KeyTalk - Protocols

Date 11-04-2018

Sioux Embedded Systems B.V.
Esp 405
5633 AJ Eindhoven
The Netherlands
Tel: +31 (0)40 2677100
Fax: +31 (0)40 2677101
www.sioux.nl

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

The purpose of this document is to describe the protocols used by the KeyTalk system. This document is the leading source for these protocols.

1.2 Scope

This document is intended for TrustAlert and all Sioux KeyTalk team members.

1.3 Definitions and abbreviations

1.3.1 Definitions

1.3.2 Abbreviations

RDD	: **R**ESEPT **D**ispatcher **D**aemon
RCDP	: **R**ESEPT **C**lient <-> RESEPT **D**ispatcher Daemon **P**rotocol
RESEPT	: The historical name of KeyTalk software

2. RCDP V2

This section describes RCDP protocol version 2. The motivation to develop a new protocol over the existing RCDPv1 is as follows:

- Offload handcrafted security handshake to a standard SSL/TLS stack implemented by HTTPS protocol
- Use RESEful way of communication based on simple HTTP GET requests and JSON responses

These changes ought to significantly simplify the protocol, make it easier to test and develop clients without diving into communication security details.

2.1 RCDPv2 versions

RCDP version	Supported KeyTalk server	Changes wrt the previous RCDP version
`2.0.0`	`5.2.0` and up	
`2.1.0`	`5.3.0` and up	Added a possibility for the caller to request a certificate download URL in the phase 3 `cert` request instead of a certificate body.

2.2 RCDPv2 overview

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

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

HTTP Header	Required	Description
GET	YES	`/rcdp/2.X.Y/<action> ?<request-params>`
Host	YES	Should contain the FQDN or IP (v4 or v6) of the 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 `'\/'`.

For example a relevant set of client headers could be:

```
GET
/rcdp/2.1.0/authentication?service=DEMO_SERVICE&PASSWD=change%21&HWSIG=12345
6&USERID=DemoUser &ips=%5B%2281.175.103.107%22%5D&caller-hw-
description=Windows+7%2C+BIOS+s%2Fn+1234567890 HTTP/1.1
Host: keytalkdemo.keytalk.com
Accept-Encoding: identity
Cookie: keytalkcookie=a622bb821bec1f5315668c8f9a8e780f
```

A relevant set of response headers:

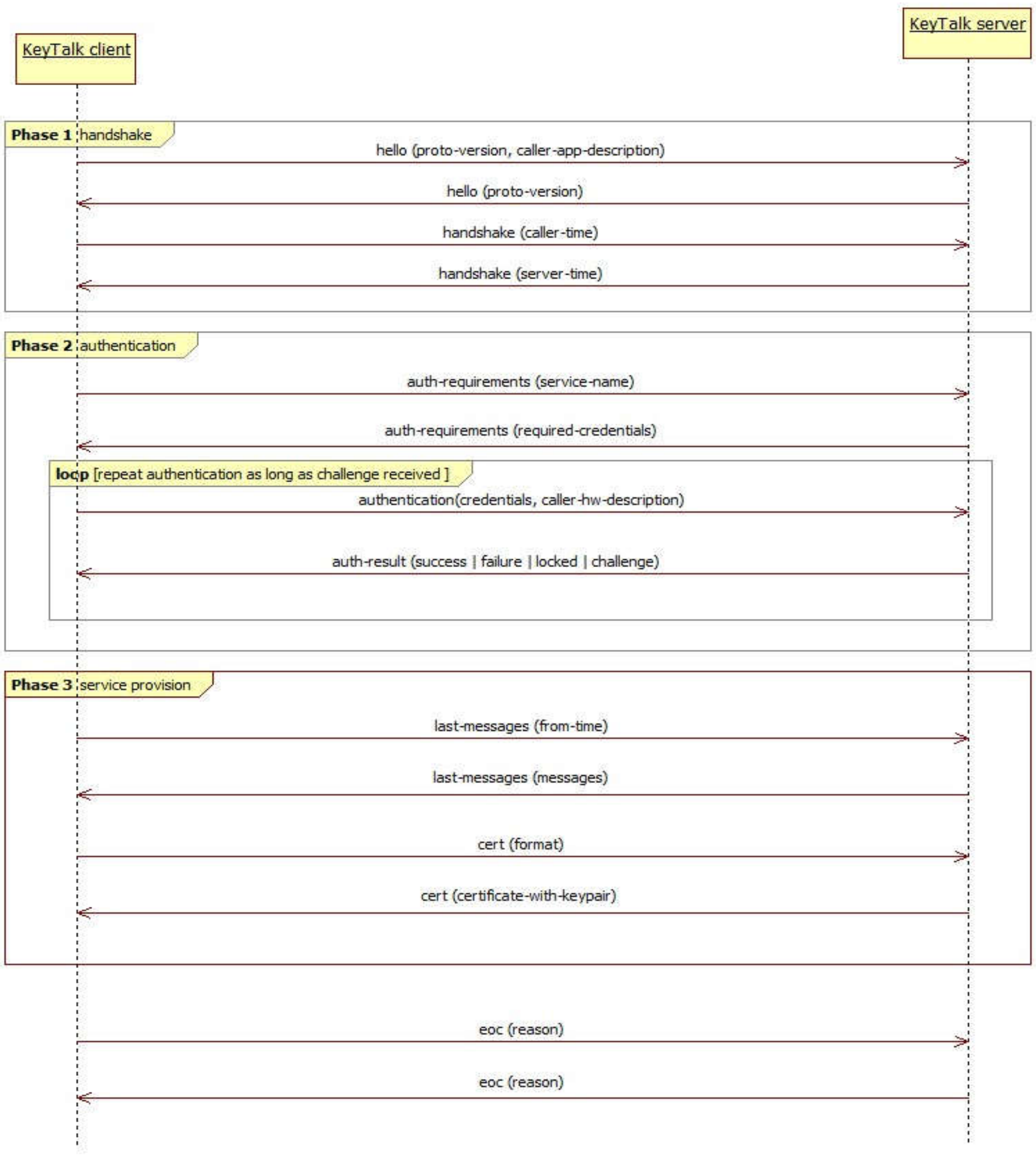
```
HTTP/1.1 200 OK
Content-type: application/json
Cache-Control: no-cache
Set-Cookie: keytalkcookie=a622bb821bec1f5315668c8f9a8e780f

{'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


KeyTalk client
KeyTalk server
Phase 1 handshake
hello (proto-version, caller-app-description)
hello (proto-version)
handshake (caller-time)
handshake (server-time)
Phase 2 authentication
auth-requirements (service-name)
auth-requirements (required-credentials)
loop [repeat authentication as long as challenge received]
authentication(credentials, caller-hw-description)
auth-result (success | failure | locked | challenge)
Phase 3 service provision
last-messages (from-time)
last-messages (messages)
cert (format)
cert (certificate-with-keypair)
eoc (reason)
eoc (reason)

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/2.1.0/eoc
```

Example:

```
/rcdp/2.1.0/eoc
/rcdp/2.1.0/eoc?reason=bye%2C+server
```

Query parameters

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

Response

```
HTTP 200 - application/json
```

```
{
  'status': 'eoc',
  [optional] 'reason': optional reason for ending communication
}
```

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

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/2.1.0/error
```

Example:

```
/rcdp/2.1.0/error?code=1066&description=invalid+response
```

Query parameters

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

Response

`HTTP 200 - application/json`

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

Error codes

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

2.5 Phase 1 (handshake)

2.5.1 Hello

Agree on RCDP protocol version and establish session ID.

Request

```
GET /rcdp/2.1.0/hello
```

Example:

```
/rcdp/2.1.0/hello
/rcdp/2.1.0/hello?caller-app-description=Demo+KeyTalk+client
```

Query parameters

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

RCDP protocol version proposed by a caller is sent as a part HTTP GET path. Currently the only supported version is `2.1.0`

Response

```
HTTP 200 - application/json
```

```
{
  "status": "hello",
  "version": proposed protocol version (currently "2.1.0")
}
```

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/2.1.0/handshake
```

Example:

```
/rcdp/2.1.0/handshake?caller-utc=2016-04-22T10%3A44%3A35.746255Z
```

Query parameters

parameter	type	required	description

`caller-utc`	*UTC string in ISO 8601 format including date and time*	yes	caller UTC

If the caller supports protocol version proposed by the server on the previous step, it proceeds with this version in HTTP GET path. Otherwise the caller ends communication. Currently the server supports RCDP version 2.0.0 and 2.1.0

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 /rcdp/2.1.0/auth-requirements`

Example:

`/rcdp/2.1.0/auth-requirements?service=DEMO_SERVICE`

Query parameters

parameter	type	required	description
`service`	*string*	yes	KeyTalk service name

Response

`HTTP 200 – application/json`

```
{
  "status": "auth-requirements",
  "credential-types": credential types,
  [optional] "hwsig_formula": HWSIG formula,
  [optional] "password-prompt": password-prompt,
  [optional] "service-uris": service URIs,
  [optional] "resolve-service-uris": need to resolve service URIs? ,
  [optional] "calc-service-uris-digest": need to calculate service URIs digest?
}
```

`credential-types`

JSON array of credential types required to authenticate against the given service. Supported credential types are: `"USERID"`, `"HWSIG"`, `"PASSWD"`, `"PIN"` and `"RESPONSE"`.
Example: `["USERID", "HWSIG", "PASSWD"]`

`hwsig_formula`

formula to calculate caller's hardware signature.
Example: `"1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16"`. Sent when `credential-types` parameter contains `HWSIG`.

`password-prompt`

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

`service-uris`

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

Example:
`[“https://demo1.keytalk.com”, “https://demo2.keytalk.com”]`
or
`[“file://%ProgramFiles%\vpn\vpn.exe”]`

`resolve-service-uris`

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

`calc-service-uris-digest`

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

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-uri”: [“https://demo.keytalk.com”],
  “resolve-service-uri” : “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

`GET /rcdp/2.1.0/authentication`

Example:

```
/rcdp/2.1.0/authentication?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
```

Query parameters

parameter	type	required	description
`service`	*string*	yes	KeyTalk service name
`caller-hw-description`	*string*	yes	Caller HW description which should be unique for the given device. For uniqueness e.g. BIOS serial number or iOS device UDID can be used. Examples: - `Windows 10, BIOS s/n 1234567890` - `iPAD: Jan’s iPAD 234567890abcdef1234567890abcdef`
`USERID`	*string*	if requested	ID of the user. Required if `USERID` was previously set by the server

			in `auth-requirements` response.
`HWSIG`	*string*	if requested	Hardware Signature of the caller's device calculated with the formula specified in the previous `auth-requirements` server response. Required if `HWSIG` was previously set by the server in `auth-requirements` response..
`PASSWD`	*string*	if requested	User password. Required if `PASSWD` was previously set by the server in `auth-requirements` response.
`PIN`	*string*	if requested	User pincode. Required if `PIN` was previously set by the server in `auth-requirements` response.
`resolved`	*JSON array*	if requested	JSON array of objects containing service URIs accompained 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: `[ { "uri":"https://demo1.keytalk.com", "ips":["81.175.10.107","81.175.103.109"] }, { "uri":"https://demo2.keytalk.com", "ips":["81.175.10.108","[2001:db8:a0b:12f0::1]"] } ]`
`digests`	*JSON array*	if requested	JSON array of objects containing service URIs accompained with SHA-256 hexadecimal digest of the underlying file. Required if `calc-service-uris-digest` was previously set in `auth-requirements` response. Example: `[ { "uri":"file://%Program Files%\vpn\vpn.exe", "digest":"01c7198fb614bf8746b46062aa551dff4506dd553ad96817622c76dafe8dc354" }, { "uri":"file://%Program Files%\vpn\vpn2.exe", "digest":"01c7198fb614bf8746b46062aa551dff4506dd553ad96817622c76dafe8dc355" } ]`

Response

5 `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.

5

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 filled in as described below in 4.5.2.1. This is called Challenge-Response authentication. Example: RADIUS EAP AKA/SIM authentication.

2.6.2.1 Challenge-response authentication

Request

`GET /rcdp/2.1.0/authentication`

Example:

```
/rcdp/2.1.0/authentication?responses=%7B%22CK%22%3A+%22123%22%2C+%22RES%22%3A
+%22456%22%2C+%22IK%22%3A+%22789%22%7D
```

Query parameters

parameter	type	required	description
`responses`	*JSON object*	yes	JSON array of responses. Response names should be the same as returned by the server on the previous `authentication`

request.
Example:
```
[
   {"name":"RES", "value":"123"},
   {"name":"IK", "value":"456"},
   {"name":"CK", "value":"789"}
]
```

Response

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:”}],
}
```

2.6.3 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

`GET /rcdp/2.1.0/change-password`

Example:

`/rcdp/2.1.0/change-password?old-password=changeme&new-password=changed`

Query parameters

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

Response

See 4.5.2, with authentication status restricted 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/2.1.0/last-messages
```

Example:

```
/rcdp/2.1.0/last-messages
/rcdp/2.1.0/last-messages?from-utc=2016-04-26T06%3A49%3A55.614010Z
```

Query parameters

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

Response

```
HTTP 200 – application/json
```

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

Example:

```
{
  "status": "last-messages",
  {"messages": [{"text": "This is user message number 1",
                 "utc": "2007-04-06T04:15:15+0000"},
                {"text": "This is user message number 2",
                 "utc": "2008-03-04T02:10:10+0000"},
                {"text": "This is user message number 3",
                 "utc": "2009-02-02T00:05:05+0000"}]
}
```

2.7.2 Retrieve certificate

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

Request

`GET /rcdp/2.1.0/cert`

Example:

```
/rcdp/2.1.0/cert?format=P12
/rcdp/2.1.0/cert?format=PEM&include-chain=True
/rcdp/2.1.0/cert?format=P12&out-of-band=True
```

Query parameters

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

Response

`HTTP 200 – application/json`

```
{
  "status": "cert",

  "cert": certificate in the desired format returned when out-of-band is not set.
              PEM-encoded certificate has its private key encrypted with the first 30 characters of the
session ID sent by the server in keytalkcookie.
              When the certificate is delivered in PKCS#12 package, the package gets encrypted with
with the first 30 characters of the session ID sent by the server in keytalkcookie and subsequently
base64 encoded to be transported with JSON,

 "cert-url-templ": certificate download URL template returned when out-of-band is set.
          The template conatins $(KEYTALK_SVR_HOST) placeholder that needs to be instantiated with
a hostname or IP address of the KeyTalk server used by the caller to make up a valid URL. The
download URL is valid for a limited amount of time (normally 5 minutes) and gets invalidated after
the first use.
          PEM-encoded certificate has its private key encrypted with the first 30 characters of the
session ID sent by the server in keytalkcookie.
```

When the certificate is delivered in PKCS#12 package , the package gets encrypted with with the first 30 characters of the session ID sent by the server in `keytalkcookie`,

`“execute-sync”:` boolean flag indicating whether the caller should invoke the service URIs synchronously (`true`) or asynchronously (`false`). Defaults to `false`.
}

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

```
{
  “status”: “cert”,
  “cert”:
“MIILjgIBAzCCC1gGCSqGSIb3DQEHAaCCC0kEggtFMIILQTCCBdcGCSqGSIb3DQEHBqCCBcgwggX
EAgEAMIIFvQYJKoZIhvcNAQcBMBwGCiqGSIb3DQEMAQYwDgQIBbLhaFnySsYCAggAgIIFkCSJcAh
E4IlFNQYJ23jqAI/+MHXBpCV+0dWleraxagN7b8QXqxXhJhLezwifrL/zBUGYcjN9pwvpdXBmZzb
nUdO+sAEPx4EDbAbyn7hDWp/fhJnyc3qD+6ilZz6zeDtB+3Eyje7VRl7VaJvNVFhN6I04RF2wmBF
R9wvmp8I/StNE0p6acN8RiLLm9JgIaVutJPsqA76e6X1yFlVJJmiBiMeqaJeuUuCcoGcrNdMOsrL
2J+/T8+Vk9RlTXAFGRj6dVAyBAjfkdTLno0qqg==“
}
```

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“
}
```