment
Content-type: application/x-www-form-urlencoded
```

Request POST parameters:

parameter	type	required	default value	description
<code>recipients</code>	<i>JSON object</i>	yes	n/a	JSON array of recipients. Each recipient contains at least an email address to send the download link of the generated S/MIME certificate to. A recipient might also include mobile phone number to receive the password for the S/MIME certificate and key. The mobile number is only required when it is enforced by the server

configuration otherwise it is optional. If mobile phone is neither enforced by the server nor supplied by the caller the password will be included in the email.

Example:

```
[
  { "email": "mike.brook@example.com",
    "mobile": "+31645610000"
  },
  { "email": "chuck.norris@badass.com"
  }
]
```

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.
<i>[as of v1.1.0]</i> synchronous	boolean	no	true	When set to false, the function does not immediately yield a certificate. Instead KeyTalk will submit a request to a configured CA, which gets responsible for communicating the certificate back to the users via e-mail, normally after asking them for approval. At the moment S/MIME asynchronous orders are only supported by KeyTalk services bound to GlobalSign PersonalSign products.

Example (for the recipients above):

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
--cert ./client-cert.pem \
--key ./client-cert-key.pem \
-d \
"recipients=%5B%7B%22email%22%3A%22mike.brook%40example.com%22%2C%22mobile%22%3A%22%2B31645610000%22%7D%2C%7B%22email%22%3A%22chuck.norris%40badass.com%22%7D%5D" \
-X POST https://test.keytalk.com:3000/ssapi/1.1.1/smime-cert-enrollment
```

Response

HTTP 200 - application/json - when synchronous enrolment requested and it succeeded for at least one recipient.

```
{
  "status": "smime-cert-enrollment",
  "enrolled-recipients": [ emails ],
```

```

    "failed-recipients": [
        {
            "email": email,
            "error": error message
        },
        ...
    ],
    "skipped-recipients": [ emails ],

    [as of v1.1.1] "apply-address-books": boolean flag indicating whether the caller
    should apply the address books, typically to be used by an email client. Applying the address books
    should allow the user to send encrypted emails and verify email signatures to other users registered to
    the address books,

    "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),
        },
        ...
    ]
}

```

When multiple recipients are supplied, the enrolment might succeed for some of them but fail along the way. If this happens the enrolment process terminates and the remaining recipients get skipped.

HTTP 200 - application/json – when asynchronous enrolment requested and at least one order has been placed successfully placed.

```

{
    "status": "smime-cert-enrollment",

    "placed-orders": [
        {
            "email": email,
            "order_id": order id
        },
        ...
    ],
    "failed-orders": [
        {
            "email": email,
            "error": error message
        },
        ...
    ],
    "skipped-orders": [ emails ],

    [as of v1.1.1] "apply-address-books": see synchronous response,

    "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),
        },
        ...
    ]
}

```

When multiple recipients are supplied, the order placement might succeed for some of them but fail along the way. If this happens the enrolment process terminates and the remaining recipients get skipped.

HTTP 400 - application/json - invalid request detected before enrolling any recipients (e.g. invalid mobile phone number supplied or SMTP not configured for the given KeyTalk service)

```

{
  "status": "error",
  "error": error message (optional)
}

```

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

```

{
  "status": "error",
  "error": error message (optional)
}

```

6. CERTIFICATE AUTHORITY RETRIEVAL API (CA API)

Certificate Authority API is meant for fetching trusted and intermediate certificate authorities used by KeyTalk and is primary used by KeyTalk agents in order to roll out the initial trust CAs on the system before the RCDP API comes into play.

The calls go over plain HTTP and, when configured on the server, also by HTTPS secured by a public trusted CA such as GlobalSign.

6.1 CA API versions

REST API version	Minimal KeyTalk server version	Changes wrt the previous API version
1.0.0	5.3.1	n/a
1.0.1	5.8.0	Allow caller requesting a desired certificate by fingerprint Allow caller specifying a desired certificate download format
1.0.2	6.2.1	Allow requesting extra Signing CAs

6.2 CA API overview

The non-SSL HTTP communication goes over port 8000. HTTPS communication, provided enabled server-side, goes over port 8443 secured by known trusted CA.

6.2.1 Fetch internal signing CA

Fetch 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 QuoVadis CA tree) corresponding to “Signing CA”, “Extra Signing CA”, “Communication CA”, “Primary CA” and “Root CA” of the KeyTak admin web panel. A typical KeyTalk internal CA tree is 2 level deep with a self-signed Primary CA and no Root CA.

Request

```
GET /ca/1.0.2/signing[?PEM|DER]
GET /ca/1.0.2/primary[?PEM|DER]
GET /ca/1.0.2/communication[?PEM|DER]
GET /ca/1.0.2/root[?PEM|DER]
GET /ca/1.0.2/extrasigning
GET /ca/1.0.2/signing/<cert-shal-fingerprint>[?PEM|DER]
GET /ca/1.0.2/primary/<cert-shal-fingerprint>[?PEM|DER]
GET /ca/1.0.2/communication/<cert-shal-fingerprint>[?PEM|DER]
GET /ca/1.0.2/root/<cert-shal-fingerprint>[?PEM|DER]
GET /ca/1.0.2/extrasigning/<cert-shal-fingerprint>[?PEM|DER]
```

Response



HTTP 200 - application/octet-stream - CA certificate is returned in HTTP response body. Note that the /extrasigning request will yield a list of extra signing CAs configured on KeyTalk concatenated in a single PEM file.

HTTP 404 - CA does not exist

7. OUTBOUND CERTIFICATE MANAGEMENT API

KeyTalk exposes a set of certificate provisioning API for application integrators. Besides being a certificate provider KeyTalk can also work as a certificate consumer by requesting certificates from third-party CA providers. There are two options for KeyTalk to implement that. First, KeyTalk can use a pre-defined certificate management API already supplied by CA providers. Another option is when KeyTalk supplies a CA service provider with a specification of the certificate management API it needs to implement. KeyTalk will then make use of this API to request certificates from this CA provider.

This section outlines the Certificate Management API to be implemented by CA providers in order to provision their certificate services to KeyTalk.

7.1 Obtaining API credentials

This section outlines the process of obtaining credentials to be used by KeyTalk to access the Certificate Management API.

1. To access the API endpoint, KeyTalk development team needs to generate a key pair.

```
# openssl genrsa -out key.pem 4096
```

2. Extract the public key from the generated key pair and send it to the CA provider representative.

```
# openssl rsa -in key.pem -outform PEM -pubout -out public.pem
```

3. The CA provider will send back an API Signing Certificate by signing the above public key along with API Access Key encrypted with that public key.

4. The KeyTalk development team will need to use their private key to decrypt the received Access Key.

```
# openssl rsautl -decrypt -inkey privatekey.pem -in apikey.enc -out  
apikey.txt
```

5. The KeyTalk development team will need to use the API Signing Certificate along with the decrypted API Access Key to access the API.

- the Signing Certificate is used to establish a mutually-trusted 2SSL connection to the server functioning as an initial, usually a coarse-grained filter for incoming API requests.
- the Access Key sent along with a request payload is used for fine-grained authentication and authorization of the API request.

7.2 Certificate Provisioning API

7.2.1 Request a new certificate

This method allows the calling client to request a new certificate given a CSR and extra certificate attributes. Any certificate attributes specified as parameters will overwrite their counterparts in the CSR. The best practice to streamline mass enrollment of certificates is to make multiple asynchronous calls by setting `synchronous` parameter to `false` and then polling and eventually retrieving the result using the query API.

Request

POST /<api-path>/<api-version>/new-cert

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

Request POST parameters:

parameter	type	required	default value	description
api-key	string	yes	n/a	API key received on the credentials establishment step
product-id	string	yes	empty	CA provider-specific identifier of a product associated with the requested certificate. For example "SMIME-CERT" or "EV-CERT"
CSR	string	yes	n/a	Base64 encoded PKCS#10 certificate signing request.
validity-not-before	string	no	current date/time	Certificate validity start datetime as UTC in ISO 8601 including date and time. For example, "2020-04-14T13:15:30+0000"
validity-duration	string	no	defined by CA provider	Certificate validity duration specified either as a number seconds followed by a 's' symbol or as number of days followed by a 'd' symbol. For example both "604800s" and "7d" represent the same duration of 7 days. When not supplied defaults to the default validity defined by the CA provider.
subject_dn	JSON object	no	subject attributes defined in CSR	Distinguished Name attributes to include in the certificate. See RFC 5280 section 4.1.2.6. Any attribute specified in <code>subject_dn</code> takes precedence over its counterpart defined in <code>CSR</code> parameter.

Supported DN attributes are:

- CN: (string) - common name
- C: (string) - ISO 3166-1 alpha-2 two-letter country code
- S (string) - state or province
- L (string) - locality
- O: (string) - organization
- OU: (array of string) -

				organizational unit - E: (<i>string</i>) - email - extra-attributes: (<i>JSON object</i>) - extra subject DN attributes specified by OID and value. Example: <pre>{ "CN": "John Wood", "C": "NL", "S": "Utrecht", "L": "Amersfoort ", "O": "KeyTalk", "OU": ["Operations", "Development"], "E": "john.wood@nospam.com", "extra-attributes": { "2.5.4.4": "van Bommel" } }</pre>
san	<i>JSON object</i>	no	SAN attributes defined in CSR	List of Subject Alternative Name attributes to include in the certificate. See RFC 5280 section 4.2.1.6. Any attribute specified in <code>san</code> takes precedence over its counterpart defined in CSR parameter. Supported SAN attributes are: - DNS: (array of string) - Email: (array of string) - IP: (array of string) - URI: (array of string) Example: <pre>{ "DNS": ["test1.example.com", "test2.example.com "], "IP": ["198.41.33.123"], "Email": ["admin@example.com"] }</pre>
synchronous	<i>boolean</i>	no	true	Boolean flag indicating whether the API call will block until the certificate is issued (<code>true</code>) or will immediately return an order-id (<code>false</code>) to be used later on to retrieve the certificate once issued using query API functions.
include-chain	<i>boolean</i>	no	false	Boolean flag indicating whether the returned certificate should also include CA trust chain certificates. Only applies when <code>synchronous</code> is

set to true.

Example:

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
-H "Expect:" \
--cert ./api-signing-cert.pem \
--key ./api-signing-key.pem \
-d "api-key=012345689ABCDEF012345689ABCDEF"
-d "product-id= SMIME-CERT"
-d "csr=-----BEGIN+CERTIFICATE+REQUEST-----
%0AMIIC1jCCAb4CAQAwgZAxCzAJBgNVBAYTAk5MMRlWEAYDVQQHDA1FaW5kaG92ZW4x%0ADDAKBgN
VBAsMA1NFUzEUMBIGA1UECgwLU21vdXggR3JvdXAxFjAUBgNVBAGMDU5v%0Ab3JkLUJhcmJhbnQxE
TAPBgNVBAMMCERlbW9Vc2VyMR4wHAYJKoZIhvcNAQkBFg90%0AZXN0dW1Ac21vdXguZXUwgGEiMA0
GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDG%0AfyCCkM7cbVhpBCSx1Nf%2BFDqa9banKf9sPRW
5VwBFYP5siLdsywnNqrFYcV0w6ss%0Ath21qK9bkjZoyiKpbzvzqW08N1bBmJfj700O18HUn2xL
vp2z6J6q3Z4rAR4d8jx%0ApwcdR1PeJO5b3OtBaURKILaJTjtsUVyCXr%2B6u%2FgiuaD0DGBKsIQ
ccyAWGy%2B1zNer%0AsmUib%2FsnWHEaAPJtvG7T2amaWACKcqIOppR%2BHDJUUNSYyju9xZqCLjx
6Y2%2B2ZXHK%0AMpFcFsP%2F8GCYGGZ2%2FAi1WtsVzKSaRWmTVJfBsy50gW3YmwI0QYgh152NIDQu
BJeoT%0AmQFxsKXpqcWjpP3KTOS5AgMBAAgGADANBgkqhkiG9w0BAQsFAAOCAQEABUVCaYm%2F%0A
w1otZaLgtCP2mIVVH%2FgHvTeVFs1436Lz%2FaKT5q1QRee81C2us1z9G7h3PG%2BM6w1N%0AUJau
wqQ2mR2c1VAidROdT52syNPR4jXeR11%2F7a%2FmsZFqaw3%2FLlwVtBJHEfOA6apU%0AJSVWi6%2
F3kUjD0FhYHAufKm2nJ10qGnwC5xpzuYvYOQsUFFobLZoyGq5NNEgnSpK8X%0A9A9j5kKGBOm9eQOr
Wxw%2F0U1wRqLpt6l76Gt5%2B1Mp5BtTCPK2uboHvJiPu4aJUuHh%0Afx9ZjKox73V%2BleOEmNSY
fesuQPE5AwIFkE988NFixGXOHw7uQdWc9SFsYFRFZG2p%0AYb%2Bm9iFyUY8AHw%3D%3D%0A-----
END+CERTIFICATE+REQUEST-----%0A" \
-d
"san=%7B%0A%20%20%22DNS%22%3A%20%5B%0A%20%20%20%20%22test1.example.com%22%2C%
0A%20%20%20%20%22test2.example.com%20%22%0A%20%20%5D%2C%0A%7D%0A" \
-X POST https://example-ca-provider.com/api/1.0.0/new-cert
```

Responses

HTTP status code 200

The certificate has been issued (synchronous call).

Media type: application/json

```
{
  "order-id": CA provider-specific order ID associated with the given issued certificate,
  "cert": PEM-encode certificate,
  "warning": warning message, if applicable
}
```

Example:

```
{
  "order-id": "PROV001200117099231",
  "cert": "-----BEGIN CERTIFICATE-----
\nMIIFGTCCAwGgAwIBAgIIWurOaAAAABYwdQYJKoZIhvcNAQELBQAwYg9xHzAdBgkq\nnhkiG9w0B
CQEWEGluZm9Aa2V5dGFsay5jb20xCzAJBgNVBAYTAk5MMRwwGgYDVQQK\nnDBNLZXL1UYWxrIElUUF
NlY3VyaXR5MRgwFgYDVQQLDA9GYWN0b3J5IERlZmF1bHQx\nnIDAeBgNVBAMMF0tleVRhbGsgRGVt
byBTAwduaW5nIENBMB4XDTE4MDUwMzA3NTUw\nnNFoXDTE4MDUwMzA5NTUwNFowZAXETAPBgNVBA
MMCERlbW9Vc2VyMQswCQYDVQQG\nnEwJOTDEWMBQGA1UECAwNTm9vcmtQmFyYmFudDESMBAGA1UE
BwwJRWluZGhvdnVu\nnMRQWEgYDVQQKDAtTaW9leCBHcm91cDEMMAoGA1UECwwDU0VTMR4wHAYJKo
ZIhvcN\nnAQkBFg90ZXN0dW1Ac21vdXguZXUwgGEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK\nnAo
```

```
IBAQDJGKTHSL16vsqxIjXvDOTKLk2q518JaIF9Q9ews88NmpVV9cDbOPRxsns\nSd1kNAXEYi05
ScmIc5pGpIV8hyNjtZ17tiolV00ALkXgk7hG7wO2Rz+bAQzCdvS\noJjtzo6gZPYcQVlfq+ENMt
39ibLqfuAnMLjVpn44fwfqxQFeEsd4do074ElbUXh7\n7KzaoxsiDAYIITYZe5Azz90147ffg3pR
Dtq\6IDYmr7xlBMOq+7QObKBu0pgwNkn\n3JTgkBspXGEXok6S1qNBqJ199NjJdYjiWjHa\9vS
pHSN8RF2s9xrBanLM3S+fnr6\nBx34P6cBoTcc1lZ9Dpr8IYNJWkanAgMBAAGjfTB7MAkGA1UdEw
QCMAAwHQYDVR01\nBBYwFAYIKwYBBQUHAWIGCCsGAQUFBwMBMAsGA1UdDwQEAwID+DAQBglghkgB
hvhC\nAQIEHRYbQ1VTVF9QQVNTV0RfSU5URVJOUXxfVEVTVFVJMBYGA1UdEQQPMAC2CC25z\nLnNp
b3V4LmV1MA0GCSqGSIb3DQEBBCwUAA4ICAQCYKF1OTJqL3eg1JgJdbLPzDo74\nnfqZbEBpNkeBF6
nQ6calHJrZNG857WGdfVKfXSorkwGHmdSN1\0XM+ySIpcNOWQf\nmM9o9rxKQigk4n\0tvjNCiVX
Ra125t5pUR1ZSyul1SWQAJYc2nPjzas15B8SwJOIet\nJV80z1pgLFh2GU7hGNiWVqJLF\U0\0t
+xZ1lWlsZ64iih49owTsLt9CL06pD6KPN6\nWvmzLNoK\0ouEeRnYgkyWXvlahGY5N2bPwlq+7+s
3BOYRo3APL4N6iVEOUfYDE78K\n05g5zdhVbn717CMx1sQpXggyF5X\0ztQLkrUB5kLT9D7eCBnL
DVdjELz112KJar\0b\nny9eumkCg+Y9PCZN2513o1zU1DLGaH9\09KdCf6yEca3D3NvnbfCmrDvx1
0AN+Ht3L\n4XU2L5Rx2rqwB9tj3rZy8i6BK7\0A+ARfg6Tqki5FQ9k667q2hBRPtr69bLeML5at\
nyn\0beKjnYnzCRcfXDgnJIKZdfKt2PBM7lh508HNn6aaRZUfHBKHxjMxwXNMdq9m\nnHk6+H8rb
RipV\04xCzEFYvaqlpYO3lOzLIrw8AohR1UzX7UFGm1Dbpn3G2qeikD1Z\nhySYTxjmjXE0DVnPL
X05+MR08Eq3hC6QDys3gBZgP3nILvfEZliOax4fqbt3ijJ9\nnoxMI+OJsawZMG0u00w==\n-----
END CERTIFICATE-----\n"
}
```

HTTP status code 201

The certificate issuance request has been accepted (asynchronous call)

The "Location" HTTP header holds an order-id associated with the certificate being created.

HTTP status code 400

Invalid request e.g. non-existing order id

Media type: application/json

```
{
  "code": CA provider-specific error code,
  "description": error description
}
```

HTTP status code 500

System cannot process the request

Media type: application/json

```
{
  "code": CA provider-specific error code (optional),
  "description": error description (optional)
}
```

7.2.2 Renew a certificate

This method allows the calling client to renew a certificate a previously issued certificate given its order id. Notice that it is not possible to alter subject or SAN fields of the certificate being renewed.

Technically renewal does not differ from creating a new certificate with the same attributes, yet it might be more beneficial for the caller from the financial point of view.

The best practice to streamline mass renewal of certificates is to make multiple asynchronous calls by setting `synchronous` parameter to `false` and then polling and eventually retrieving the result using the query API.

Request

POST /<api-path>/<api-version>/cert-renewal
Content-type: application/x-www-form-urlencoded

Request POST parameters:

parameter	type	required	default value	description
api-key	string	yes	n/a	API key received on the credentials establishment step
order-id	string	yes	n/a	Order id referencing a certificate, previously issued using the new certificate creation call.
CSR	string	yes	n/a	Base64 encoded PKCS#10 certificate signing request. Only the public key from the CSR is honoured.
validity-not-before	string	no	current date/time	Certificate validity start datetime as UTC in ISO 8601 including date and time. For example "2020-04-14T13:15:30+0000"
validity-duration	string	no	defined by CA provider	Certificate validity duration specified either as a number seconds followed by a 's' symbol or as number of days followed by a 'd' symbol. For example both "604800s" and "7d" represent the same duration of 7 days. When not supplied defaults to the default validity defined by the CA provider.
synchronous	boolean	no	true	Boolean flag indicating whether the API call will block until the certificate is issued (<code>true</code>) or will immediately return an order-id (<code>false</code>) to be used later on to retrieve the certificate once issued using query API functions.
include-chain	boolean	no	false	Boolean flag indicating whether the returned certificate should also include CA trust chain certificates. Only applies when <code>synchronous</code> is set to <code>true</code> .

Example:

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

```
--cert ./api-signing-cert.pem \
--key ./api-signing-key.pem \
-d "api-key=012345689ABCDEF012345689ABCDEF"
-d "order-id=PROV001200117099231" \
-d "csr=-----BEGIN+CERTIFICATE+REQUEST-----
%0AMIIC1jCCAb4CAQAwZAxCAJBGNVBAYTAk5MMRIwEAYDVQQHDA1FaW5kaG92ZW4x%0ADDAKBgN
VBAsMA1NFUzEUMBIGA1UECgwLU21vdXggR3JvdXAxFjAUBGNVBAGMDU5v%0Ab3JkLUJhcmJhbnQxE
TAPBgNVBAMMCERlbW9Vc2VyMR4wHAYJKoZIhvcNAQkBFg90%0AZXN0dWlAc21vdXguZXUwgGEiMA0
GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDG%0AfyCCkM7cbVhpBCSx1Nf%2BFDqa9banKf9sPRW
5VwBFYP5siLdsywnNqrFYcV0w6ss%0Ath21qK9bkjZoyiKpbzvzqQw08NlbBmJfj700O18HUn2xL
vp2z6J6q3Z4rAR4d8jx%0ApwcdRlPeJO5b3OtBaURKILaJTjtsUVyCXr%2B6u%2FgiuaD0DGBKsIQ
ccyAWGy%2B1zNer%0AsmUib%2FsnWHEaAPJtvG7T2amaWACKcqIOppR%2BHDJUUNSYyju9xZqCLjx
6Y2%2B2ZXHK%0AMPFcFsP%2F8GCYGG2%2FAi1WtsVzKSaRWmTVJfBsy50gW3YmwIOQYgh152NIDQu
BJeoT%0AmQFxsKXpqcWjpP3KTOS5AgMBAAGgADANBgkqhkiG9w0BAQsFAAOCAQEAbUVCaYm%2F%0A
wlotZaLgtCP2mIVVH%2FgHvTeVFs1436Lz%2FaKT5q1QRee81C2us1z9G7h3PG%2BM6w1N%0AUJau
wqQ2mR2c1VAidROdT52syNPR4jXeR11%2F7a%2FmsZFqaw3%2FLlwVtBJHEfOA6apU%0AjsVWvi6%2
F3kUjd0FhYHAufKm2nJl0qGnwC5xpzuvYOQsUFFobLZoyGq5NNEgnSpK8X%0A9A9j5kKGBom9eQOr
Wxxw%2F0UlwRqLpt6l76Gt5%2B1Mp5BtTCPK2uboHvJiPu4aJUuHh%0Afx9ZjKox73V%2BleOEmNSY
fesuQPE5AwifKE988NFixGXOHw7uQdWc9SFsYFRFZG2p%0AYb%2Bm9iFyUY8AHw%3D%3D%0A-----
END+CERTIFICATE+REQUEST-----%0A" \
-X POST https://example-ca-provider.com/api/1.0.0/cert-renewal
```

Responses

See responses of [new certificate creation](#). Please note that the order id returned after renewal will differ from the order id of the certificate being renewed.

7.2.3 Query a certificate

This method allows the calling client to query a status of certificates and to retrieve issued certificates.

Request

POST /<api-path>/<api-version>/cert-query
Content-type: application/x-www-form-urlencoded

Request POST parameters:

parameter	type	required	default value	description
api-key	string	yes	n/a	API key received on the credentials establishment step
order-id	string	yes	n/a	Order id referencing a certificate, previously requested or issued using the new certificate creation or the certificate renewal call.
include-cert	boolean	no	false	Boolean flag indicating whether, besides sending back status, also send the for certificate provided it has ISSUED status.
include-chain	boolean	no	false	Boolean flag indicating whether, besides sending back status, also send a CA trust chain of the certificate provided it has ISSUED status.

Example:

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
--cert ./api-signing-cert.pem \
--key ./api-signing-key.pem \
-d "api-key=012345689ABCDEF012345689ABCDEF"
-d "order-id=PROV001200117099231" \
-X POST https://example-ca-provider.com/api/1.0.0/cert-query
```

Responses

HTTP status code 200

Media type: application/json

```
{
  "cert-status": one of "REQUESTED", "ISSUED", "REVOKED". More statuses might be
  added later on,
  "product-id": product id associated with this order
  "last-updated": last status update UTC as ISO 8601 including date and time,
  "renewed-from-order-id": order ID of a certificate used to renew the given certificate, if
  applicable
}
```

Example response for status-only query:

```
{
  "cert-status": "ISSUED",
  "product-id": "SMIME-CERT",
  "last-updated": "2020-04-14T16:30:59+0000",
  "renewed-from-order-id": "PROV001200117099230",
}
```

HTTP status code 400

Invalid request e.g. non-existing order id

Media type: application/json

```
{
  "code": CA provider-specific error code,
  "description": error description
}
```

HTTP status code 500

System cannot process the request

Media type: application/json

```
{
  "code": CA provider-specific error code (optional),
  "description": error description (optional)
}
```

7.2.4 Revoke a certificate

This method allows the calling client to revoke a certificate.

Request

POST /<api-path>/<api-version>/cert-revocation
Content-type: application/x-www-form-urlencoded

Request POST parameters:

parameter	type	required	default value	description
api-key	string	yes	n/a	API key received on the credentials establishment step
order-id	string	yes	n/a	Order id referencing a certificate issued using the new certificate creation or the certificate renewal call.

Example:

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
--cert ./api-signing-cert.pem \
--key ./api-signing-key.pem \
-d "api-key=012345689ABCDEF012345689ABCDEF" \
-d "order-id=PROV001200117099231" \
-X POST https://example-ca-provider.com/api/1.0.0/cert-revocation
```

Responses

HTTP status code 200

Media type: application/json

```
{
  "warning": warning message, if applicable (e.g. the certificate is already revoked)
}
```

HTTP status code 400

Invalid request e.g. non-existing order id

Media type: application/json

```
{
  "code": CA provider-specific error code,
  "description": error description
}
```

HTTP status code 500

System cannot process the request

Media type: application/json

```
{  
  "code": CA provider-specific error code (optional),  
  "description": error description (optional)  
}
```

7.2.5 Query an account information

This method allows the calling client to query information associated with his account such as the remaining certificate issuance quota or the remaining funds.

Request

POST /<api-path>/<api-version>/account-info
Content-type: application/x-www-form-urlencoded

Request POST parameters:

parameter	type	required	default value	description
api-key	string	yes	n/a	API key received on the credentials establishment step

Example:

```
$ curl -H "Content-Type: application/x-www-form-urlencoded" \
--cert ./api-signing-cert.pem \
--key ./api-signing-key.pem \
-d "api-key=012345689ABCDEF012345689ABCDEF"
-X POST https://example-ca-provider.com/api/1.0.0/account-info
```

Responses

HTTP status code 200

Media type: application/json

```
[
  {
    "product-id": CA provider-specific identifier of a product e.g. "SMIME-CERT" or "EV-CERT",
    "remaining-cert-quota": (integer) remaining certificate issuance quota for the given product of the calling account,
    "remaining-funds": (string), if supported by CA provider, remaining funds the given product of the calling account e.g. "EUR1000"
  },
  ...
]
```

HTTP status code 400

Invalid request e.g. non-existing order id

Media type: application/json

```
{
  "code": CA provider-specific error code,
  "description": error description
}
```

HTTP status code 500

System cannot process the request

Media type: application/json

```
{  
  "code": CA provider-specific error code (optional),  
  "description": error description (optional)  
}
```