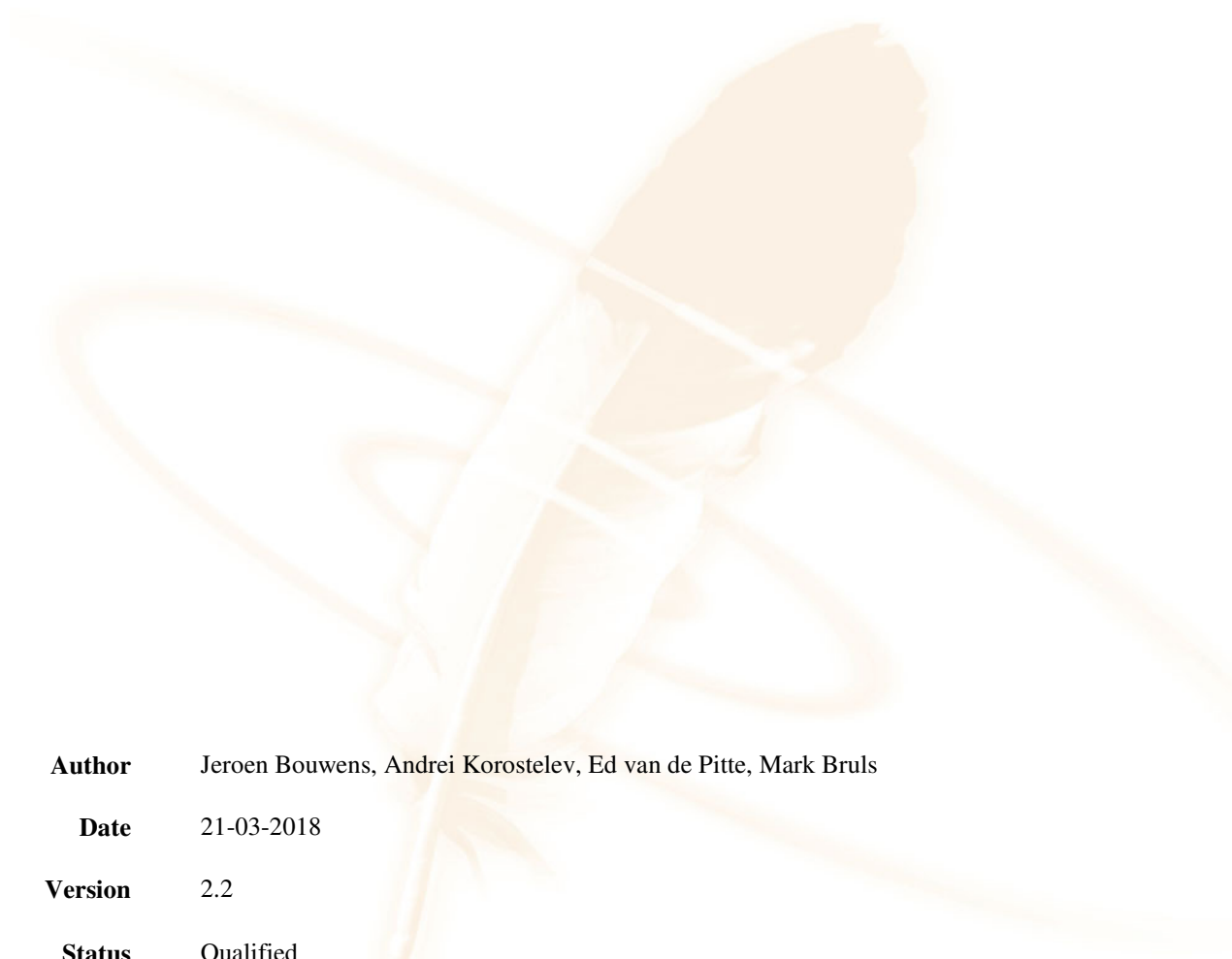




KeyTalk - Protocols



Author	Jeroen Bouwens, Andrei Korostelev, Ed van de Pitte, Mark Bruls
Date	21-03-2018
Version	2.2
Status	Qualified
Ref. no.	Resept036
Ref. name	KEYTALK_PROT
Project	KEYTALK

TABLE OF CONTENTS

	1. CHANGE HISTORY -----	II
	2. INTRODUCTION -----	1
5	2.1 Purpose-----	1
	2.2 Scope -----	1
	2.3 Definitions and abbreviations-----	1
	2.3.1 Definitions	1
	2.3.2 Abbreviations	1
10	2.3.3 Referenced documents	2
	2.3.4 Controlling documents	2
	2.3.5 Controlled documents	2
	3. RCDP V1 -----	3
	3.1 Client -> Server-----	3
	3.2 Server -> Client-----	4
15	3.3 RCDP specification-----	5
	3.3.1 ABNF Example	5
	3.3.2 Communication phases	5
	3.3.3 Common rules	5
20	3.3.4 Phase 1 (security mechanism and protocol handshake)	6
	3.3.5 Phase 2 (encryption establishment)	7
	3.3.6 Phase 3 (authentication)	10
	3.3.7 Phase 4 (maintenance)	16
	3.3.8 Phase 5 (service provision)	18
	3.3.9 RCDP versioning	21
25	3.4 Client RCDP state diagram-----	23
	4. RCDP V2 -----	25
	4.1 RCDPv2 overview -----	25
	4.2 RCDPv2 communication phases -----	26
30	4.3 Messages sent in all phases -----	27
	4.3.1 End Of communication	27
	4.3.2 Error	27
	4.4 Phase 1 (handshake) -----	29
	4.4.1 Hello	29
	4.4.2 Handshake	29
35	4.5 Phase 2 (authentication)-----	31
	4.5.1 Request authentication requirements	31
	4.5.2 Authentication	32
	4.5.3 Change password	37
40	4.6 Phase 3 (service provision)-----	39
	4.6.1 Check for the last messages	39
	4.6.2 Retrieve certificate	40
	4.7 RCDPv2 state diagram-----	41

1. CHANGE HISTORY

Version	Date	Author	Description
0.1	18-apr-2008	Jeroen Bouwens	Initial version.
0.2	27-Dec-2010	Andrei Korostelev	Added random noise to pubkey-encrypted RCDP messages (i.e. before the shared key is setup)
0.3	5-Jan-2011	Andrei Korostelev	Used CBC mode for AES used for symmetric encryption with shared key to improve security
1.0	17-feb-2011	Andrei Korostelev	Server signs all outgoing messages before shared key is established Included in R-4.0
1.1	31-Mar-2011	Andrei Korostelev	Moved RBP to BackendAuthd doc
1.2	11-Apr-2012	Andrei Korostelev	Reviewed RCDP spec
1.3	25-Jul-2013	Ed van de Pitte	Extended AuthResult with remaining time until password becomes invalid.
1.4	30-Aug-2013	Mark Bruls	Include of LDAP Change Password function.
1.5	04-Sep-2013	Andrei Korostelev	RCDP changes wrt added RADIUS CR support
1.6	10-Sep-2013	Pierre Urlings	Added hardware description to AUTH message.
1.7	13 Sep-2013	Andrei Korostelev	RCDP changes wrt added RADIUS CR support
1.8	23 Sep-2013	Andrei Korostelev	Password credential name is added to AUTHREQ message
1.9	20 Nov 2013	Andrei Korostelev	Added support for RCDP-1.3: CHALLENGE and client-server version negotiation.
1.10	14 Jan 2014	Mark Bruls & Andrei Korostelev	Added support for RCDP-1.4: - Use random IV for symmetric encryption (phases 3-5) - Use AES GCM for symmetric encryption (phases 3-5)
1.11	06 May 2014	Andrei Korostelev	Removed support for RCDP versions 1.0 to 1.3 to disallow downgrading to less secure proto version. From now the minimal supported RCDP version is 1.4
1.12	17 June 2014	Andrei Korostelev	Added RCDP-1.5. Desktop clients shall also be backward-compatible with old servers just like mobile clients.
2.0	19 May 2016	Andrei Korostelev	RCDP-2.0
2.0.1	3 January 2016	Andrei Korostelev	Update RCDP-2.0 docu regarding JSON serialization
2.1	12 December 2017	Andrei Korostelev	Remove HWSIG from get-challenge REMAP request
2.2	21 March 2018	Andrei Korostelev	Remove REMAP spec because of phasing out mod_relay and backendauthd

2. INTRODUCTION

2.1 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.

2.2 Scope

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

2.3 Definitions and abbreviations

2.3.1 Definitions

:
:
:

2.3.2 Abbreviations

RDD : RESEPT Dispatcher Daemon
RC : RESEPT Client
RCDP : RESEPT Client <-> RESEPT Dispatcher Daemon Protocol
ABNF : Augmented Backus-Naur Form

2.3.3 Referenced documents**2.3.4 Controlling documents****2.3.5 Controlled documents**
n/a

3. RCDP V1

This section describes RCDP protocol version 1. RCDP is used for communication between RESEPT Client and RESEPT Dispatcher Daemon on the server side.

5 RCDP uses HTTP as a transport. This requires the client to act as a regular webclient while sending messages to the server. A part of this act is using browser-style HTTP headers for every request.

3.1 Client -> Server

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

10

HTTP Header	Required	Description
POST	YES	/resept HTTP/1.1.
Host	YES	Should contain the FQDN OR IP (v4 or v6) of the server.
User-Agent	YES	Should contain any value valid in the HTTP specification. It may either reflect the usage of RESEPT or fake a genuine browser
Cookie	YES except for VSMREQ	Session identifier previously received from the server.
Content-type	YES	Should contain "multipart/form-data; boundary={ mime boundary}"
Cache-Control	YES	Should always be "no-cache"
Accept	NO	Contains the expected response content type (e.g. 'multipart/mixed')
Accept-Encoding	NO	If the header is supplied (e.g. "gzip,deflate") and honoured by the server, the client shall be able to process the encoded response.
Accept-Charset	NO	Can be for example "ISO-8859-1,utf-8;q=0.7,*;q=0.7"

A typical set of client headers could be:

```
POST /resept HTTP/1.1
User-Agent: RESEPT/4.2
Host: {SERVER-FQDN-OR-IP}
Accept: multipart/mixed
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Content-type: multipart/form-data; boundary={mime boundary}
Cookie: RESEPTSID=243fdccc5a7c4866c90ce900ef0517e3
Cache-Control: no-cache
```

15 After HTTP headers, RESEPT request payload message is sent. The payload is encoded using "multipart/form-data". The message name is composed of "RCDP_" followed by message type. Currently only one message is allowed per request. Message body is base64 encoded.

```
-----{mime boundary}
Content-Disposition: form-data; name="RCDP_{REQUEST-TYPE}"
Content-Transfer-Encoding: base64

{base64 encoded message body}
-----{mime boundary}--
```

An example payload could look like this:

```
-----13640682751323854200293646657
Content-Disposition: form-data; name="RCDP_EOC"
Content-Transfer-Encoding: base64

RU9D
-----13640682751323854200293646657--
```

A client should support at least basic and digest forms of HTTP authentication. Describing HTTP authentication is out of scope of this document. The server sends the Content-Transfer-Encoding but does not accept it itself.

3.2 Server -> Client

The set header set that the server needs to return to a client is:

HTTP Header	Required	Description
HTTP	YES	Contains "HTTP/ 1.1 200OK"
Set-Cookie	For VSMACK only	Contains session identifier generated by the server.
Content-type	YES	Should always contain "multipart/mixed; boundary={mime boundary}"

A valid set of headers could be:

```
HTTP/1.1 200 OK
Content-type: multipart/mixed; boundary={mime boundary}
Set-Cookie: RESEPTSESSIONID=fb6e7f61912ceac0b5dd33f487f1747;

-----{mime boundary}
Content-Disposition: inline; name="RCDP_{RESPONSE-TYPE}"
Content-Transfer-Encoding: base64

{base64 encoded message body}
-----{mime boundary}
```

When requested by a client, the server should be able to deal both with closed connections (HTTP/1.0) as well as with persistent connections (HTTP/1.1).

3.3 RCDP specification

This section describes RCDP using ABNF (RFC 5234). Below we give an example to get a quick understanding of ABNF syntax, please refer to RFC 5234 (<http://rfc.net/rfc5234.txt>) for complete ABNF description.

5

3.3.1 ABNF Example

We show how ABNF can be used to describe postal address:

```
postal-address = name-part street zip-part

name-part     = *(personal-part SP) last-name [SP suffix] CRLF
name-part     =/ personal-part CRLF

personal-part = first-name / (initial ".")
first-name    = *ALPHA
initial       = ALPHA
last-name     = *ALPHA
suffix        = ("Jr." / "Sr." / 1*("I" / "V" / "X"))

street        = [apt SP] house-num SP street-name CRLF
apt           = 1*4DIGIT
house-num     = 1*8(DIGIT / ALPHA)
street-name   = 1*VCHAR

zip-part      = town-name "," SP state 1*2SP zip-code CRLF
town-name     = 1*(ALPHA / SP)
state         = 2ALPHA
zip-code      = 5DIGIT ["-" 4DIGIT]
```

3.3.2 Communication phases

10 The complete communication circle between the client and the server consists of 5 phases:

1. Security mechanism and protocol handshake
2. Encryption establishment
3. Authentication
4. Maintenance
- 15 5. Service Provision

Below we describe message semantics on each phase in detail.

3.3.3 Common rules

20 There are a couple of ABNF rules used in the communication phases below. We define them here to avoid repeating ourselves later.

```
base64-str      = *( ALPHA / DIGIT / "+" / "-" ) * "="
non-empty-base64-str = +( ALPHA / DIGIT / "+" / "-" ) * "="
non-neg-dec     = 1*DIGIT
```



```

positive-dec      = %x31-39 *DIGIT
boolean           = 0 / 1
version           = ( ( positive-dec "." non-neg-dec ) / ( non-neg-dec
                      "." positive-dec ) ) [ "." non-neg-dec ]

```

3.3.4 Phase 1 (security mechanism and protocol handshake)

VSMREQ

Description: Initiate the handshake of the security mechanism and protocol and establish common session id

Sender: client

Syntax:

```

VSMREQ           = "VSMREQ" key-agreement-type proto-version [ client-
                      desc ]
key-agreement-type = "R3" / "EC"
proto-version      = version
client-desc        = non-empty-base64-str

```

"R3"

client wants key agreement (phase 2) using traditional RSA/Diffie-Hellman approach

(RESEPT 3 - alike)

"EC"

client wants key agreement (phase 2) using Elliptic Curve. Elliptic Curve-based key agreement is not yet officially supported by RESEPT.

proto-version

protocol version to use proposed by the client in *major.minor[.subminor=0]* format

client-descr

Single line¹ base64-encoded string describing the client such as "KeyTalk Windows Client-4.3.3" or "iOS KeyTalk Client-4.3.3"

VSMACK

Description: Handshake the security mechanism and protocol version and return session id. Session id is returned in HTTP 'Set-Cookie' header

Sender: server

Syntax:

```

SIGNED-VSMACK    = signature VSMACK
VSMACK           = "VSMACK" proto-version
proto-version     = version

```

signature

sha-1 digest of the VSMACK message signed with the client-server communication private key

proto-version

protocol version as *major.minor[.subminor=0]* proposed by the server for further communication. See [RCDP versioning](#) for more info.

session-id

hexadecimal-encoded 128 bit session ID (sent in HTTP headers)

¹ Single-line base64-encoding mimics the behavior of 'openssl base64 -e -A' in contrast to the traditional multiline behavior of 'openssl base64 -e'

EOC

Description: end of communication

Sender: client&server

Syntax:

```
EOC          = "EOC"
```

No error code is returned until the connection gets encrypted. A shared key for encrypting the connection is established in the phase 2. EOC is sent if either party cannot continue or session time-out has occurred.

5

3.3.5 Phase 2 (encryption establishment)**CT**

Description: Send the current client time

Sender: client

Syntax:

```
ENC-CT      = <encrypted-CT>
              ; encrypted with the client-server communication
              public key2

CT          = "CT" time random-noise
time        = <ISO8601-time>
random-noise = base64-str
```

time

Client time in ISO8601 format, e.g. "2008-11-05T13:15:30Z"

random-noise

Single line base64-encoded random noise

R3USE

Description: Send the server part of the DH++.

Sender: server

Syntax:

```
SIGNED-R3USE = signature R3USE

R3USE       = "R3USE" salt p g dh-svr-pkey
salt        = base64-str

p           = 1*HEXDIG
g           = 1*HEXDIG
dh-svr-pkey = 1*HEXDIG
```

signature

² Encryption is performed using RSA algorithm with PKCS#1 v2.0 EME-OAEP padding applied to the blocks of data constructed as <data-block-size>#<data-block><padding>

sha-1 digest of the R3USE message signed with the client-server communication private key

salt Single line base64-encoded UTF-8 string containing hexadecimal server salt for DH algorithm

p Hex DH module

g DH generator; usually “2” or “5”

dh-svr-pkey Hex Server DH public key (Alice’s) generated from the server DH private key, p and g.

R3RESP

Description: Client side response to the DH++

Sender: client

Syntax:

```
ENC-R3RESP      = <encrypted-R3RESP>
                  ; encrypted with the client-server communication public
                  key3

R3RESP           = "R3RESP" salt dh-clnt-pkey random-noise

salt             = base64-str

dh-clnt-pkey     = 1*HEXDIG

random-noise     = base64-str
```

dh-clnt-pkey Hex client DH public key (Bob’s) generated from DH private key, p and g received from the server

salt Single line base64-encoded UTF-8 string containing hexadecimal client salt for DH algorithm. This salt is appended to the server salt thus making up DH++ salt. *Client-side salt is sent onwards RCDP-1.5.*

random-noise Single line base64-encoded random noise

R3OK

Description: Server side response to the DH++⁴

Sender: server

Syntax:

```
SIGNED-R3OK     = signature R3OK

R3OK            = "R3OK" time

time            = <ISO8601-time>
```

signature

³ Encryption is performed using RSA algorithm with PKCS#1 v2.0 EME-OAEP padding applied to the blocks of data constructed as <data-block-size>#<data-block><padding>

⁴ DH++ stands for establishing shared key using Diffie–Hellman (DH) mechanism and adding salt for extra security. The salt is composed of client salt for RCDP version 1.4 and of <server salt><client salt><hex-of-sha-256 of all phase2 traffic communicated in both directions > for RCDP 1.5+

sha-1 digest of the R3OK message signed with the client-server communication private key
time

Server time in ISO8601 format, e.g. "2008-11-05T13:15:30Z". This time may be used by the client to synchronize (not implemented yet).

ECUSE

Description: send the server part of the ECDH++

Sender: server

Syntax:

```
SIGNED-ECUSE = signature ECUSE
ECUSE        = "ECUSE" curve-name conv generator order cofactor p a b
curve-name   = base64-str
conv         = [-] 1*DIGIT
generator    = 1*HEXDIG
order        = 1*HEXDIG
cofactor     = 1*HEXDIG
p            = 1*HEXDIG
a            = 1*HEXDIG
b            = 1*HEXDIG
ecdh-svr-pkey = 1*HEXDIG
```

curve-name

Single line base64-encoded ASCII string containing named curve

conv

Conversion form

generator

Generator as HEX string

order

Order of the generator as HEX string

cofactor

Co-factor of the generator as HEX string

p

P parameter (prime) as HEX string

a

A parameter as HEX string

b

B parameter as HEX string

dh-svr-pkey

X and Y coordinates of the server-side public key (point) as HEX string

ECRESP

Description: Client side response to the ECDH++

Sender: client

Syntax:

```

ENC-ECRESP      =      <encrypted-ECRESP>
                        ; encrypted with the client-server communication
                        public key

ECRESP          =      "ECRESP" pubkey random-noise
ecdh-clnt-pkey  =      1*HEXDIG
random-noise    =      base64-str

```

ecdh-clnt-pkey

X and Y coordinates of the client-side public key (point) as HEX string

random-noise

Single line base64-encoded random noise

ECOK

Description: Server side response to the ECDH++

Sender: server

Syntax:

```

SIGNED-ECOK     = signature ECOK

ECOK            = "ECOK" time

time            = <ISO8601-time>

```

signature

sha-1 digest of the ECOK message signed with the client-server communication private key

time

Server time in ISO8601 format, e.g. 2008-11-05T13:15:30Z. Time may be used by the client to synchronize.

EOC

Same as on Phase 1 (security mechanism and protocol handshake).

3.3.6 Phase 3 (authentication)

For phase 3 and higher, encryption is performed using symmetric AES algorithm using shared key generated the Phase 2 (key agreement).

AES GCM algorithm is used for encryption. Authentication data and randomized Initialization Vector (IV) are prepended to the encrypted message.

The encrypted message is formatted follows:

```
<16-byte-tag><16-byte-iv><encrypted-msg>
```

where *tag* is AES GCM authentication tag to verify integrity of the received message, *IV* is Initialization Vector, and *encrypted-msg* is a message encrypted with the shared session key established on the Phase 2 (key agreement).

AUTH

Description: Client's request for the service

Sender: client

Syntax:

```
ENC-AUTH    = <encrypted-AUTH>
              ; encrypted with the shared session key
              ; established on the Phase 2 (key agreement)

AUTH        = "AUTH" service hwdesc request

hwdesc      = non-empty-base64-str

service     = non-empty-base64-str

request     = [ renew ] [ sync ]

renew       = "RENEW"

sync        = "SYNC"
```

service

Single line base64-encoded UTF-8 service name string

hwdesc

Description of the client hardware containing:

- Windows Desktop Client: Windows name and BIOS serial number, e.g. "Windows 7 , BIOS s/n 1234567890"
- iOS client: Device model, device name and device UDID e.g. "iPad: Jan's iPad 234567890abcdef1234567890abcdef"

"RENEW"

client is requesting a renewal of the previous session. This flag is for future releases⁵.

"SYNC"

Client informs that it has synchronized its time with the server time, which means that no time difference check between the client and the server need to be performed before issuing the certificate or performing any other service where timesync. matters. If the flag is not set, the timesync. check will be performed with certificate TTL and TFC values.

AUTHREQ

Description: Authentication type required for this service

Sender: server

Syntax:

```
ENC-AUTHREQ    = <encrypted-AUTHREQ>
                  ; encrypted with the shared session key
                  ; established on the Phase 2 (key agreement)
```

⁵ The "RENEW" flag is added for allowing certificate renewal without need for providing username/password. For this purpose the server needs to keep a list of generated keypairs used in issued certificates (keyrollover). When the server gets a RENEW request it sends a challenge to the client. As a response the client shall send back a challenge encrypted with the cert's private key. If the server manages to decrypt the challenge it will issue a new (prolonged) certificate using the same keypair.

```

AUTHREQ      = "AUTHREQ" required-auth
required-auth = 1*( "USERID" / "HWSIG:"hwsig_formula /
                  "PASSWD:"password-prompt / "PIN" / "RESPONSE" /
                  "RESPONSE:"challenge":"response-names / "BIO" /
                  "CSR")
hwsig_formula = positive-dec * ("," positive-dec)
challenge     = challenge-name "#" challenge-value * ( ","
               challenge-name "#" challenge-value )
response-names = response-name * ( "," response-name)
hwsig_formula  = base64-str
challenge-name = base64-str
challenge-value = non-empty-base64-str
response-name   = non-empty-base64-str
password-prompt = base64-str

```

"USERID"

Userid required

"HWSIG"

Hardware signature required

"PASSWD"

Password required

"PIN"

Pincode required

"RESPONSE"

Response will be requested in the next AUTHREQ message.

"RESPONSE:challenge"

Response is required for the provided challenge.

"BIO"

BIO-stream is required

"CSR"

certificate signing request is required.

hwsig_formula

Single-line base-64 encoded formula the client should use to calculate HWSIG

challenge-name

Single-line base-64 encoded challenge prompt displayed on the client.

challenge-value

Non-empty single-line base-64 encoded challenge.

password-prompt

Single-line base-64 string. This prompt is normally is displayed on the client e.g. "Tokencode"

response-name

Single-line base-64 encoded string. When multiple responses are required by the server, response name allows identifying each response sent by the client.

AUTHRESP

Description: Authentication response by the client

Sender: client

Syntax:

```

ENC-AUTHRESP      = <encrypted-AUTHRESP>
                    ; encrypted with the shared session key
                    ; established on the Phase 2 (key agreement)

AUTHRESP          = "AUTHRESP" auth-response
auth-response     = 1*( ( "USERID:"userid ) /
                        ( "HWSIG:"hwsig ) /
                        ( "PASSWD:"passwd ) /
                        ( "PIN:"pin ) /
                        ( "RESPONSE:"responses )
                      )
userid            = non-empty-base64-str
hwsig             = base64-str
passwd            = base64-str
pin               = base64-str
responses         = response-name "#" response-value * ( "," response-
                    name "#" response-value )
response-name     = non-empty-base64-str
response-value    = non-empty-base64-str

```

"USERID"

Userid provided

userid

Single line base64-encoded UTF-8 user id string

"HWSIG"

Hardware signature provided

hwsig

Single line base64-encoded HW signature

"PASSWD"

Password provided

passwd

Single line base64-encoded UTF-8 password string

"PIN"

Password provided

pin

Single line base64-encoded UTF-8 pin string

"RESPONSE"

Response is provided.

response

Dictionary of response name to response value.

AUTHRESULT

Description: Authentication result.

Sender: server

Syntax:


```

ENC-AUTHRESULT    = <encrypted-AUTHRESULT>
                    ; encrypted with the shared session key
                    ; established on the Phase 2 (key agreement)

AUTHRESULT        = "AUTHRESULT" auth-result
auth-result       = ("OK" [pwdtime]) / ( "DELAY" dtime ) / "LOCKED" /
                    "EXPIRED" / ( "CHALLENGE" challenge:"response-
                    names )
challenge         = challenge-name "#" challenge-value * ( ","
                    challenge-name "#" challenge-value )
challenge-name    = base64-str

response-names    = response-name * ( "," response-name)
challenge-value   = non-empty-base64-str
pwdtime          = dec
dtime            = non-neg-dec

```

"OK"

Authentication was successful

pwdtime

when authentication succeeds, indicates a number of seconds until the password expires or -1 if password never expires. This parameter appears only if password validity information is supplied by authentication backend.

"DELAY"

Authentication was not successful.

dtime

when authentication was not successful, indicates time in seconds for which the server will not allow to re-authenticate, can be zero.

"LOCKED"

Cannot login e.g. because the user is locked on the server.

"EXPIRED"

Authentication was not successful because the password of the user trying to authenticate is expired.

"CHALLENGE"

Another challenge is supplied by the server.

challenge-name

Single-line base-64 encoded challenge prompt displayed on the client e.g. "Enter your tokencode".

challenge-value

Non-empty single-line base-64 encoded challenge.

response-name

Single-line base-64 encoded string. When multiple responses are required by the server, response name allows identifying each response sent by the client.

LDAPCHANGEPWD

Description: Request by the client to change the LDAP password for the specified user

Sender: client

Syntax:

```


```

```

ENC-LDAPCHANGEPWD      = <encrypted-LDAPCHANGEPWD >
                        ; encrypted with the shared session key
                        ; established on the Phase 2 (key agreement)

LDAPCHANGEPWD          = "LDAPCHANGEPWD" ldapchangepwd-response
ldapchangepwd -
response               = 1*( ( "USERID:"userid ) /
                          ( "PASSWD:"passwd ) /
                          ( "NEWPASSWD:"passwd )
                          )

userid                 = non-empty-base64-str
passwd                 = base64-str

```

"USERID"
 Userid provided
 userid
 Single line base64-encoded UTF-8 user id string
 "PASSWD"
 Password provided
 "NEWPASSWD"
 Password provided
 passwd
 Single line base64-encoded UTF-8 password string

LDAPCHANGEPWDRESULT

Description: LDAP Change User Password result.

Sender: server

Syntax:

```

ENC-LDAPCHANGEPWDRESULT = <encrypted-LDAPCHANGEPWDRESULT>
                        ; encrypted with the shared session key
                        ; established on the Phase 2 (key agreement)

LDAPCHANGEPWDRESULT     = "LDAPCHANGEPWDRESULT " result
result                  = "OK" / ("DELAY" dtime) / "LOCKED"

dtime                   = non-neg-dec

```

"OK"
 LDAP Change password for user was successful
 "DELAY"
 LDAP Change password for user was not successful.
 dtime
 when LDAP Change password for user was not successful, indicates time in seconds for which
 the server will not allow to re-authenticate, can be zero.
 "LOCKED"
 LDAP Change password for user failed because the user is locked on the server.

On failed attempted (DELAY message) the client should try again. The client is not supposed to try another LDAP Change password for the user again until this time. If it does the server will respond with another DELAY message without trying to change the LDAP password for the user. The server is allowed to send the same or an updated delay message.

ERR

Description: Error in communication

Sender: client & server

Syntax:

```

ENC-ERR          = <encrypted-ERR>
                  ; encrypted with the shared session key
                  ; established on the Phase 2 (key agreement)

ERR              = "ERR" error-number [ error-description ]
error-number     = ["-"] 1*DIGIT
error-description = base64-str

```

error-number

error number. See below for the supported error codes.

error-description

Single line base64-encoded UTF-8 error description.

NOTSUP

Description: Not supported

Sender: client & server

Syntax:

```

ENC-NOTSUP       = <encrypted-NOTSUP>
                  ; encrypted with the shared session key
                  ; established on the Phase 2 (key agreement)

NOTSUP           = "NOTSUP" error-number [ error-description ]
error-number     = ["-"] 1*DIGIT
error-description = base64-str

```

error-number

error number.

error-description

Single line base64-encoded UTF-8 error description. Can be empty.

EOC

Same as on Phase 1 (security mechanism and protocol handshake).

3.3.7 Phase 4 (maintenance)**LICENSE**

Description: Client license

Sender: client & server

Syntax:

```
ENC-LICENSE      = <encrypted-LICENSE>
                  ; encrypted with the shared session key
                  ; established on the Phase 2 (key agreement)

LICENSE          = "LICENSE" license
license          = base64-str
```

license

Single line base64-encoded UTF-8 text of the license. License exchange is due future releases.

LASTMSG

Description: Check for the last messages received for the client

Sender: client

Syntax:

```
ENC-LASTMSG      = <encrypted-LASTMSG>
                  ; encrypted with the shared session key
                  ; established on the Phase 2 (key agreement)

LASTMSG          = "LASTMSG" timestamp
timestamp         = utc / number-of-messages / "ALL"
time              = <ISO8601-time>
number-of-messages = non-neg-dec
```

time

UTC time starting from which the client would like to receive messages. Time is in ISO8601 format (e.g. "2008-11-05T13:15:30+0000")

number-of-messages

Maximum number of messages the client would like to receive.

"ALL"

Request all available messages.

NEWMSG

Description: The new messages

Sender: server

Syntax:

```
ENC-NEWMSG       = <encrypted-NEWMSG>
                  ; encrypted with the shared session key
                  ; established on the Phase 2 (key agreement)

NEWMSG           = "NEWMSG" [ utc message [ utc message ] ... ]
utc              = <ISO8601-time>
message          = base64-str
```

utc

message UTC timestamp in ISO8601 format (e.g. "2008-11-05T13:15:30+0000")

message

Single line base64-encoded UTF-8 message text

The messages are returned sorted by date in ascending order

ERR

Same as on Phase 3 (authentication).

NOTSUP

Same as on Phase 3 (authentication).

EOC

Same as on Phase 1 (security mechanism and protocol handshake).

3.3.8 Phase 5 (service provision)

FORMAT

Description: Certificate format requested by the client

Sender: client

Syntax:

```
ENC-FORMAT      = <encrypted-FORMAT>
                  ; encrypted with the shared session key
                  ; established on the Phase 2 (key agreement)

FORMAT          = "FORMAT" cert-format
cert-format     = ( "PEM" / "P12" ) [ "chain" ]
```

"PEM"

Request PEM-encoded certificate

"P12"

Request PKCS#12-encoding certificate

"chain"

Request the whole certificate chain, i.e. with issuing CAs

URL

Description: URL resolution request

Sender: server

Syntax:

```

ENC-URL      = <encrypted-URL>
               ; encrypted with the shared session key
               ; established on the Phase 2 (key agreement)

URL          = "URL" rfc-url
rfc-url      = base64-str

```

rfc-url

Single line base64-encoded RFC 3986-compliant URL.

RESOLVED

Description: IP address(es) resolved from the URL

Sender: client

Syntax:

```

ENC-RESOLVED  = <encrypted-RESOLVED>
                ; encrypted with the shared session key
                ; established on the Phase 2 (key agreement)

RESOLVED      = "RESOLVED" ip-list
ip-list       = ip *( " " ip )
ip            = base64-str

```

ip

Single line base64-encoded IPv4 or IPv6 address according to RFC 3986

EXEC

Description: executable digest request

Sender: server

Syntax:

```

ENC-EXEC      = <encrypted-EXEC>
                ; encrypted with the shared session key
                ; established on the Phase 2 (key agreement)

EXEC          = "EXEC" program execute-sync
program       = executable-name
executable-name = base64-str
execute-sync  = boolean

```

executable-name

Single line base64-encoded RFC 3986-compliant UTF-8 name of the program main executable (may be full path such as file://%ProgramFiles%\vpn\vpn.exe or just a file name <file://vpn.exe>)

execute-sync

Boolean flag indicating whether the client should invoke the executable URI synchronously. Web URIs are always executed asynchronously.

DIGEST

Description: calculated digest of the executable

Sender: client

Syntax:

```
ENC-DIGEST      = <encrypted-DIGEST>
                  ; encrypted with the shared session key
                  ; established on the Phase 2 (key agreement)

DIGEST          = "DIGEST" dgst
dgst            = 64HEXDIG
```

digest

hexadecimal sha-256 hash of the executable.

CERT

Description: Send service URI and certificate in the requested format

Sender: server

Syntax:

```
ENC-CERT        = <encrypted-AUTH>
                  ; encrypted with the shared session key
                  ; established on the Phase 2 (key agreement)

CERT            = "CERT" cert [ uri [ execute-sync ] ]
cert            = base64-str
uri             = base64-str
execute-sync    = boolean
```

cert

Single line base64-encoded requested certificate(s). Private key parts of PFX or PEM is protected with the password constructed from the first 30 characters of the hexadecimal representation of the session key.

uri

Single line base64-encoded RFC 3986-compliant service URI same as in “URL” or “EXEC” rdd responses above. Service URI is only sent if it was not previously sent with “URL” or “EXEC” responses.

execute-sync

Boolean flag indicating whether the client should invoke the executable URI synchronously. The flag exists only if ‘uri’ option exists and it is file:// URI.

FIN

Description: End of Communication

Sender: client & server

Syntax:

```
ENC-FIN        = <encrypted-FIN>
```

```

; encrypted with the algorithm and parameters
; established on the Phase 2 (key agreement)

FIN      = "FIN"

```

ERR

Same as on Phase 3 (authentication).

NOTSUP

Same as on Phase 3 (authentication).

EOC

Same as on Phase 1 (security mechanism and protocol handshake).

The table below lists error code that can appear in Phase 5 ERR message per RCDP version.

Error number	Error description	Remarks
1001 (ErrResolvedIpInvalid)	N/A	Sent by the server when none of IPs resolved by the client and by the server match.
1002 (ErrDigestInvalid)	N/A	Sent by the server when the client's calculated executable digest does not match the digest stored on the server.
1003 (ErrTimeOutOfSync)	Client UTC - Server UTC in seconds	Sent by the server when the client time is out of sync with the server's time.
1004 (ErrMaxLicensedUsersReached)	N/A	Sent by the server when no certificate can be supplied because the max number of licensed users has been reached
1005 (ErrPasswordExpired)	N/A	Sent by the server when the password of the user trying to authenticate is expired but the client is not supposed to start password change procedure.

3.3.9 RCDP versioning

Both client and a server implementing the given protocol version **must** support all released protocol versions onwards 1.4. This allows new clients to communicate against old servers and other way round.

During RCDP handshake phase KeyTalk client proposes the highest RCDP version it implements to the KeyTalk server. The server responds with the highest supported protocol version which is equal or lower to the one proposed by the client. If the client supports the protocol version proposed by the server, handshake is complete. Otherwise the client sends EOC to the server and communication ends.

Protocol version is given as *1.minor[.subminor = 0]*. Minor part determines compatibility between client and server. Optional subminor part, if present, specifies non-significant changes in the protocol which do not require changes to another communication counterpart. An example of such a change is refined message description in ERR message. Subminor version is ignored during handshake and defaults to 0.

Below we show compatibility between KeyTalk Client and KeyTalk Server.

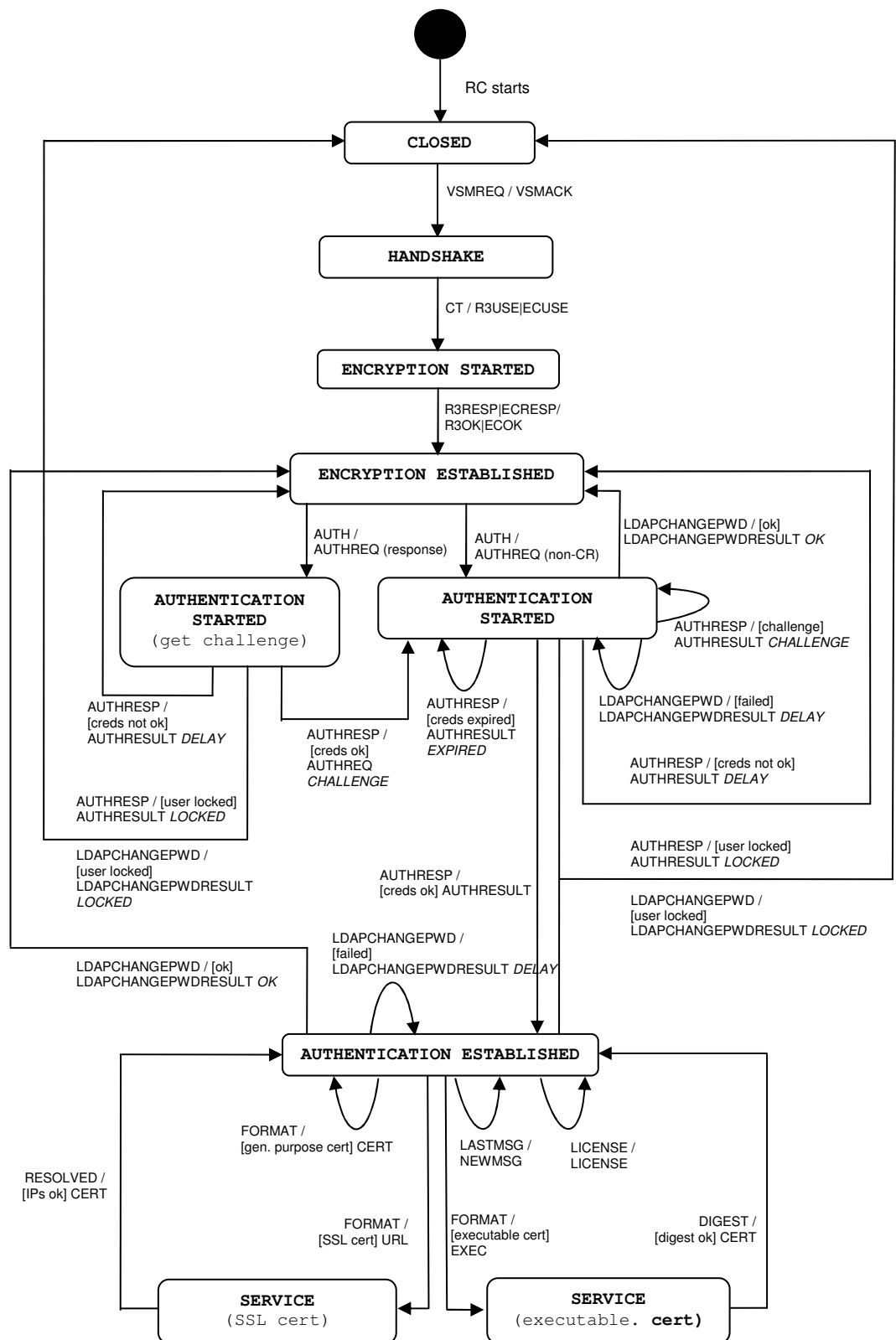
RCDP Version	Supporting server versions	Supporting client versions	Changes wrt the previous version
1.4	4.3.3+	4.3.3+	N/A
1.5	4.4.0+	4.4.0+	<ul style="list-style-type: none">- Both client & server contribute entropy for session key generation by adding phase2 traffic communicated in both directions to the DH++ salt- Client also contributes entropy for session key generation by sending salt in R3RESP to the server- All clients and all servers from version 4.3.3 shall be able to work with each other, handshaking protocol version.

3.4 Client RCDP state diagram

Below we show RCDP state diagram for a client.

The following transitions are and are not shown on the state diagram for the sake of simplicity:

1. From any state: when EOC or ERR is received from the server or when the client encounters an error (e.g. unexpected response received) the client enters CLOSED state
2. From any state: after sending VSMREQ and getting VSMACK back the client enters HANDSHAKE state.
3. From AUTHENTICATION ESTABLISHED state: after sending FIN and getting FIN back from the server, the client enters CLOSED state.



4. RCDP V2

This section describes RCDP protocol version 2. The motivation to develop a new protocol 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
- Add a possibility to generate key on the client and let the server sign it after successful authentication

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

4.1 RCDPv2 overview

Communication in RCDPv2 is encapsulated in HTTPS/TLSv1.2+ protocol and uses port 443. KeyTalk server utilises lighttpd webserver, which handles HTTPS and forwards underlying payload traffic to RDD daemon. Server identity should be verified by a caller using KeyTalk communication Certificate Authority installed from RCCD file.

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/2.0.0/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'}
```

4.2 RCDPv2 communication phases

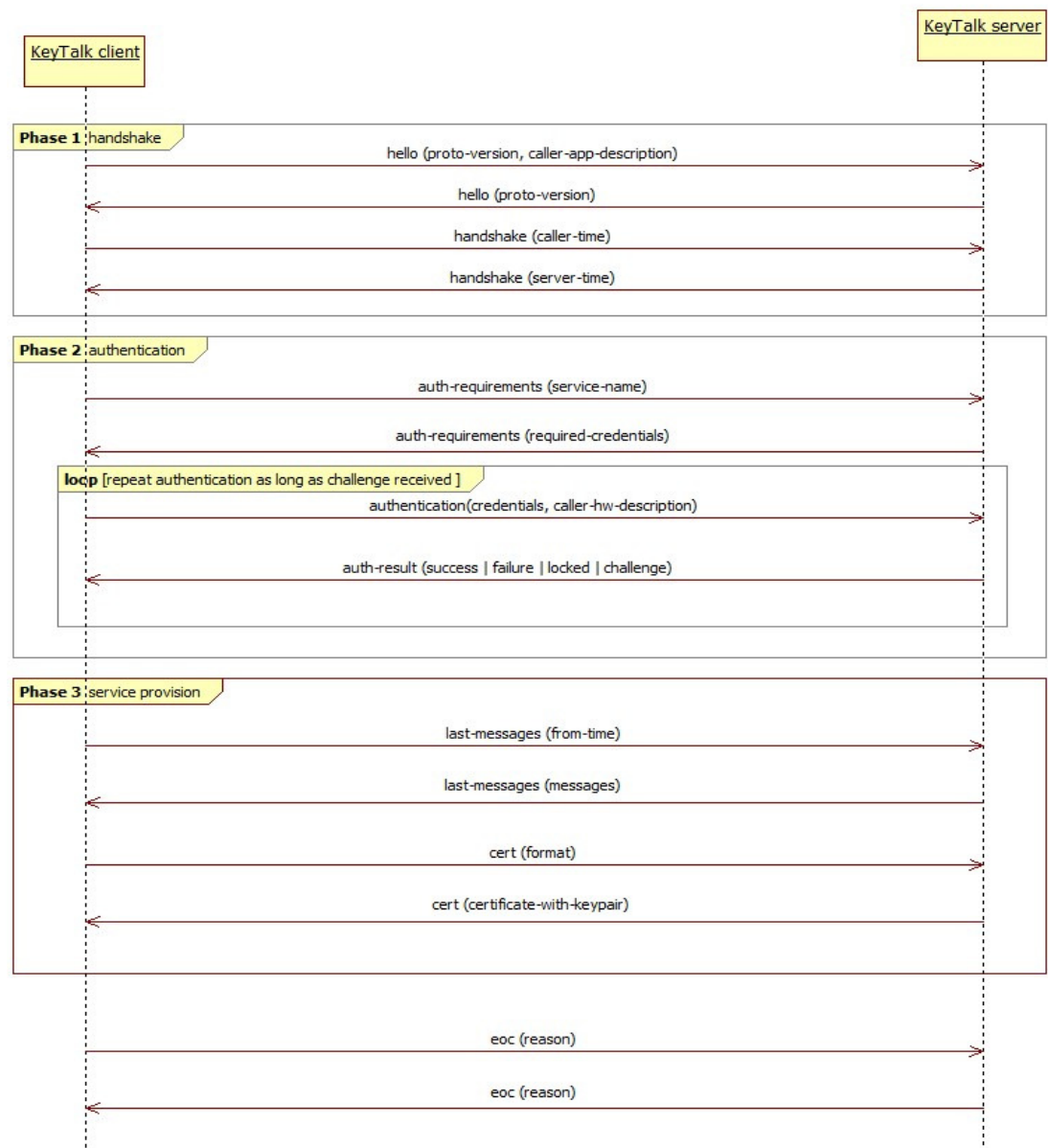
The complete RCDPv2 communication circle consists of 3 phases:

Phase1: handshake

Phase 2: authentication

Phase 3: service provision

5



Further we describe message semantics on each phase in detail.

10

4.3 Messages sent in all phases

4.3.1 End Of communication

Request

GET /rcdp/2.0.0/eoc

Example:

/rcdp/2.0.0/eoc
/rcdp/2.0.0/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.

4.3.2 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/2.0.0/error

Example:

/rcdp/2.0.0/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.

5

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 match 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.

10

4.4 Phase 1 (handshake)

4.4.1 Hello

Agree on RCDP protocol version and establish session ID.

Request

GET /rcdp/2.0.0/hello

Example:

/rcdp/2.0.0/hello
/rcdp/2.0.0/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 2.0.0

Response

HTTP 200 - application/json

```
{
  "status": "hello",
  "version": "proposed protocol version (currently \"2.0.0\")"
}
```

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

4.4.2 Handshake

Confirm version handshake and exchange time information.

Request

GET /rcdp/2.0.0/handshake

Example:

/rcdp/2.0.0/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 only supported version is 2.0.0

Response

HTTP 200 - application/json

```
{
  "status": "handshake",
  "server-utc": server UTC in ISO 8601 format including date and time
}
```

4.5 Phase 2 (authentication)

4.5.1 Request authentication requirements

Request authentication requirements from the server.

Request

GET /rcdp/2.0.0/auth-requirements

Example:

/rcdp/2.0.0/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"
}
```

4.5.2 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/2.0.0/authentication

Example:

```
/rcdp/2.0.0/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	<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: <ul style="list-style-type: none"> Windows 10, BIOS s/n 1234567890 iPAD: Jan's iPAD 234567890abcdef1234567890abcdef
USERID	<i>string</i>	if requested	ID of the user. Required if USERID was previously set by the server

			in <code>auth-requirements</code> response.
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..
PASSWD	<i>string</i>	if requested	User password. Required if PASSWD was previously set by the server in <code>auth-requirements</code> response.
PIN	<i>string</i>	if requested	User pincode. Required if PIN was previously set by the server in <code>auth-requirements</code> response.
resolved	<i>JSON array</i>	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 <code>resolve-service-uris</code> was previously set in <code>auth-requirements</code> 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	<i>JSON array</i>	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: [{ "uri": "file://%Program Files%\vpn\vpn.exe", "digest": "01c7198fb614bf8746b46062aa551dff4506dd553ad96817622c76dafa8dc354" }, { "uri": "file://%Program Files%\vpn\vpn2.exe", "digest": "01c7198fb614bf8746b46062aa551dff4506dd553ad96817622c76dafa8dc355" }]

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.

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:" }],
}
```

5 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": "1011112131415161718191a1b1c1d1e1f" } }],
  "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.

4.5.2.1 Challenge-response authentication

Request

GET /rcdp/2.0.0/authentication

Example:

/rcdp/2.0.0/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

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

10

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:"}],
}
```

5

4.5.3 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

10 GET /rcdp/2.0.0/change-password

Example:

/rcdp/2.0.0/change-password?old-password=changeme&new-password=changed

15

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 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.

4.6 Phase 3 (service provision)

4.6.1 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/2.0.0/last-messages

Example:

```
/rcdp/2.0.0/last-messages
/rcdp/2.0.0/last-messages?from-utc=2016-04-26T06%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": "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"}
  ]
}
```

4.6.2 Retrieve certificate

Retrieve a certificate along with a keypair in the desired format. The keypair is either supplied by the caller or is generated on the server

Request

GET /rcdp/2.0.0/cert

Example:

```
/rcdp/2.0.0/cert?format=P12
/rcdp/2.0.0/cert?format=PEM&include-chain=True
```

Query parameters

parameter	type	required	description
format	<i>string</i>	yes	One of: <ul style="list-style-type: none"> - "PEM" for PEM-encoded X.509 certificate and private key - "P12" for PKCS#12-encoded X.509 certificate and private key
include-chain	<i>true false</i>	no	Request the whole certificate chain, i.e. with issuing CAs. Defaults to false.
keypair	<i>JSON object</i>	no	When keypair is supplied the caller, the server will use that for signing and generating a certificate. Otherwise the server will generate a keypair itself. Format: <pre>{ "pubkey": "PEM-encoded PKCS#1 public key", "privkey": "PEM-encoded PKCS#5 private key" }</pre> <p>Currently only RSA keys are supported. PKCS#1-encoded PEM RSA public key should begin with "-----BEGIN RSA PUBLIC KEY-----"</p> <p>PKCS#5-encoded PEM RSA private key should begin with "-----BEGIN RSA PRIVATE KEY-----"</p>

Response

HTTP 200 - application/json

```
{
  "status": "cert",
  "cert": certificate. PEM certificate has its private key encrypted with the first 30 characters of
```

the session ID (sent by the server in `keytalkcookie` HTTP cookie) . PKCS#12 is internally encrypted with the above password and then base64 encoded to be transported with JSON,

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

```
}

```

Example:

```
{
  "status": "cert",
  "cert":
  "MIILjgIBAzCCC1gGCSqGSIb3DQEHAaCCC0kEggtFMIILQTCCBdcGCSqGSIb3DQEHBqCCBcgwggXEAgEAMIIFvQYJKoZIhvcNAQcBMBwGCiqGSIb3DQEMAQYwDgQIBbLhaFnySsYCAggAgIIFkCSJcAhE4I1FNQYJ23jqAI/+MHXBpCV+0dWleraxagN7b8QXqxXhJhLezwiFrL/zBUGYcjN9pwvpdXBmZzbNudO+sAEPx4EDbAbyn7hDWp/fhJnyc3qD+6ilZz6zeDtB+3Eyje7VRl7VaJvNVFhN6I04RF2wmBFR9wvmp8I/StNE0p6acN8RiLLm9JgIaVutJPsqA76e6XlyFlVJJmiBiMegaJeuUuCcoGcrNdMOsrL2J+/T8+Vk9RlTXAFGRj6dVAyBAjfkdtLno0qqg=="
}
```

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

4.7 RCDPv2 state diagram

@todo make multiple diagrams per use case: CR, non-CR, password change...