ST-1506 Remote management specification ST-1506-remote-management-specification

Contents

1 Document Information

Document Identification

Project	ST-1506
Title	Remote management specification
Document ID	ST-1506-remote-management-specification
Version	2.8
Git Hash	94a8c5b86789223f6e63a3e26e190df3e1784223
PDF Generator	Sphinx (sphinx-build) 1.5.3
Issue Date	2025-03-24
Status	Release
Editor	Martin Elshuber, Cherry Digital Health GmbH
Contributor(s)	Bert Schüszler, Cherry Digital Health GmbH

Document Distribution

Distribution Status	unrestricted
Distribution List	public

Revision History

0.1	2020-01-20	Initial version
0.2	2020-02-10	Improvements from first review round
0.3	2020-03-18	Rework remote JSON API
0.4	2020-03-24	Add error handling details.
0.5	2020-04-01	Small corrections.
0.6	2020-05-01	Adding idleScreenMessage and apiVersion
0.7	2020-05-01	Allow apiVersion without authentication
0.8	2020-09-25	Fix Error Code in SetDeviceName
0.9	2020-11-13	Corrections for FSP OR v3
1.0	2020-11-16	Corrections for FSP OR v4
1.1	2021-11-16	Adaptions for FW 3.0
1.2	2021-11-22	Adaptions for Remote SMCB API
1.3	2022-01-10	• Fix typos in Remote SMCB API • Extend reboot request for (Factory Reset)

Continued on next page

Table 1.1 – continued from previous page

1.4	2022-01-24	• Add messages: Get/SetDisplaysettings, Get/SetPinEntry Get/SetServiceAnnouncement Get/SetVpnConfiguration, SetRemotePin GetSicctEnvironment • Extend NetworkConfig/Info and SetActiveInterface with USB-CDC-ECM option • Extend DeviceInformation with MAC/gSMCKT information • Extend GetVpnMschapConfigurationRequest with mschapPasswordSet
1.5	2022-01-25	• Removed renamed message: Get/Set DisplayConfiguration
1.6	2022-02-04	• SMCB API + SMCB/InputPinRequest: fix typos
1.7	2022-02-04	• API version set version to 2.0
1.8	2022-02-22	• Remove confidential notice • Change distribution list • Change editor and contributions • Change issuer to *Cherry Digital Health GmbH* • Rename document from cobra to ST-1506
1.9	2022-03-24	• corrected info that messageId field is needed for most commands to: sessionId • added units for dpdDelay/dpdTimeout/ service interval • added possible values for SetRemotePin • clarified note for smbc auth. timeout • corrected some typos
2.0	2022-04-06	• added property networkMode to Get/ SetNetworkConfiguration and GetDeviceInformation • added current API version to introduction
2.1	2022-10-19	• API version set version to 3.0

Continued on next page

Table 1.1 – continued from previous page

2.2	2022-11-18	• Extend VpnMschapConfigurationRequest with mschapClientIdType and ..IdText Extend VpnTlsConfigurationRequest with tlsClientIdType • SetDeviceNameRequest: restrict devicename and idleMessage pattern to ASCII/ ASCII+DE646
2.3	2022-12-07	• remove syntax highlighting from schemata boxes • Fix API version to 3.0 • Fix typos • Remove unimplemented mschapPasswordSet from GetVpnMschapConfigurationResponse
2.4	2023-01-10	• update GetVpnMschapConfiguration • update GetNetworkConfiguration
2.5	2023-01-02	• Update field translations in Invalid input error messages • Document various timeout units
2.6	2023-11-06	• bump API version to 3.1 • add note on version number implications in section GetApiVersion • add new API messages for kiosk-mode
2.7	2025-03-12	• bump API version to 3.2 • ST1506: add GetWireguardInformation, GetWireguardConfiguration and SetWireguardConfiguration • add new API messages for NTP configuration • extend Get/SetVpnConfiguration/Inform.
2.8	2025-03-25	• change Theobroma/202x to Cherry/2025

2 Introduction

The remote management interface of the Cobra terminal provides the ability to interact with the device in terms of obtaining system information and changing configuration data without physical presence of an administrator by providing a network socket interface.

The remote management interface can be accessed by a simple web application running in a browser or any other client application. The communication with the device is always secured by an authenticated TLS connection.

This document specifies the messaging interface of the remote management interface and the protocols and standards which they rely on.

The current version of the application interface is “3.2”.

2.1 Architecture

All remote management related data exchange with components outside of the device is handled by the remote management service. It creates a network server socket (TCP on port 443) and listens for client connections on this.

Based on the TCP server socket the remote management service uses communication encryption via TLS 1.2. For authentication of the used server socket the certificate C.SMKT.AUT of the gSMC-KT will be used.

The server will not perform a client certificate verification. Based on the TLS 1.2 layer the HTTP protocol (resulting in HTTPS) is used for providing the ability to download a client application, which can be run in a browser (based on HTML, CSS and JavaScript).

The client application provided by the remote management service (or any other client application) can interact with the system using a Websocket interface. This Websocket connection is established using a specific HTTP request and needs therefore no additional TCP port (i.e. Websocket connections are established via a HTTPS request to port 443).

Websockets provide a bidirectional message-based connection between server and client. The supported messages are defined in the next chapter of this document. The message exchange involves the following technologies:

- TCP (RFC 793)
- TLS 1.2 (RFC 5246)
- HTTP/1.1 (RFC 7230)
- WebSocket protcol (RFC 6455)
- JSON (RFC 4627)

3 Remote management API

This chapter describes the messages supported by the remote management service. In general message exchange follows the request-response pattern, where the client sends the requests and the remote management service replies with a response.

The connection is created as defined in RFC 6455, with the protocol name *cobra*:

```
new WebSocket(uri, "cobra");
```

4 Message encoding (JSON/Remote management API)

WS supports two types of messages from the client. The distinction is made by the opcode in the WS protocol header (see RFC 6455, Sec. 11.8):

1. *Opcode=1 Text Frame*: JSON encoded messages, and
2. *Opcode=2 Binary Frame*: Binary encoded request messages

Note: Responses from the device are always encoded in JSON format.

4.1 Fragmentation

Since protocol version 1.1 the remote management service supports fragmentation. Messages are fragmented according the websockets RFC (RFC 6455 Sec. 5.4). This adoption is fully compatible with protocol versions prior 1.1. Further, most client WS implementations internally transparently fragment messages if necessary.

4.2 JSON encoded messages

In general messages are encoded using the JavaScript Object Notation (JSON) according to RFC 4627. Messages which do not conform to this specification will be dropped and the socket connection will be closed.

Each message must further follow the following JSON schema (specified using IETF draft for JSON Schema v4[1]):

```
{
  "type": "object",
  "properties": {
    "header": {
      "type": "object",
      "properties": {
        "msgId": {
          "type": "string",
          "pattern": "[0-9a-f]{32}"
        },
        "inReplyToId": {
          "type": "string",
          "pattern": "[0-9a-f]{32}"
        },
        "sessionId": {
          "type": "string",
          "pattern": "[0-9a-f]{32}"
        }
      },
      "required": [ "msgId" ]
    },
    "payloadType": {
      "type": "string",
      "enum": [
        "GetApiVersionRequest",
        "GetApiVersionResponse",
        "LoginRequest",
```

[1] http://json-schema.org/

```
        "LoginResponse",
        "LogoutRequest",
        "LogoutResponse",
        "RebootRequest",
        "RebootResponse",
        "GetDeviceInformationRequest",
        "GetDeviceInformationResponse",
        "GetNetworkConfigurationRequest",
        "GetNetworkConfigurationResponse",
        "SetNetworkConfigurationRequest",
        "SetNetworkConfigurationResponse",
        "SetActiveInterfaceRequest",
        "SetActiveInterfaceResponse",
        "InitiateFwUpdateRequest",
        "InitiateFwUpdateResponse",
        "InitiateFwUpdateFromFileResponse",
        "GetUpdateStateRequest",
        "GetUpdateStateResponse",
        "GetPairingInformationRequest",
        "GetPairingInformationResponse",
        "DeletePairingBlockRequest",
        "DeletePairingBlockResponse",
        "DeletePairingKeyRequest",
        "DeletePairingKeyResponse",
        "SetLoginCredentialsRequest",
        "SetLoginCredentialsResponse",
        "SetEnvironmentRequest",
        "SetEnvironmentResponse",
        "SetDeviceNameRequest",
        "SetDeviceNameResponse",
        "SetKioskModeRequest",
        "SetKioskModeResponse",
        "GetNtpConfigurationRequest",
        "GetNtpConfigurationResponse",
        "SetNtpConfigurationRequest",
        "SetNtpConfigurationResponse",
        "GetVpnInformationRequest",
        "GetVpnInformationResponse",
        "GetVpnConfigurationRequest",
        "GetVpnConfigurationResponse",
        "SetVpnConfigurationRequest",
        "SetVpnConfigurationResponse",
        "GetVpnTlsConfigurationRequest",
        "GetVpnTlsConfigurationResponse",
        "SetVpnTlsConfigurationRequest",
        "SetVpnTlsConfigurationResponse",
        "GetVpnMschapConfigurationRequest",
        "GetVpnMschapConfigurationResponse",
        "SetVpnMschapConfigurationRequest",
        "SetVpnMschapConfigurationResponse",
        "GetWireguardInformationRequest",
        "GetWireguardInformationResponse",
        "GetWireguardConfigurationRequest",
        "GetWireguardConfigurationResponse",
        "SetWireguardConfigurationRequest",
        "SetWireguardConfigurationResponse",
        "SetRemotePinRequest",
        "SetRemotePinResponse",
        "GetServiceAnnouncementRequest",
        "GetServiceAnnouncementResponse",
        "SetServiceAnnouncementRequest",
        "SetServiceAnnouncementResponse",
        "GetDisplaySettingsRequest",
```

```
        "GetDisplaySettingsResponse",
        "SetDisplaySettingsRequest",
        "SetDisplaySettingsResponse",
        "GetPinEntryRequest",
        "GetPinEntryResponse",
        "SetPinEntryRequest",
        "SetPinEntryResponse",
        "GetSicctEnvironmentRequest",
        "GetSicctEnvironmentResponse",
        "SmcbAuthenticationRequest",
        "SmcbAuthenticationResponse",
        "GetPinpadInformationRequest",
        "GetPinpadInformationResponse",
        "SetPinpadConfigurationRequest",
        "SetPinpadConfigurationResponse",
        "DeletePinpadPairingRequest",
        "DeletePinpadPairingResponse"
      ]
    },
    "payload": {
      "type": "object"
    }
  },
  "required": [ "header", "payloadType", "payload" ]
}
```

The field `msgId` should contain a unique message ID, which will be used for correlating request messages and their responses. Response messages must have the `inReplyToId` set to the value of the `msgId` field of their request message. Request messages should not have an `inReplyToId` field, if they have, then their value will be ignored.

All request messages except `LoginRequest` and `GetApiVersion` must have `sessionId` set in the header. The value of a `LoginRequest`‘s or a `GetApiVersion`‘s `sessionId` field (if any) will be ignored. All response messages must not have sessionId set in the header.

The payloadType field defines the type of the payload object and is defined below. In general messages are exchanged using a strict request/response pattern, which is also exposed in the payload type postfix. Every request message has a payload type with a postfix of `Request` and every response has a payload type with a postfix of `Response`.

The payload field contains the actual data corresponding to a message and embedded into an JSON object. Each payload type has its own payload structure. The payload types and their payload object structures are defined below.

Responses might contain an error field; in this case, no other fields are to be expected in the payload. All request fields are required, unless specified otherwise.

Since the message must be parsed all at once, the total message size after defragmentation is restricted to at most 131072 Bytes (128 kilobytes).

4.3 Binary encoded request messages

Binary encoded messages have the form

Offset[bytes]	Size[bytes]	Content
0	4	MAGIC: ‘x00BIN’
4	16	SessionId: Binary coded SessionId
20	16	MessageId: Binary coded MessageId
36	4	TYPE: Message Type
40	PAYLOAD_LEN	Message payload

The `MAGIC` must always be present and equal to 'x00BIN'.

`SessionId` and `MessageId` have the same meaning as defined for JSON messages. Note, however the data is transmitted as byte array, instead of bytes encoded as hex string.

The `TYPE` replaces the payload type defined for JSON encoded messages. For binary messages this is a 32bit encoded integer (big endian). The values are defined in the request section for the corresponding messages.

The value for `PAYLOAD_LEN` is derived from the underlying WS messaging.

Binary encoded request messages can be arbitrarily long, though, depending on the request, further restrictions might apply. This is described at the request level.

Messages which do not conform to this specification will be dropped and the socket connection will be closed.

4.3.1 Request-response messages

Each message exchange follows a strict request/response pattern. Request messages are sent from an external client to the device. Each request performs a specific operation on the device. Afterwards the device will reply with a response message to the client.

All request messages except the `LoginRequest` and `GetApiVersion` message need a valid session ID set in the message header's `sessionId` field.

4.4 Error handling

During processing of messages error conditions can happen. The response to errors is either to close the socket (in case the request message could not be identified) or to provide an error string as part of the response message containing the error condition.

The message processing and thus which error can occur at which point of processing is specified in Section 5.

Most response messages can contain an `error` field, that includes an error message in form of an error string, that reports details in case of an error. If the `error` field is present all other fields are absent. If `error` is absent, all other fields are mandatory.

The table below lists the possible error strings and their interpretation.

Error string	**Interpretation**
Internal error	An internal error occured during processing a valid request. This unlikely error occurs if communication with an internal service timed out or failed.
Invalid session Id	The given session Id is not valid.
Invalid credentials	The given credentials are not valid.
Login disabled	Login currently disabled (too many invalid attempts).
Session store full	Too many concurrent connections.
Invalid input value (xxx)	The value of the field xxx is not valid.

The table below provides the translations of the error strings for a German user interface.

Error string	German error string
Internal error	Interner Fehler
Invalid session Id	Ungültige Session ID
Invalid credentials	Ungültige Zugangsdaten
Login disabled	Login derzeit nicht möglich
Session store full	Session Speicher voll
Invalid input value (xxx)	Ungültiger Eingabewert (xxx)

4.5 GetApiVersion

Encoding: JSON

The GetApiVersion message can be used to obtain the API version.

The payload of the GetApiVersionRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the GetApiVersionResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "apiVersion": {
      "type": "string"
    }
  }
}
```

The `apiVersion` consists of a major and a minor version number. An increase of the major number between two versions may break existing API handlers due to new, removed or changed fields in request messages or other changes in the API handling within the device. A change of the minor version number should not break existing third party software.

The value of the field `apiVersion` is "3.2".

4.6 Login

Encoding: JSON

The Login message can be used to login and receive a session identifier, which is needed for subsequent messages.

The payload of the LoginRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "username": {
      "type": "string",
      "pattern": "^admin$"
    },
    "password": {
      "type": "string",
      "maxLength": 256
    }
```

```
  },
  "required": [ "username", "password" ]
}
```

The maximum length of the `password` fields's value is 256 characters.

The payload of the LoginResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "sessionId": {
      "type": "string",
      "pattern": "[0-9a-f]{32}"
    }
  },
  "required": [ "sessionId" ]
}
```

The returned session ID from the `LoginResponse` message can be used for accessing the functionality offered by other calls.

Each message exchange with a valid session id will reset a timer for the session ID to 5 minutes. After the timer expires the session ID becomes invalid. The session becomes also invalid when the system reboots.

If the sessionId field within the message header is present and valid, then old session is invalidated before creating a new session.

The service handles at most one simultanous session. In this case the error "Session store full" is returned.

Note that `LoginRequest` messages, which are not successful because of invalid credentials increase a persistent error counter. In case the counter exceeds certain thresholds, the login function is disabled for the following time:

- error counter within interval [3,6]: login disabled for 1 minute
- error counter within interval [7,10]: login disabled for 10 minutes
- error counter within interval [11,20]: login disabled for 1 hour
- error counter within interval greather than 20: login disabled for 1 day

After a successful login request the error counter is reset to zero.

The list below enumerates the possible error messages of the LoginResponse message:

- "Invalid credentials"
- "Session store full"
- "Invalid input value (username)" (German: Benutzername)
- "Invalid input value (password)" (German: Passwort)
- "Login disabled"
- "Internal error"

4.7 Logout

Encoding: JSON

The Logout message can be used to invalidate a session ID obtained by a Login message.

The payload of the LogoutRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the LogoutResponse message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

There are no reported errors for this message.

4.8 Reboot (and factory reset)

Encoding: JSON

The Reboot message can be used to reboot the system.

- If the optional field `adminPassword` is absent the system reboots.
- If the optional field `adminPassword` is present and if it matches the Webadmin password than the system also executes a factory reset during the reboot.

Note that after the reboot the session ID is not valid any more. Device will reboot immediately, a response will be received only if reboot fails.

The payload of the RebootRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "adminPassword": {
      "type": "string",
      "maxLength": 256
    }
  }
}
```

The payload of the RebootResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the RebootResponse message:

- “Invalid credentials”
- “Invalid session Id”
- “Invalid input value (adminPassword)” (German: Admin Passwort)
- “Login disabled”

- “Internal error”

4.9 GetDeviceInformation

Encoding: JSON

The GetDeviceInformation message can be used to obtain information about the systems hardware and firmware versions, as well as current network status.

The payload of the GetDeviceInformationRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the GetDeviceInformationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "providerId": {
      "type": "string"
    },
    "productShortName": {
      "type": "string"
    },
    "hwVersion": {
      "type": "string"
    },
    "productType": {
      "type": "string"
    },
    "productTypeVersion": {
      "type": "string"
    },
    "fwVersion": {
      "type": "string"
    },
    "fwGroupVersion": {
      "type": "string"
    },
    "buildVersion": {
      "type": "string"
    },
    "environment": {
      "type": "string"
    },
    "activeInterface": {
      "type": "string"
    },
    "useDhcp": {
      "type": "boolean"
    },
    "networkMode": {
      "type": "string",
      "enum": ["Dhcp", "StaticIp", "Rfc3927"]
    },
    "ipAddress": {
```

```
      "type": "string",
      "pattern": "(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))|(^$)"
    },
    "ipNetmask": {
      "type": "string",
      "pattern": "(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))|(^$)"
    },
    "dns1": {
      "type": "string",
      "pattern": "(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))|(^$)"
    },
    "dns2": {
      "type": "string",
      "pattern": "(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))|(^$)"
    },
    "gateway": {
      "type": "string",
      "pattern": "(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))|(^$)"
    },
    "deviceName": {
      "type": "string"
    },
    "idleScreenMessage": {
      "type": "string"
    },
    "kioskMode": {
      "type": "boolean"
    },
    "remotepin1": {
      "type": "integer"
    },
    "remotepin2": {
      "type": "integer"
    },
    "remotepin3": {
      "type": "integer"
    },
    "remotepin4": {
      "type": "integer"
    },
    "bootloaderVersion": {
      "type": "string"
    },
    "ethernetMacAddress": {
      "type": "string",
      "pattern": "([0-9a-f]{2})(:[0-9a-f]{2}){5}"
    },
    "usbMacAddress": {
      "type": "string",
      "pattern": "([0-9a-f]{2})(:[0-9a-f]{2}){5}"
    },
    "usbHostMacAddress": {
      "type": "string",
      "pattern": "([0-9a-f]{2})(:[0-9a-f]{2}){5}"
    },
    "ntpSyncStatus": {
      "type": "string",
      "enum": ["stopped", "nosync", "init", "step", "synced", "unknown"]
```

```
    },
    "ntpSyncTimestamp": {
      "type": "integer"
    },
    "ntpSyncServer": {
      "pattern": "(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))|(^$)"
    },
    "sicctPeerIp": {
      "type": "string",
      "pattern": "(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))|(^$)"
    },
    "smcktProductTypeVersion": {
      "type": "string"
    },
    "smcktSerialNumber": {
      "type": "string"
    },
    "smcktPersonalization": {
      "type": "string",
      "pattern": "(RSA)|(ECC)|(RSA,ECC)|(^$)"
    },
    "smcktExpirationDateAUT": {
      "type": "string",
      "pattern" : "([0-9]{8})|(^$)",
      "format" : "yyyyMMdd"
    },
    "smcktExpirationDateAUT2": {
      "type": "string",
      "pattern" : "([0-9]{8})|(^$)",
      "format" : "yyyyMMdd"
    },
    "smcktExpirationDateAUTD": {
      "type": "string",
      "pattern" : "([0-9]{8})|(^$)",
      "format" : "yyyyMMdd"
    }
  }
}
```

The property `networkMode` is only available for API versions since 3.0. The property `kioskMode` is only available for API versions since 3.1. The properties `sicctPeerIp`, `ntpSyncStatus`, `ntpSyncTimestamp` and `ntpSyncServer` are only available for API versions since 3.2.

The property value `ntpSyncServer` is only set to a valid IP address if the system time of the device is synchronized with a NTP server. It is reset when the connection to the server is lost. The `ntpSyncTimestamp` is set to the time when the last valid NTP sync packet was received as UNIX timestamp in seconds. If synchronization to a NTP server was not yet established the value is 0. The value is not reset when the connection to NTP server is lost.

The property value `sicctPeerIp` is only set to a valid IP address if a SICCT connection is active otherwise the property is empty.

The list below enumerates the possible error messages of the GetDeviceInformationResponse message:

- “Invalid session Id”
- “Internal error”

4.10 GetSicctEnvironment

Encoding: JSON

The GetSicctEnvironment message can be used to obtain information about the certificates stored for the Sicct regions of trust.

The field `which` is a bit field denoting the environment information to retrieve.

- Bit 0 (0x01): If present the PU environment is sent within the `pu` object within the response
- Bit 1 (0x02): If present the RU environment is sent within the `ru` object within the response
- Bit 2 (0x04): If present the TU environment is sent within the `tu` object within the response
- Bit 3 (0x08): If present the SICCTLITE environment is sent within the `sicctlite` object within the response (API versions since 3.0)

If `which` is absent the value defaults to 0x07.

The payload of the GetSicctEnvironmentRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "which": {
      "type": "integer",
      "minimum": 1,
      "maximum": 15
    }
  }
}
```

The payload of the GetSicctEnvironmentResponse message contains the following fields:

```
{
    "type": "object",
    "properties": {
        "error": {
            "type": "string"
        },
        "pu": {
            "type": "object",
            "properties": {
                "version": {
                    "type": "string"
                },
                "certs": {
                    "type": "array",
                    "items": {
                        "type": "object",
                        "properties": {
                            "name": {
                                "type": "string"
                            },
                            "fingerprint": {
                                "type": "string",
                                "pattern": "^[0-9A-F]{64}$"
                            }
                        },
                        "required": [
                            "name",
```

```
                        "fingerprint"
                    ]
                }
            }
        },
        "required": [
            "version",
            "certs"
        ]
    },
    "ru": {
        "type": "object",
        "properties": {
            "version": {
                "type": "string"
            },
            "certs": {
                "type": "array",
                "items": {
                    "type": "object",
                    "properties": {
                        "name": {
                            "type": "string"
                        },
                        "fingerprint": {
                            "type": "string",
                            "pattern": "^[0-9A-F]{64}$"
                        }
                    },
                    "required": [
                        "name",
                        "fingerprint"
                    ]
                }
            }
        },
        "required": [
            "version",
            "certs"
        ]
    },
    "tu": {
        "type": "object",
        "properties": {
            "version": {
                "type": "string"
            },
            "certs": {
                "type": "array",
                "items": {
                    "type": "object",
                    "properties": {
                        "name": {
                            "type": "string"
                        },
                        "fingerprint": {
                            "type": "string",
                            "pattern": "^[0-9A-F]{64}$"
                        }
                    },
                    "required": [
                        "name",
                        "fingerprint"
```

```
                        ]
                    }
                }
            },
            "required": [
                "version",
                "certs"
            ]
        },
        "sicctlite": {
            "type": "object",
            "properties": {
                "version": {
                    "type": "string"
                },
                "certs": {
                    "type": "array",
                    "items": {
                        "type": "object",
                        "properties": {
                            "name": {
                                "type": "string"
                            },
                            "fingerprint": {
                                "type": "string",
                                "pattern": "^[0-9A-F]{64}$"
                            }
                        },
                        "required": [
                            "name",
                            "fingerprint"
                        ]
                    }
                }
            },
            "required": [
                "version",
                "certs"
            ]
        }
    }
}
```

The list below enumerates the possible error messages of the GetSicctEnvironmentResponse message:

- “Invalid session Id”
- “Internal error”

4.11 SetActiveInterface

Encoding: JSON

The SetActiveInterface message can be used to set the active network interface of the device to either “ethernet”, “usbRndis” or “usbCdcEcm”. After setting the configuration, the network service and remote management service are restarted.

The payload of the SetActiveInterfaceRequest contains the following fields:

```
{
  "type": "object",
  "properties": {
```

```
    "activeInterface": {
      "type": "string",
      "pattern": "^(ethernet|usbRndis|usbCdcEcm)$"
    }
  },
  "required": [ "activeInterface" ]
}
```

The payload of the SetActiveInterfaceResponse contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the SetActiveInterfaceResponse message:

- "Invalid session Id"
- "Invalid input value (activeInterface)" (German: Aktive Schnittstelle)
- "Internal error"

4.12 SetEnvironment

Encoding: JSON

The SetEnvironment message can be used to set the operational environment of the device to either `PU` (Produktivumgebung), `RU` (Referenzumgebung), or `TU` (Testumgebung).

The payload of the SetEnvironmentRequest contains the following fields:

```
{
  "type": "object",
  "properties": {
    "environment": {
      "type": "string",
      "pattern": "^(PU|RU|TU)$"
    }
  },
  "required": [ "environment" ]
}
```

The payload of the SetEnvironmentResponse contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the SetEnvironmentResponse message:

- "Invalid session Id"

- “Invalid input value (environment)” (German: Vertrauensraum)
- “Internal error”

4.13 SetDeviceName

Encoding: JSON

The SetDeviceName message can be used to set the device name and the idle screen message.

The payload of the SetDeviceNameRequest contains the following fields:

```
{
  "type": "object",
  "properties": {
    "deviceName": {
      "type": "string",
      "pattern": "^[ -~]{1,32}$"
    },
    "idleScreenMessage": {
      "type": "string",
      "pattern": "^[ -~ÄäÖöÜüß§]{1,32}$"
    }
  },
  "required": [ "deviceName", "idleScreenMessage" ]
}
```

The payload of the SetDeviceNameResponse contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the SetDeviceNameResponse message:

- “Invalid session Id”
- “Invalid input value (deviceName)” (German: Gerätename)
- “Invalid input value (idleScreenMessage)” (German: Ruhebildschirmtext)
- “Internal error”

4.14 SetKioskMode

Encoding: JSON

The SetKioskMode message can be used to enable or disable the kiosk mode (API versions since 3.1). If kiosk mode is enabled, the icon to access the menu on the device is no longer shown on the display and the user is not able to change the settings of the device.

The payload of the SetKioskModeRequest contains the following fields:

```
{
  "type": "object",
  "properties": {
    "kioskMode": { "type": "boolean" }
  },
```

```
  "required": ["kioskMode"]
}
```

The payload of the SetKioskModeResponse contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the SetKioskModeResponse message:

- “Invalid session Id”
- “Internal error”

4.15 GetNetworkConfiguration

Encoding: JSON

The GetNetworkConfiguration message can be used to obtain information about the network configuration of the device.

The payload of the GetNetworkConfigurationRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "interface": {
      "type": "string",
      "pattern": "^(ethernet|usbRndis|usbCdcEcm)$"
    }
  },
  "required": [ "interface" ]
}
```

The payload of the GetNetworkConfigurationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "interface": {
      "type": "string"
    },
    "useDhcp": {
      "type": "boolean"
    },
    "networkMode": {
      "type": "string",
      "enum": ["Dhcp", "StaticIp", "Rfc3927"]
    },
    "ipAddress": {
      "type": "string",
      "pattern": "(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))|(^$)"
    },
    "ipNetmask": {
      "type": "string",
```

```
      "pattern": "(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))|(^$)"
    },
    "dns1": {
      "type": "string",
      "pattern": "(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))|(^$)"
    },
    "dns2": {
      "type": "string",
      "pattern": "(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))|(^$)"
    },
    "gateway": {
      "type": "string",
      "pattern": "(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))|(^$)"
    }
  }
}
```

The property `networkMode` is only available for API versions since 3.0.

The list below enumerates the possible error messages of the GetNetworkConfigurationResponse message:

- “Invalid session Id”
- “Invalid input value (interface)” (German: Netzwerkschnittstelle)
- “Internal error”

4.16 SetNetworkConfiguration

Encoding: JSON

The SetNetworkConfiguration message can be used to set the network configuration of the device.

The payload of the SetNetworkConfigurationRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "interface": {
      "type": "string",
      "pattern": "^(ethernet|usbRndis|usbCdcEcm)$"
    },
    "useDhcp": {
      "type": "boolean"
    },
    "networkMode": {
      "type": "string",
      "enum": ["Dhcp", "StaticIp", "Rfc3927"]
    },
    "ipAddress": {
      "type": "string",
      "pattern": "^(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))$|(^$)"
    },
    "ipNetmask": {
      "type": "string",
      "pattern": "^(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))$|(^$)"
```

```
    },
    "dns1": {
      "type": "string",
      "pattern": "^(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))$|(^$)"
    },
    "dns2": {
      "type": "string",
      "pattern": "^(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))$|(^$)"
    },
    "gateway": {
      "type": "string",
      "pattern": "^(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9]))$|(^$)"
    }
  },
  "oneOf": [
    { "required": [ "useDhcp" ] },
    { "required": [ "networkMode" ] }
  ],
  "required": ["interface", "ipAddress", "ipNetmask",
    "dns1", "dns2", "gateway"]
}
```

The property `networkMode` exists for API versions since 3.0. Either `networkMode` or `useDhcp` must be present in the request message.

The payload of the SetNetworkConfigurationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The property `networkMode` exists for API versions since 3.0. In case of `network_mode` `Rfc3927`, the property `useDhcp` is set to `false`.

The list below enumerates the possible error messages of the SetNetworkConfigurationResponse message:

- “Invalid session Id”
- “Invalid input value (interface)” (German: Netzwerkschnittstelle)
- “Invalid input value (networkMode)” (German: IP Netzwerk Modus)
- “Invalid input value (ipAddress)” (German: IP Adresse)
- “Invalid input value (ipNetmask)” (German: IP Netzmaske)
- “Invalid input value (dns1)” (German: DNS1)
- “Invalid input value (dns2)” (German: DNS2)
- “Invalid input value (gateway)” (German: Gateway)
- “Internal error”

4.17 GetNtpConfiguration

Encoding: JSON

The GetNetworkConfiguration message can be used to obtain information about the NTP client configuration of the device (API versions since 3.2).

The payload of the GetNtpConfigurationRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the GetNtpConfigurationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "active": {
      "type": "string",
      "pattern": "^(on|off)$"
    },
    "serverList": {
      "type": "string",
      "pattern": "^(([-a-zA-Z0-9._:]{0,255}){1}(,[-a-zA-Z0-9._:]{0,255}){0,7})$|(^$)
↪"
    },
    "minPollInterval": {
      "type": "integer",
      "minimum": 3,
      "maximum": 16
    },
    "maxPollInterval": {
      "type": "integer",
      "minimum": 4,
      "maximum": 17
    }
  }
}
```

The list below enumerates the possible error messages of the GetNtpConfigurationResponse message:

- "Invalid session Id"
- "Internal error"

4.18 SetNtpConfiguration

Encoding: JSON

The SetNetworkConfiguration message can be used to set the NTP client configuration of the device (API versions since 3.2).

The payload of the SetNtpConfigurationRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
```

```
    "active": {
      "type": "string",
      "pattern": "^(on|off)$"
    },
    "serverList": {
      "type": "string",
      "pattern": "^(([-a-zA-Z0-9._:]{0,255}){1}(,[-a-zA-Z0-9._:]{0,255}){0,7})$|(^$)
↪"
    },
    "minPollInterval": {
      "type": "integer",
      "minimum": 3,
      "maximum": 16
    },
    "maxPollInterval": {
      "type": "integer",
      "minimum": 4,
      "maximum": 17
    }
  },
  "required": [ "active", "serverList" ]
}
```

The payload of the SetNtpConfigurationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The properties `minPollInterval` and `maxPollInterval` are optional in general but if one is present the other must also be set. The value of `minPollInterval` must always be less than the value of `maxPollInterval`.

The list below enumerates the possible error messages of the SetNtpConfigurationResponse message:

- “Invalid session Id”
- “Invalid input value (active)” (German: Aktiv)
- “Invalid input value (serverList)” (German: Server-Liste)
- “Invalid input value (pollInterval)” (German: Poll-Intervall)
- “Internal error”

4.19 GetVpnInformation

Encoding: JSON

The GetVpnInformation message can be used to obtain information about the VPN configuration and status of the device.

The payload of the GetVpnInformationRequest message is empty:

```
{
  "type": "object",
  "properties": {
```

```
  }
}
```

The payload of the GetVpnInformationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "useVpn": {
      "type": "string",
      "pattern": "^(off|tls|mschap)$"
    },
    "serverIpAddress": {
      "type": "string",
      "pattern": "(^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9])$)|(^$)"
    },
    "serverCertFp": {
      "type": "string",
      "pattern": "(^[0-9A-F]{2}(:[0-9A-F]{2}){31}$)|(^$)"
    },
    "clientCertFp": {
      "type": "string",
      "pattern": "(^[0-9A-F]{2}(:[0-9A-F]{2}){31}$)|(^$)"
    },
    "caCertFp": {
      "type": "string",
      "pattern": "(^[0-9A-F]{2}(:[0-9A-F]{2}){31}$)|(^$)"
    },
    "serverCertSubject": {
      "type": "string"
    },
    "clientCertSubject": {
      "type": "string"
    },
    "caCertSubject": {
      "type": "string"
    },
    "mschapUsername": {
      "pattern": "(^[^\\t\\r\\n\\v\\f]{0,511}[^\\t\\r\\n\\v\\f ]$)|(^$)",
      "type": "string"
    },
    "crlUri": {
      "type": "string",
      "pattern": "^((https?://[^\\\\,' ]+){1}(,(https?://[^\\\\,' ]+){1}){0,})$|(^$)
↪",
      "maxLength": 102400
    },
    "dpdDelay": {
      "type": "integer",
      "minimum": 0,
      "maximum": 65535
    },
    "dpdTimeout": {
      "type": "integer",
      "minimum": 0,
      "maximum": 65535
    },
    "status": {
      "type": "string",
```

```
      "pattern" : "^(up|down|off)$|(^$)"
    },
    "localIp": {
      "type": "string",
      "pattern": "(^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9])$)|(^$)"
    },
    "webinterfaceAlwaysOn": {
      "type": "boolean"
    },
    "localBypassRoutes": {
      "type": "string",
      "pattern": "^(auto|((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-
↪5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])/(3[0-2]|[1-2]{0,1}[0-9]))(,((25[0-5]|(2[0-4]|1
↪{0,1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])/(3[0-
↪2]|[1-2]{0,1}[0-9])){0,3}$|^$"
    }
  }
}
```

The list below enumerates the possible error messages of the GetVpnInformationResponse message:

- “Invalid session Id”
- “Internal error”

4.20 GetVpnConfiguration

Encoding: JSON

The GetVpnConfiguration message can be used to obtain information about the basic VPN and Wireguard (Wireguard for API version since 3.2) configuration.

The payload of the GetVpnConfigurationRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the GetVpnConfigurationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "useVpn": {
      "type": "string",
      "pattern": "^(off|tls|mschap|wireguard)$"
    },
    "serverIpAddress": {
      "type": "string",
      "pattern": "(^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9])$)|(^$)"
    },
    "localBypassRoutes": {
      "type": "string",
      "pattern": "^(auto|((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-
↪5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])/(3[0-2]|[1-2]{0,1}[0-9]))(,((25[0-5]|(2[0-4]|1
↪{0,1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])/(3[0-
↪2]|[1-2]{0,1}[0-9])){0,3}$|^$"
```

```
    },
    "caCert": {
      "type": "string",
      "pattern": "(^-----BEGIN [A-Z ]{0,16}CERTIFICATE-----(\n|\r|\r\n)([0-9a-zA-Z+/
↪=]{64}(\n|\r|\r\n))*([0-9a-zA-Z+/=]{1,63}(\n|\r|\r\n))?-----END [A-Z ]{0,16}
↪CERTIFICATE-----( |\n|\r|\r\n){0,4}$)|($^)",
      "minLength": 32,
      "maxLength": 102400
    },
    "dpdDelay": {
      "type": "integer",
      "minimum": 0,
      "maximum": 65535
    },
    "dpdTimeout": {
      "type": "integer",
      "minimum": 0,
      "maximum": 65535
    },
    "crlUri": {
      "type": "string",
      "pattern": "^((https?://[^\\\\,' ]+){1}(,(https?://[^\\\\,' ]+){1}){0,})$|(^$)
↪",
      "maxLength": 102400
    },
    "wireguardPeerAddress": {
      "type": "string",
      "pattern": "^[-a-zA-Z0-9._:]{0,255}$"
    },
    "wireguardPeerPublicKey": {
      "type": "string",
      "pattern": "^[a-zA-Z0-9+/]{43}=$"
    },
    "wireguardPersistentKeepaliveInterval": {
      "type": "integer",
      "minimum": 0,
      "maximum": 65535
    },
    "wireguardAllowedIps": {
      "type": "string",
      "pattern": "^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9])(/(3[0-2]|[1-2]{0,1}[0-9])){0,1}(,((25[0-5]|(2[0-4]|1{0,
↪1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])(/(3[0-2]|[1-
↪2]{0,1}[0-9])){0,1}){0,7}$|^$"
    },
    "wireguardClientIp": {
      "type": "string",
      "pattern": "^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9])(/(3[0-2]|[1-2]{0,1}[0-9])){0,1}$|^$"
    },
    "wireguardDnsIps": {
      "pattern": "^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9])(,((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.){3,3}
↪(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])){0,5}$|(^$)",
      "type": "string"
    },
    "wireguardMtuSize": {
      "type": "integer",
      "minimum": 0,
      "maximum": 65535
    },
    "webinterfaceAlwaysOn": {
      "type": "boolean"
```

```
    }
  }
}
```

The list below enumerates the possible error messages of the GetVpnConfigurationResponse message:

- “Invalid session Id”
- “Internal error”

4.21 SetVpnConfiguration

Encoding: JSON

The SetVpnConfiguration message can be used to set either the basic VPN or Wireguard (Wireguard for API version since 3.2) configuration. The unit for dpdDelay, dpdTimeout and wireguardPersistentKeepaliveInterval is seconds.

The payload of the SetVpnConfigurationRequest message contains the following fields:

```
{
  "anyOf": [
    {
      "type": "object",
      "properties": {
        "useVpn": {
          "type": "string",
          "pattern": "^(off|tls|mschap|wireguard)$"
        }
      },
      "required": [
        "useVpn"
      ]
    },
    {
      "oneOf": [
        {
          "type": "object",
          "properties": {
            "serverIpAddress": {
              "type": "string",
              "pattern": "(^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-
↪5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])$)|(^$)"
            },
            "caCert": {
              "type": "string",
              "pattern": "(^-----BEGIN [A-Z ]{0,16}CERTIFICATE-----(\n|\r|\r\n)([0-
↪9a-zA-Z+/=]{64}(\n|\r|\r\n))*([0-9a-zA-Z+/=]{1,63}(\n|\r|\r\n))?-----END [A-Z ]{0,
↪16}CERTIFICATE-----( |\n|\r|\r\n){0,4}$)|($^)",
              "maxLength": 102400
            },
            "dpdDelay": {
              "type": "integer",
              "minimum": 0,
              "maximum": 65535
            },
            "dpdTimeout": {
              "type": "integer",
              "minimum": 0,
              "maximum": 65535
            },
            "crlUri": {
```

```
              "type": "string",
              "pattern": "^((https?://[^\\\\,' ]+){1}(,(https?://[^\\\\,' ]+){1}){0,
↪})$|(^$)",
              "maxLength": 102400
            },
            "webinterfaceAlwaysOn": {
              "type": "boolean"
            },
            "localBypassRoutes": {
              "type": "string",
              "pattern": "^(auto|((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.){3,3}
↪(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])/(3[0-2]|[1-2]{0,1}[0-9]))(,((25[0-
↪5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-
↪9])/(3[0-2]|[1-2]{0,1}[0-9])){0,3}$|^$"
            }
          },
          "required": [
            "useVpn",
            "serverIpAddress",
            "caCert",
            "dpdDelay",
            "dpdTimeout",
            "crlUri"
          ]
        },
        {
          "type": "object",
          "properties": {
            "wireguardPeerAddress": {
              "type": "string",
              "pattern": "^[-a-zA-Z0-9._:]{0,255}$"
            },
            "wireguardPeerPublicKey": {
              "type": "string",
              "pattern": "^[a-zA-Z0-9+/]{43}=$|^$"
            },
            "wireguardPersistentKeepaliveInterval": {
              "type": "integer",
              "minimum": 0,
              "maximum": 65535
            },
            "wireguardAllowedIps": {
              "type": "string",
              "pattern": "^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-
↪5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])(/(3[0-2]|[1-2]{0,1}[0-9])){0,1}(,((25[0-
↪5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-
↪9])(/(3[0-2]|[1-2]{0,1}[0-9])){0,1}){0,7}$|^$"
            },
            "wireguardClientIp": {
              "type": "string",
              "pattern": "^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-
↪5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])(/(3[0-2]|[1-2]{0,1}[0-9])){0,1}$|^$"
            },
            "wireguardDnsIps": {
              "pattern": "^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.){3,3}(25[0-
↪5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])(,((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\\.)
↪{3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])){0,5}$|(^$)",
              "type": "string"
            },
            "wireguardMtuSize": {
              "type": "integer",
              "minimum": 0,
              "maximum": 65535
```

```
            },
            "webinterfaceAlwaysOn": {
              "type": "boolean"
            }
          },
          "required": [
            "wireguardPeerAddress",
            "wireguardPeerPublicKey",
            "wireguardPersistentKeepaliveInterval",
            "wireguardAllowedIps",
            "wireguardClientIp",
            "wireguardDnsIps",
            "wireguardMtuSize"
          ]
        }
      ]
    }
  ]
}
```

- `useVpn` is optional (optional for API version since 3.2) and if it is contained in the message, it will lead to a restart of the network system and services. Transmitting only a Ipsec or Wireguard configuration will not automatically change a currently running VPN session therefore.
- `webinterfaceAlwaysOn` is optional (API version since 3.2). If set to true, the webservice will be available:
  - locally in case VPN or Wireguard are configured, but the connection is not yet established.
  - locally in case VPN or Wireguard connections are established.

  If set to false, the webservice is only available over the established secure connection if VPN or Wireguard are configured.
- `localBypassRoutes` is optional (API version since 3.2) and allows a list of up to 4 address ranges that are not routed over the VPN tunnel (bypass routes to local network). If the placeholder *auto* is used as one of the entries, the currently set network IP/Netmask will also be added as bypass route. Accessing the webservice from the local network during an established VPN connection needs a correct bypass route to be set.
- `wireguardAllowedIps` requires a valid address range format to be accepted (API version since 3.2). The base IP value must contain zeroes corresponding to the given address range (eg. 172.88.23.0/24).
- `wireguardPersistentKeepaliveInterval` sets the timespan in seconds for periodic keep alive packets sent to the server for keeping up NAT routes (API version since 3.2). Set to zero to turn off.
- `wireguardMtuSize` sets the MTU size on the wireguard interface (API version since 3.2). If set to zero, its automatically chosen by the system (1420 for ethernet connections). If not zero, it has to be set greater than 127.

The payload of the SetVpnConfigurationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the SetVpnConfigurationResponse message:

- “Invalid session Id”
- “Invalid input value (useVpn)” (German: VPN Konfiguration)
- “Invalid input value (serverIpAddress)” (German: Server IP Adresse)
- “Invalid input value (crlUri)” (German: CRL URI Liste)
- “Invalid input value (caCert)” (German: CA Zertifikat)
- “Invalid input value (dpdDelay)”
- “Invalid input value (dpdTimeout)”
- “Invalid input value (wireguardPeerAddress)” (German: Server/Peer IP Adresse)
- “Invalid input value (wireguardPeerPublicKey)” (German: Server/Peer öffentlicher Schlüssel)
- “Invalid input value (wireguardPersistentKeepaliveInterval)” (German: Keep alive Zeitspanne)
- “Invalid input value (wireguardAllowedIps)” (German: Zugelassene IP Adressbereiche)
- “Invalid input value (wireguardClientIp)” (German: Terminal/Client IP Adresse)
- “Invalid input value (wireguardDnsIps)” (German: DNS IP Adressen)
- “Invalid input value (wireguardMtuSize)” (German: MTU)
- “Invalid input value (localBypassRoutes)” (German: Erlaubte lokale Routen)
- “Internal error”

4.22 GetVpnTlsConfiguration

Encoding: JSON

The GetVpnTlsConfiguration message can be used to obtain information about the TLS related VPN configuration.

The payload of the GetVpnTlsConfigurationRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the GetVpnTlsConfigurationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "clientCert": {
      "type": "string",
      "pattern": "(^-----BEGIN [A-Z ]{0,16}CERTIFICATE-----(\n|\r|\r\n)([0-9a-zA-Z+/
↪=]{64}(\n|\r|\r\n))*([0-9a-zA-Z+/=]{1,63}(\n|\r|\r\n))?-----END [A-Z ]{0,16}
↪CERTIFICATE-----( |\n|\r|\r\n){0,4}$)|($^)",
      "minLength": 32,
      "maxLength": 102400
    },
```

```
    "tlsClientIdType": {
      "type": "string",
      "pattern": "^(fingerprint|subject)$"
    }
  }
}
```

The list below enumerates the possible error messages of the GetVpnTlsConfigurationResponse message:

- “Invalid session Id”
- “Internal error”

4.23 SetVpnTlsConfiguration

Encoding: JSON

The SetVpnTlsConfiguration message can be used to set the TLS related VPN configuration.

The payload of the SetVpnTlsConfigurationRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "clientCert": {
      "type": "string",
      "pattern": "(^-----BEGIN [A-Z ]{0,16}CERTIFICATE-----(\n|\r|\r\n)([0-9a-zA-Z+/
↪=]{64}(\n|\r|\r\n))*([0-9a-zA-Z+/=]{1,63}(\n|\r|\r\n))?-----END [A-Z ]{0,16}
↪CERTIFICATE-----( |\n|\r|\r\n){0,4}$)|($^)",
      "maxLength": 102400
    },
    "clientCertPrivateKey": {
      "type": "string",
      "pattern": "(^-----BEGIN [A-Z ]{0,16}PRIVATE KEY-----(\n|\r|\r\n)([0-9a-zA-Z+/
↪=]{64}(\n|\r|\r\n))*([0-9a-zA-Z+/=]{1,63}(\n|\r|\r\n))?-----END [A-Z ]{0,16}
↪PRIVATE KEY-----( |\n|\r|\r\n){0,4}$)|($^)",
      "maxLength": 102400
    },
    "tlsClientIdType": {
      "type": "string",
      "pattern": "^(fingerprint|subject)$"
    }
  },
  "required": ["clientCert", "clientCertPrivateKey"]
}
```

The payload of the SetVpnTlsConfigurationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the SetVpnTlsConfigurationResponse message:

- “Invalid session Id”

- “Invalid input value (clientCert)” (German: Client Zertifikat)
- “Invalid input value (clientCertPrivateKey)” (German: Privater Client Zertifikatschlüssel)
- “Invalid input value (tlsClientIdType)” (German: TLS Client ID Modus)
- “VPN client private key doesn’t match certificate key” (German: VPN client Schlüssel passt nicht zum Zertifikat)
- “Internal error”

4.24 GetVpnMschapConfiguration

Encoding: JSON

The GetVpnMschapConfiguration message can be used to obtain information about the MSChapv2 related VPN configuration.

The payload of the GetVpnMschapConfigurationRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the GetVpnMschapConfigurationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "mschapUsername": {
      "type": "string",
      "pattern": "(^[^\\t\\r\\n\\v\\f]{0,511}[^\\t\\r\\n\\v\\f ]$)|(^$)"
    },
    "mschapClientIdType": {
      "type": "string",
      "pattern": "^(off|serialnumber|text)$"
    },
    "mschapClientIdText": {
      "type": "string",
      "pattern": "^[ -~]{1,256}$"
    }
  }
}
```

It is not allowed to set a username without a password, therefore if the field `mschapUsername` contains a value this infers that a password is also set.

The list below enumerates the possible error messages of the GetVpnMschapConfigurationResponse message:

- “Invalid session Id”
- “Internal error”

4.25 SetVpnMschapConfiguration

Encoding: JSON

The SetVpnMschapConfiguration message can be used to set the MSChapv2 related VPN configuration.

It has to be noted that both `mschapUsername` and `mschapPassword` have to be either empty or set to a valid string. It is not allowed to set a username without a password and vice versa. Otherwise an error message is returned.

The payload of the SetVpnMschapConfigurationRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "mschapUsername": {
      "type": "string",
      "pattern": "(^[^\\t\\r\\n\\v\\f]{0,511}[^\\t\\r\\n\\v\\f ]$)|(^$)"
    },
    "mschapPassword": {
      "type": "string",
      "pattern": "(^[^\\t\\r\\n\\v\\f]{0,511}[^\\t\\r\\n\\v\\f ]$)|(^$)"
    },
    "mschapClientIdType": {
      "type": "string",
      "pattern": "^(off|serialnumber|text)$"
    },
    "mschapClientIdText": {
      "type": "string",
      "pattern": "^[ -~]{1,256}$"
    }
  },
  "required": ["mschapPassword", "mschapUsername"]
}
```

The payload of the SetVpnMschapConfigurationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the SetVpnMschapConfigurationResponse message:

- “Invalid session Id”
- “Invalid input value (mschapUsername)” (German: MSCHAPv2 Benutzername)
- “Invalid input value (mschapPassword)” (German: MSCHAPv2 Passwort)
- “Invalid input value (mschapClientIdType)” (German: MSCHAPv2 Client ID Modus)
- “Invalid input value (mschapClientIdText)” (German: MSCHAPv2 Client ID Text)
- “Internal error”

4.26 GetWireguardInformation

Encoding: JSON

The GetWireguardInformation message can be used to obtain information about the Wireguard configuration and status of the device (API version since 3.2).

The payload of the GetWireguardInformationRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the GetWireguardInformationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "peerAddress": {
      "type": "string",
      "pattern": "(^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9])$)|(^[-a-zA-Z0-9._]{1,255}$)|(^$)"
    },
    "peerPublicKey": {
      "type": "string",
      "pattern": "^[a-zA-Z0-9+/]{43}=$"
    },
    "persistentKeepaliveInterval": {
      "type": "integer",
      "minimum": 0,
      "maximum": 65535
    },
    "allowedIps": {
      "type": "string",
      "pattern": "^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9])(/(3[0-2]|[1-2]{0,1}[0-9])){0,1}(,((25[0-5]|(2[0-4]|1{0,
↪1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])(/(3[0-2]|[1-2]
↪{0,1}[0-9])){0,1}){0,7}$"
    },
    "clientIp": {
      "type": "string",
      "pattern": "^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9])(/(3[0-2]|[1-2]{0,1}[0-9])){0,1}$|^$"
    },
    "dnsIps": {
      "pattern": "^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9])(,((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-
↪5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])){0,5}$|(^$)",
      "type": "string"
    },
    "wireguardMtuSize": {
      "type": "integer",
      "minimum": 0,
      "maximum": 65535
    },
    "status": {
      "type": "string",
      "pattern": "^(up|down|off)$"
    },
    "information": {
      "type": "string"
    },
```

```
    "webinterfaceAlwaysOn": {
      "type": "boolean"
    }
  }
}
```

- `information` optional connection information, if this field contains just a number it represents an error code.

The list below enumerates the possible error messages of the GetWireguardInformationResponse message:

- "Invalid session Id"
- "Internal error"

4.27 GetWireguardConfiguration

Encoding: JSON

The GetWireguardConfiguration message can be used to obtain information about the Wireguard credentials (API version since 3.2). Private and preshared key of the configuration can not be read out of the device.

The payload of the GetWireguardConfigurationRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the GetWireguardConfigurationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "peerPresharedKey": {
      "type": "string",
      "pattern": "(^set$)|(^$)"
    },
    "clientPrivateKey": {
      "type": "string",
      "pattern": "(^set$)|(^$)"
    }
  }
}
```

The list below enumerates the possible error messages of the GetWireguardConfigurationResponse message:

- "Invalid session Id"
- "Internal error"

4.28 SetWireguardConfiguration

Encoding: JSON

The SetWireguardConfiguration message can be used to set the Wireguard client credentials (API version since 3.2). The fields containing keys must be valid base64 encoded values.

The fields "peerPresharedKey" and "clientPrivateKey" can be set to empty strings for clearing them on the device. If they are not contained in the request message, their values will remain unchanged.

The payload of the SetWireguardConfigurationRequest message contains the following fields:

```
{
  "anyOf": [
    {
      "type": "object",
      "properties": {
        "peerPresharedKey": {
          "type": "string",
          "pattern": "^[a-zA-Z0-9+/]{43}=$|^$"
        }
      },
      "required": [
        "peerPresharedKey"
      ]
    },
    {
      "type": "object",
      "properties": {
        "clientPrivateKey": {
          "type": "string",
          "pattern": "^[a-zA-Z0-9+/]{43}=$|^$"
        }
      },
      "required": [
        "clientPrivateKey"
      ]
    }
  ]
}
```

The payload of the SetWireguardConfigurationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the SetWireguardConfigurationResponse message:

- "Invalid session Id"
- "Invalid input value (peerPresharedKey)" (German: gemeinsamer Schlüssel (PSK))
- "Invalid input value (clientPrivateKey)" (German: Terminal/Client privater Schlüssel)
- "Internal error"

4.29 GetDisplaySettings

Encoding: JSON

The GetDisplaySettings message can be used to obtain information about the display configuration. The brightness value is the percentage of the maximum brightness and the screen timeout unit is seconds.

The payload of the GetDisplaySettingsRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the GetDisplaySettingsResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "timeout": {
      "type": "integer",
      "minimum": 0,
      "maximum": 3599
    },
    "brightness": {
      "type": "integer",
      "minimum": 10,
      "maximum": 100
    },
    "touchSound": {
      "type": "string",
      "pattern": "^(on|off)$"
    }
  }
}
```

The list below enumerates the possible error messages of the GetDisplaySettingsResponse message:

- “Invalid session Id”
- “Internal error”

4.30 SetDisplaySettings

Encoding: JSON

The SetDisplaySettings message can be used to configure the display. The brightness value is interpreted as percentage of maximum brightness and the screen timeout unit is seconds.

The payload of the SetDisplaySettingsRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "timeout": {
      "type": "integer",
      "minimum": 0,
      "maximum": 3599
    },
    "brightness": {
      "type": "integer",
```

```
      "minimum": 10,
      "maximum": 100
    },
    "touchSound": {
      "type": "string",
      "pattern": "^(on|off)$"
    }
  },
  "required": [
    "timeout",
    "brightness",
    "touchSound"
  ]
}
```

The payload of the SetDisplaySettingsResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the SetDisplaySettingsResponse message:

- "Invalid session Id"
- "Invalid input value (timeout)" (German: Bildschirm-Timeout)
- "Invalid input value (brightness)" (German: Helligkeit)
- "Invalid input value (touchSound)" (German: Berührungstöne)
- "Internal error"

4.31 InitiateFwUpdate

Encoding: JSON

The InitiateFwUpdate message can be used to initiate a firmware update. The device will reboot after performing the operation. If there is already an update in progress the error "Update in progress" is returned.

An update is complete after the device is running for a grace period of 5 minutes after reboot. During the grace period the device is fully operational, but updates are disabled.

The payload of the InitiateFwUpdateRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "updateUrl": {
      "type": "string",
      "pattern": "^https?://.+",
      "maxLength": 256
    }
  },
  "required": [ "updateUrl" ]
}
```

The payload of the InitiateFwUpdateResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the InitiateFwUpdateResponse message:

- "Invalid session Id"
- "Update in progress" (German: Updatevorgang bereits aktiv)
- "Invalid input value (updateUrl)" (German: Update URL)
- "Internal error"

4.32 InitiateFwUpdateFromFile

Encoding: Binary

Binary TYPE:

- 1: FILE_CHUNK (with more to come)
- 2: LAST_FILE_CHUNK

The InitiateFWUpdateFromFile message can be used to initiate a firmware update. The device will reboot after performing the operation. If there is already an update in progress the error "Update in progress" is returned.

An update is complete after the device is running for a grace period of 5 minutes after reboot. During the grace period the device is fully operational, but updates are disabled.

While InitiateFwUpdate is used to transmit an URL from where the device shall download the update file, InitiateFwUpdateFromFile uses binary encoded messages to push the file onto the device. After receiving all chunks the device starts the firmware update in the same way as defined by InitiateFwUpdate.

The update file is transmitted as a series of chunks of arbitrary size. The last chunk has the message type *2*, all other chunks have the message type *1*. It is allowed that the first chunk is also the last chunk.

After receiving a chunk the device responds with a InitiateFwUpdateFromFileResponse.

The payload of the InitiateUpdateFromFileRequest contains a part of the firmware update file. All chunks **MUST** be transmitted in order, and within a single websockets session. Concurring InitiateFwUpdateFromFileRequest imessages in other websockets sessions are considered as a separate dedicated update process, and will return the *"Update in progress"* error.

The firmware update is executed under the following condition:

1. The InitiateFwUpdateFromFileRequest with message type *2* was received and processed without error.

The firmware update is canceled under the following conditions:

1. The InitiateFwUpdateFromFileResponse indicates some error.
2. The client disconnects the websocket.

The payload of the InitiateFwUpdateResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the InitiateFwUpdateFromFileResponse message:

- “Invalid session Id”
- “Update in progress” (German: Updatevorgang bereits aktiv)
- “Internal error”

4.33 GetUpdateState

Encoding: JSON

The GetUpdateState message can be used to get the status about ongoing firmware updates.

Note: After reboot the current status is cleared unless it indicates a grace period.

The payload of the GetUpdateStateRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the GetUpdateStateResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "state": {
      "type": "string"
    }
  }
}
```

If the request succeeds without error the state field contains one of the following strings:

- “Idle”: Update engine is idle and awaits InitiateFwUpdateRequests.
- “Idle (Invalid signature)”: Last update failed with an invalid signature. This state is cleared to “Idle” on reboot. Update engine is idle and awaits InitiateFwUpdateRequests.
- “Idle (Invalid version)”: Last update failed with an invalid version. This state is cleared to “Idle” on reboot. Update engine is idle and awaits InitiateFwUpdateRequests.
- “Busy (Remote management)”: Update engine is currently downloading the image passed by an InitiateFwUpdateRequest.
- “Busy (Remote management upload)”: Update engine is currently downloading the image passed by an InitiateFwUpdateFromFileRequest.

- “Busy (SICCT)”: Update engine is currently downloading the image using SICCT DOWNLOAD.
- “Busy (Writing data)”: Update engine has verified the update bundle and is currently writing it onto the device.
- “Busy (Waiting for reboot)”: Update engine is awaiting reboot.
- “Busy (Grace period)”: Update has completed reboot and is wait for the grace period to timeout.

The list below enumerates the possible error messages of the GetUpdateStateResponse message:

- “Invalid session Id”
- “Internal error”

4.34 GetPairingInformation

Encoding: JSON

The GetPairingInformation message can be used to get the current set pairing information.

The payload of the GetPairingInformationRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the GetPairingInformationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "pairingBlocks": {
      "type": "object",
      "properties": {
        "pairingBlock1": {
          "type": "object",
          "properties": {
            "isUsed": {
              "type": "boolean"
            },
            "key1": {
              "type": "object",
              "properties": {
                "certificateFingerprint": {
                  "type": "string"
                },
                "key": {
                  "type": "string"
                }
              }
            },
            "key2": {
              "type": "object",
              "properties": {
                "certificateFingerprint": {
                  "type": "string"
```

```
            },
            "key": {
              "type": "string"
            }
          }
        },
        "key3": {
          "type": "object",
          "properties": {
            "certificateFingerprint": {
              "type": "string"
            },
            "key": {
              "type": "string"
            }
          }
        }
      }
    },
    "pairingBlock2": {
      "type": "object",
      "properties": {
        "isUsed": {
          "type": "boolean"
        },
        "key1": {
          "type": "object",
          "properties": {
            "certificateFingerprint": {
              "type": "string"
            },
            "key": {
              "type": "string"
            }
          }
        },
        "key2": {
          "type": "object",
          "properties": {
            "certificateFingerprint": {
              "type": "string"
            },
            "key": {
              "type": "string"
            }
          }
        },
        "key3": {
          "type": "object",
          "properties": {
            "certificateFingerprint": {
              "type": "string"
            },
            "key": {
              "type": "string"
            }
          }
        }
      }
    },
    "pairingBlock3": {
      "type": "object",
      "properties": {
```

```
            "isUsed": {
              "type": "boolean"
            },
            "key1": {
              "type": "object",
              "properties": {
                "certificateFingerprint": {
                  "type": "string"
                },
                "key": {
                  "type": "string"
                }
              }
            },
            "key2": {
              "type": "object",
              "properties": {
                "certificateFingerprint": {
                  "type": "string"
                },
                "key": {
                  "type": "string"
                }
              }
            },
            "key3": {
              "type": "object",
              "properties": {
                "certificateFingerprint": {
                  "type": "string"
                },
                "key": {
                  "type": "string"
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Note that only the keys and pairing blocks which exist will be present in the response object.

The list below enumerates the possible error messages of the GetPairingInformationResponse message:

- “Invalid session Id”
- “Internal error”

4.35 DeletePairingBlock

Encoding: JSON

The DeletePairingBlock message can be used to delete the whole pairing block (including all keys within).

The payload of the DeletePairingBlockRequest message contains the following fields:

```
{
  "type": "object",
```

```
  "properties": {
    "blockId": {
      "type": "integer",
      "minimum": 1,
      "maximum": 3
    }
  },
  "required": ["blockId"]
}
```

The payload of the DeletePairingBlockResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the DeletePairingBlockResponse message:

- “Invalid session Id”
- “Invalid input value (blockId)” (German: Pairingblock)
- “Internal error”

4.36 DeletePairingKey

Encoding: JSON

The DeletePairingKey message can be used to delete a key from a pairing block.

The payload of the DeletePairingKeyRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "blockId": {
      "type": "integer",
      "minimum": 1,
      "maximum": 3
    },
    "keyId": {
      "type": "integer",
      "minimum": 1,
      "maximum": 3
    }
  },
  "required": ["blockId", "keyId"]
}
```

The payload of the DeletePairingKeyResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
```

```
  }
}
```

The list below enumerates the possible error messages of the DeletePairingKeyResponse message

- “Invalid session Id”
- “Invalid input value (blockId)” (German: Pairingblock)
- “Invalid input value (keyId)” (German: Pairingschlüssel)
- “Internal error”

4.37 SetLoginCredentials

Encoding: JSON

The SetLoginCredentials message can be used to change the login credentials for the remote management service or the SICCT user.

The payload of the SetLoginCredentialsRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "username": {
      "type": "string",
      "pattern": "^admin$"
    },
    "adminPassword": {
      "type": "string",
      "maxLength": 256
    },
    "passwordType": {
      "type": "string",
      "pattern": "^(remotemgmt|sicct)$"
    },
    "passwordUser": {
      "type": "string",
      "maxLength": 256
    },
    "newPassword": {
      "type": "string",
      "maxLength": 256
    }
  },
  "required": [ "username", "adminPassword", "passwordType", "passwordUser",
↪"newPassword" ]
}
```

The payload of the SetLoginCredentialsResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The pair username/adminPassword is the current remote administration password.

Since username/adminPassword is validated before accepting any changes the error counter is increased/reset on every failed password/successful verification as defined in Section 4.6.

The rules for locking passwords (see Section 4.6) apply. Hence, “Login disabled” is returned if the remote management login is currently disabled.

In case a password for a locked account is set, the call will succeed. However, the lockout time will remain.

The following passwordType/passwordUser pairs are supported:

- “remotemgmt”/”admin”: Update remote management password
- “sicct”/”admin”: Update the sicct admin password

The list below enumerates the possible error messages of the SetLoginCredentialsResponse message:

- “Invalid session Id”
- “Invalid input value (username)” (German: Benutzername)
- “Invalid input value (adminPassword)” (German: Admin Passwort)
- “Invalid input value (passwordType)” (German: Passworttyp)
- “Invalid input value (passwordUser)” (German: Passwort Username)
- “Invalid input value (newPassword)” (German: Neues Passwort)
- “Invalid credentials”
- “Login disabled” (German: Login derzeit nicht möglich”)

4.38 SmcbAuthentication

Encoding: JSON

The SmcbAuthentication message requests an ephemeral authentication key from the *Remote Management API*. The device answers with a freshly generated AES256 key used to compute the CMAC required to authenticate to the *Remote SMCB API* (see Section 7.7.3). If there was already an active key, the old key is removed (API versions since 3.0).

It is **not** allowed to regenerate a key if,

- there is an active connection to the *Remote SMCB API*, or
- the service is not enabled on at least one slot.

The key **must** only be used to compute a single CMAC.

The payload of the SmcbAuthenticationRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the SmcbAuthenticationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "key": {
      "type": "string",
      "pattern": "^[0-9a-f]{64}$"
    }
```

```
  }
}
```

The list below enumerates the possible error messages of the SmcbAuthenticationResponse message

- “Invalid session Id”
- “Remote SMCB already connected”
- “Remote SMCB disabled”
- “Internal error”

4.39 SetRemotePin

Encoding: JSON

The SetRemotePin message can be used to enable or disable remote pin management for the four card slots independently. Setting a remotepin to:

- “0” disables remote pin entry
- “1” enables remote pin entry via PP-1516 (only API versions since 3.0)
- “2” enables remote pin entry via webinterface

for the corresponding card slot.

The payload of the SetRemotePinRequest contains the following fields:

```
{
  "type": "object",
  "properties": {
    "remotepin1": {
      "type": "string",
      "pattern": "^(0|1|2)$"
    },
    "remotepin2": {
      "type": "string",
      "pattern": "^(0|1|2)$"
    },
    "remotepin3": {
      "type": "string",
      "pattern": "^(0|1|2)$"
    },
    "remotepin4": {
      "type": "string",
      "pattern": "^(0|1|2)$"
    }
  },
  "required": [ "remotepin1", "remotepin2", "remotepin3", "remotepin4" ]
}
```

The payload of the SetRemotePinResponse contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the SetRemotePinResponse message:

- “Invalid session Id”
- “Invalid input value (remotepin1)” (German: Kartenslot 1)
- “Invalid input value (remotepin2)” (German: Kartenslot 2)
- “Invalid input value (remotepin3)” (German: Kartenslot 3)
- “Invalid input value (remotepin4)” (German: Kartenslot 4)
- “Internal error”

4.40 GetServiceAnnouncement

Encoding: JSON

The GetServiceAnnouncement message can be used to obtain information about the service announcement configuration. The unit of the service announcement interval is minutes.

The payload of the GetServiceAnnouncementRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the GetServiceAnnouncementResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "active": {
      "type": "string",
      "pattern": "^(on|off)$"
    },
    "interval": {
      "type": "integer",
      "minimum": 1,
      "maximum": 30
    },
    "unicast": {
      "type": "string",
      "pattern": "^(on|off)$"
    },
    "unicastPort": {
      "type": "integer",
      "minimum": 0,
      "maximum": 65535
    },
    "unicastIp": {
      "type": "string",
      "pattern": "(^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9])$)|(^$)"
    }
  }
}
```

The list below enumerates the possible error messages of the GetServiceAnnouncementResponse message:

- “Invalid session Id”
- “Internal error”

4.41 SetServiceAnnouncement

Encoding: JSON

The SetServiceAnnouncement message can be used to set the service announcement configuration. The unit of the service announcement interval is minutes.

The payload of the SetServiceAnnouncementRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "active": {
      "type": "string",
      "pattern": "^(on|off)$"
    },
    "interval": {
      "type": "integer",
      "minimum": 1,
      "maximum": 30
    },
    "unicast": {
      "type": "string",
      "pattern": "^(on|off)$"
    },
    "unicastPort": {
      "type": "integer",
      "minimum": 0,
      "maximum": 65535
    },
    "unicastIp": {
      "type": "string",
      "pattern": "(^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-
↪4]|1{0,1}[0-9]){0,1}[0-9])$)|(^$)"
    }
  },
  "required": [
    "active",
    "interval",
    "unicast",
    "unicastPort",
    "unicastIp"
  ]
}
```

The payload of the SetServiceAnnouncementResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the SetServiceAnnouncementResponse message:

- “Invalid session Id”
- “Invalid input value (active)” (German: Aktiv)
- “Invalid input value (interval)” (German: Intervall)
- “Invalid input value (unicast)”
- “Invalid input value (unicastPort)”
- “Invalid input value (unicastIp)”
- “Internal error”

4.42 GetPinEntry

Encoding: JSON

The GetPinEntry message can be used to obtain information about the configuration of the pin entry scrambling.

The payload of the GetPinEntryRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the GetPinEntryResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "mode": {
      "type": "string",
      "pattern": "^(scrambled|unscrambled)$"
    }
  }
}
```

The list below enumerates the possible error messages of the GetPinEntryResponse message:

- “Invalid session Id”
- “Internal error”

4.43 SetPinEntry

Encoding: JSON

The SetPinEntry message can be used to set the scrambling of the pin entry.

The payload of the SetPinEntryRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "mode": {
```

```
      "type": "string",
      "pattern": "^(scrambled|unscrambled)$"
    }
  },
  "required": [
    "mode"
  ]
}
```

The payload of the SetPinEntryResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the SetPinEntryResponse message:

- “Invalid session Id”
- “Invalid input value (mode)” (German: Modus)
- “Internal error”

4.44 GetPinpadInformation

Encoding: JSON

Get information on the Pinpad configuration and peer (API versions since 3.0).

The payload of the GetPinpadInformationRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the GetPinpadInformationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    },
    "enabled": {
      "type": "string",
      "pattern": "^(enabled|disabled)$"
    },
    "pairing": {
      "type": "object",
      "properties": {
        "fingerprint": {
          "type": "string",
          "pattern": "^[0-9a-fA-F]{96}$"
        },
        "lastIp": {
          "type": "string",
```

```
          "pattern": "(^((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-
↪5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])$)|(^$)"
        },
        "lastInterface": {
          "type": "string",
          "pattern": "^(local|extern)$"
        },
        "lastName": {
          "type": "string"
        }
      },
      "required": [
        "fingerprint"
      ]
    },
    "peer": {
      "type": "object",
      "properties": {
        "fwVersion": {
          "type": "string"
        },
        "hwVersion": {
          "type": "string"
        },
        "macAddress": {
          "type": "string",
          "pattern": "^(([0-9a-fA-F]{2}):){5}([0-9a-fA-F]{2})"
        }
      },
      "required": [
        "fwVersion",
        "hwVersion",
        "macAddress"
      ]
    }
  },
  "required": [
    "enabled"
  ]
}
```

- `enabled`: The Pinpad support is either `enabled` or `disabled`.
- `pairing`: Information on the paired Pinpad. The field is present if (and only if) the Pinpad is enabled and a Pinpad is paired. It is absent otherwise.
  - `fingerpint`: SHA384 fingerprint of the certificate used for the SICCT connection
  - `lastIp`: IP number of the Pinpad it was recently connected from.
  - `lastInterface`: If the value is `local` the Pinpad connection was made by a direct connection on the physical USB-A Jack of the terminal. If the value is `extern` then the connection was made by a connection on one of the external interfaces (either the USB-C or the Ethernet jack).
  - `lastName`: The name of the Pinpad that the Pinpad sent during the most recent connection.
- `peer`: Information on the connected Pinpad. The field is present if (and only if) the Pinpad is enabled and a Pinpad is connected. It is absent otherwise.
  - `fwVersion`: Firmware version of the connected Pinpad.
  - `hwVersion`: Hardware version of the connected Pinpad.
  - `macAddress`: MAC address of the connected Pinpad.

The list below enumerates the possible error messages of the GetPinpadInformation message:

- “Invalid session Id”
- “Internal error”

4.45 SetPinpadConfiguration

Encoding: JSON

Set information on the Pinpad configuration and peer (API versions since 3.0).

The payload of the SetPinpadConfigurationRequest message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "enabled": {
      "type": "string",
      "pattern": "^(enabled|disabled)$"
    }
  },
  "required": [
    "enabled"
  ]
}
```

- `enabled`: The Pinpad support shall be either `enabled` or `disabled`.

The payload of the SetPinpadConfigurationResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the SetPinpadConfiguration message:

- “Invalid session Id”
- “Invalid input value (enabled)” (German: Aktiviert)
- “Internal error”

4.46 DeletePinpadPairing

Encoding: JSON

The DeletePinpadPairing message can be used to delete the pairing secret shared with the paired Pinpad (API versions since 3.0).

The payload of the DeletePinpadPairingRequest message is empty:

```
{
  "type": "object",
  "properties": {
  }
}
```

The payload of the DeletePinpadPairingResponse message contains the following fields:

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "string"
    }
  }
}
```

The list below enumerates the possible error messages of the DeletePinpadPairingResponse message:

- “Invalid session Id”
- “Internal error”

5 Message processing (Remote Management API)

All messages received via Remote management API are validated. The validation consists of the following stages (performed in this order):

1. JSON validation
2. Format validation
3. Header validation
4. Session identification
5. Payload contents validation
6. Process message according to the request description

5.1 JSON validation

Received messages are parsed using JSON grammar. If the message is invalid or incomplete JSON, it will be dropped and the connection will be closed.

5.2 Format validation

Once JSON validation is passed, the validity of the message is checked. It is checked that:

- the message is an object,
- the object contains the three fields: “header”, “payloadType” and “payload”,
- the three envelope fields are of the correct type (object, string and object respectively),
- the contents of the header fields have the correct type,
- the payload type is known,
- all required fields in the payload are present,
- all present fields in the payload are of the correct type

If this validation stage fails, there will be no reply from the remote management service and the connection will be dropped.

5.3 Header validation

Once format validation is passed, it is checked that the message header contains correctly formatted content. It is checked that:

- the header contains a valid message id (the pattern is correct),
- if header contains a session id it must be valid (the pattern is correct),

If this validation stage fails, there will be no reply from the remote management service and the connection will be dropped.

5.4 Session identification

If a session ID is present, it is checked that the session is valid (validation, that the session is known by the system and not timed out).

If the session ID is not present or invalid and the request requires a valid session ID (all requests but LoginRequest and GetApiVersion), the error "Invalid session Id" is responded (see Section 4.4).

5.5 Payload contents validation

The contents of the message are validated. This is specific for each message. For example, the existence of a valid session with the received ID, whether an URL is reachable or not and so on. If an error occurs at this stage, it will be reported in the error field of the respective Response message.

5.6 Patterns used in validation

The following patterns/rules are used for message validation:

- `[0-9a-f]{32}` : session ID, message ID (also reply to ID)
- `^(on|off)$` : option that could be set to on / off
- `^admin$` : username for Login
- `^[ -~]{1,32}$` : device name
- `^[ -~ÄäÖöÜüß§]{1,32}$` : idle screen message
- `^(PU|RU|TU)$` : environment (SetEnvironment)
- `^(((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]))$|(^$)` : all IP addresses
- `^https?://.+` : url (InitiateFwUpdate)
- `^(([-a-zA-Z0-9._:]{0,255}){1}(,[-a-zA-Z0-9._:]{0,255})*)$|(^$)` : NTP server list
- `^(ethernet|usbRndis|usbCdcEcm)$` : ethernet device selection
- `^(remotemgmt|sicct)$` : username for SetLoginCredentials
- `^(scrambled|unscrambled)$` : pin entry mode
- `^(0|2)$` : remote pin entry source
- `^(enabled|disabled)$` : option that could be enabled / disabled (PinpadConfiguration)
- `^(off|tls|mschap)$` : VPN type configuration
- `^((https?://[^\\\\,' ]+){1}(,(https?://[^\\\\,' ]+){1}){0,})$|(^$)` : VPN CRL URI's
- `(^[^\\t\\r\\n\\v\\f]{0,511}[^\\t\\r\\n\\v\\f ]$)|(^$)` : VPN username / password
- `^(fingerprint|subject)$` : VPN client ID type (TLS)
- `^(off|serialnumber|text)$` : VPN client ID type (MS-CHAP)
- `^[ -~]{1,256}$` : VPN client ID text
- `(^-----BEGIN [A-Z ]{0,16}CERTIFICATE-----(\n|\r|\r\n)([0-9a-zA-Z+/=]{64}(\n|\r|\r\n))*([0-9a-zA-Z+/=]{1,63}(\n|\r|\r\n))?-----END [A-Z ]{0,16}CERTIFICATE-----( |\n|\r|\r\n){0,4}$)|($^)` : VPN certificates

- `(^-----BEGIN [A-Z ]{0,16}PRIVATE KEY-----(\n|\r|\r\n)([0-9a-zA-Z+/=]{64}(\n|\r|\r\n))*([0-9a-zA-Z+/=]{1,63}(\n|\r|\r\n))?-----END [A-Z ]{0,16}PRIVATE KEY-----( |\n|\r|\r\n){0,4}$)|($^)` : VPN private keys

6 Remote SMCB API

This chapter describes the messages supported by the remote management service. In general messages exchange follows the request-response pattern, where the remote management service sends the requests and the client replies with a response. This service is only available for API versions since 3.0.

The connection is created as defined in RFC 6455, with the protocol name *cobra-smcb*:

```
new WebSocket(uri, "cobra-smcb");
```

7 Message encoding (JSON/Remote SMCB API)

All data received on the WS protocol `cobra-smcb` are decoded as JSON messages.

7.1 Fragmentation

Before parsing the message is defragmented according to RFC 6455 Sec. 5.4. The maximum total message size is 16 kilobytes.

In case the size is exceeded the received data is dropped and the connection is closed.

7.2 JSON encoded messages

In general messages are encoded using the JavaScript Object Notation (JSON) according to RFC 4627.

Each message must further follow the following JSON schema (specified using IETF draft for JSON Schema v4[1]):

```
{
  "type": "object",
  "properties": {
    "Header": {
      "type": "object",
      "properties": {
        "MsgId": {
          "type": "string",
          "pattern": "^[0-9a-f]{32}$"
        },
        "InReplyToId": {
          "type": "string",
          "pattern": "^[0-9a-f]{32}$"
        },
        "SessionId": {
          "type": "string",
          "pattern": "^[0-9a-f]{32}$"
        }
      },
      "required": [ "MsgId" ]
    },
    "PayloadType": {
      "type": "string",
      "enum": [
        "Notify",
        "Cancel",
        "AuthenticateRequest",
        "AuthenticateResponse",
        "OutputRequest",
        "OutputResponse",
        "InputPinRequest",
        "InputPinResponse"
      ]
    },
```

[1] http://json-schema.org/

```
    "Payload": {
      "type": "object"
    }
  },
  "required": [ "Header", "PayloadType", "Payload" ]
}
```

The field `MsgId` should contain a unique message ID, which will be used for correlating request messages and their responses. Response messages must have the `InReplyToId` set to the value of the `MsgId` field of their request message. Request messages should not have an `InReplyToId` field, if they have, then their value will be ignored.

All messages except `AuthenticateRequest`, `AuthenticateResponse` must have the field `SessionId` present and valid. The `Cancel` message may omit the field `SessionId` if an `AuthenticateRequest` is canceled.

The `PayloadType` field defines the type of the `Payload` object and is defined below. In general messages are exchanged using a strict request/response/notify pattern, which is also exposed in the payload type postfix. Every request message has a payload type with a postfix of `Request` and every response has a payload type with a postfix of `Response`. The `Notify` identifies the matching `Response` with the `InReplyToId` field.

The payload field contains the actual data corresponding to a message and embedded into a JSON object. Each payload type has its own payload structure. The payload types and their payload object structure are defined below.

7.3 Messaging

Fig. 7.1 Depicts the general messaging schemata. The ST-1506 sends a request to the client, the client responds and finally the ST-1506 acknowledges the response by sending a notify with the outcome. Note the *OutputRequest* has the option to not expect a response from the client. In this only the request is sent, but the *Response* and *Notify* are omitted.

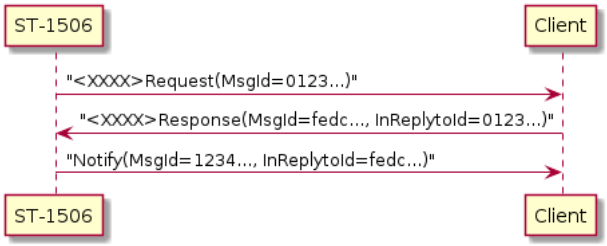

Fig. 7.1: Messaging Schema

Fig. 7.2 depicts the message schemata when ST-1506 cancels a request. The is no response expected from the client.

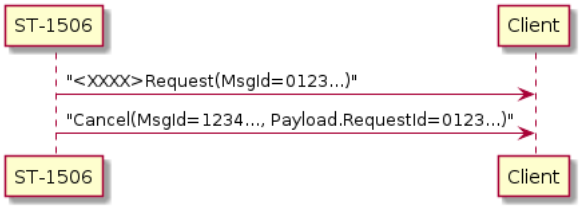

Fig. 7.2: Messaging Schema

7.4 Authentication

The *Remote SMCB API* uses a dedicated connection which is independent to the *Remote Management API*. At any time at most one connection to the *Remote SMCB API* is allowed. This connection is formally divided into 3 states.

- *Disconnected*: While disconnected an authentication token can be retrieved (see Section 4.38). If a client connects one of the following two actions occurs:
  1. There is an *Authentication Key* available: ST-1506 sends an `AuthenticateRequest` (see Section 7.7.3) and enters the state `Connected`.
  2. There is **no** *Authentication Key* available: ST-1506 disconnects WS connection.
- *Connected*: While connected the *Remote SMCB API* waits for the reception of an `AuthenticateResponse` (see Section 7.7.4). If
  1. a valid response is received the *Remote SMCB API* transitions into the state *Authenticated*. It generates a new `SessionId` and sends a `Notify` messages containing the `SessionId`.
  2. an invalid response (or after a timeout) is received the authentication key is invalidated. No state transitions are taken. In order to retry, the client must disconnect and restart the authentication procedure.
- *Authenticated*: While authenticated the Remote SMCB API will send `OutputRequests` and `InputPinRequests` if a PIN verification is requested on the respective slot from the Konnektor.

  While authenticated the Remote SMCB API sends the SessionId in each message.

Fig. 7.3 outlines the messages required to successfully authenticate to the *Remote SMCB API*.

1. Messages 1-3: The client requires an authenticated connection to the *Remote Management Interface* (see Section 3, Section 4.6).

   Note, that it is also possible to reuse an standing session as long as the `sessionId` is known.

2. Messages 4,5: Request a new authentication key by executing the `SmcbAuthentication` request (see Section 4.38).

   On success you receive a `Key` that stays valid for 30 seconds. Within this period the client must connect to the Websocket `cobra-smcb` (Message 8 in Fig. 7.3).

3. Messages 6,7: Optionally the connection to the *Remote Management Interface* may be closed.

4. Messages 8-11: Connect to the Websocket `cobra-smcb` and complete the Challenge response.

   After connecting to the Websocket `cobra-smcb` (Message 8) ST-1506 sends an `AuthenticateRequest` (Message 9) containing a `Challenge`.

   The client is expected to respond with a Challenge Response (Message 10) computed as follows:

   ```
   CMAC(Key, Challenge)
   ```

   The *CMAC* is computed over the block cipher *AES256*, using the `Key` received in Message 5 and the `Challenge` received in Message 9.

   ST-1506 verifies the response and sends a `Notify` with the proper `Code`. On success the `Notify` contains the `SessionId` used on subsequent messages.

Note: The timeout for authentication is refreshed when the ST-1506 receives Message 8. The maximal overall timeout for establishing a successful connection is therefore 30+30 seconds.

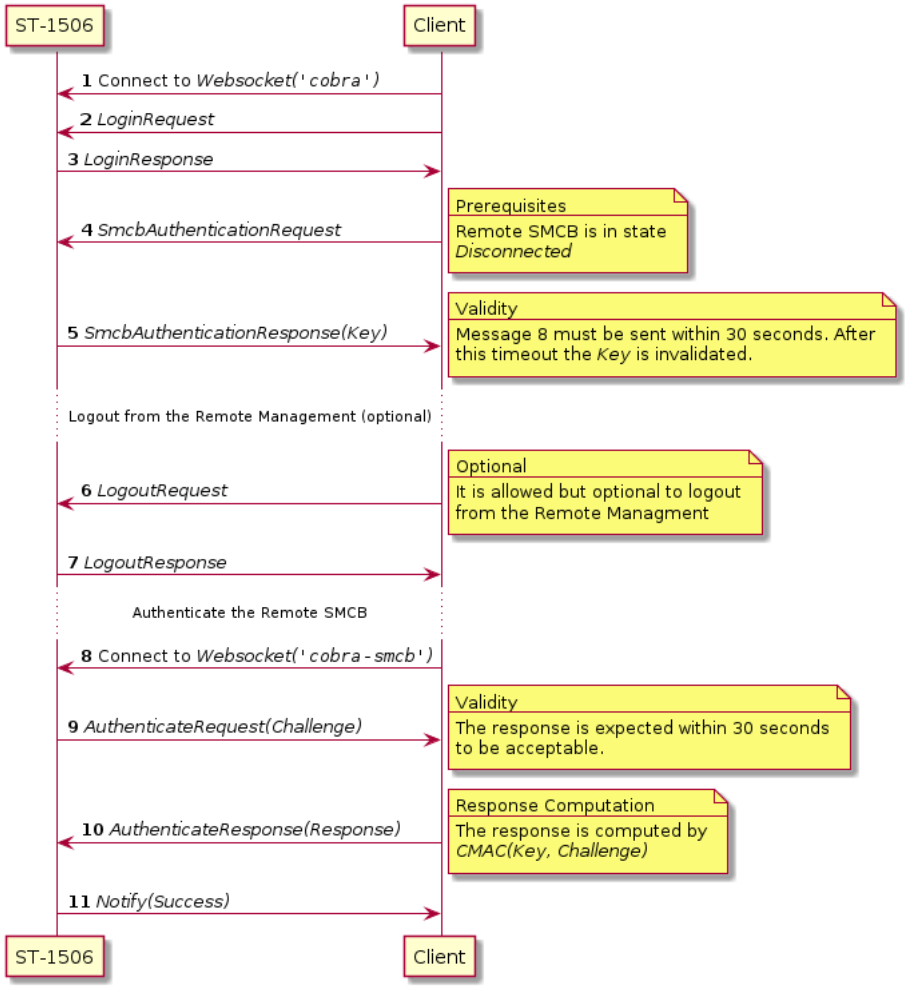

Fig. 7.3: Messaging Schema

7.5 Error handling

In case of errors the error code in the *Notify* message is sent to the client.

The payload contains a field code which denotes the error:

Code	Name	Interpretation
0	Success	All good
1	ParserError	Failed to decode JSON data
2	AuthenticationRequired	Message requires authentication
3	AuthenticationDenied	Authentication not successful
4	InvalidSessionId	Invalid `SessionId` field
5	InvalidMessageId	Invalid or missing `MessageId` field
6	InvalidInReplyToId	Invalid `InReplyToId` field
7	MissingInReplyToId	Missing `InReplyToId` field
8	InvalidChallengeResponse	Invalid `InvalidChallengeResponse` field
9	MissingPin	Missing `Pin` field
10	InvalidPin	Invalid `Pin` field
11	InvalidPinLength	Invalid `Pin` length
12	UnexpectedPin	`Pin` field present, but not expected
13	InvalidOutputCode	The code from a output response was invalid

During processing of messages error conditions can happen. The behavior is described in Section 8

7.6 Denial of WS connections

WS connections are denied if:

- The remote SMCB is disabled on all Slots
- If the service is **not** in the state *Disconnected*
- If the service is *Disconnected* but no authentication key is available.

These rules ensure that at most one connection is available.

7.7 Message Definition

7.7.1 Notify

Direction: ST1506 to Client

Response: *None*

ST-1506 sends a `Notify` to the client to acknowledge the reception and processing of client responses. The field `Code` depicts the outcome of the message (see Section 7.5) for possible error codes.

The payload of `Notify` contains the following fields:

```
{
  "type": "object",
  "properties": {
    "Code": {
      "type": "int",
      "min": 0,
      "max": 13
    }
  },
  "required": [ "Code" ]
}
```

7.7.2 Cancel

Direction: ST1506 to Client

Response: *None*

ST-1506 sends a `Cancel` to the client to cancel an ongoing request. When the client receives this message it shall abort processing the referenced request without sending the respective response. The request is referenced by the `RequestId` field in the payload.

The payload of `Cancel` contains the following fields:

```
{
  "type": "object",
  "properties": {
    "RequestId": {
      "type": "string",
      "pattern": "^[0-9a-f]{32}$"
    }
  },
  "required": [ "RequestId" ]
}
```

7.7.3 AuthenticateRequest

Direction: ST1506 to Client

Response: `AuthenticateResponse` (Section 7.7.4)

Before the ST-1506 is willing to forward PIN verification requests the client must authenticate itself. For this after connecting the ST-1506 sends a challenge containing a `Challenge` to be completed to the client.

Since this is the first message exchanged the request also contains some information on the protocol.

The timeout until a response is accepted is 30 seconds.

The following fields are included in the `AuthenticateRequest`:

- `ApiVersion`: Contains the string "1.0" referring to this specification.
- `Challenge`: String containing 64 hexadecimal characters containing the challenge.

```
{
  "type": "object",
  "properties": {
    "ApiVersion": {
      "type": "string",
      "pattern": "^1.0$"
    },
    "Challenge": {
      "type": "string",
      "pattern": "^[0-9a-f]{64}$"
    }
  },
  "required": [ "ApiVersion", "Challenge" ]
}
```

7.7.4 AuthenticateResponse

Direction: Client to ST1506

Response: `Notify` (Section 7.7.1)

To complete the authentication the client must respond to a `AuthenticateRequest` with a proper `Response` within 30 seconds.

The `SessionId` in the header may be absent, it is ignored if present.

The response is computed as follows (pseudo code):

```
key = convert_hex_string_to_binary(Key)
challenge = convert_hex_string_to_binary(Challenge)
alg = AES256-CBC
response = CMAC(alg, key, challenge)
```

The following fields are included in the AutenticateResponse:

- `Response`: Contains response encoded as a string with 32 hexadecimal characters.

```
{
  "type": "object",
  "properties": {
    "Response": {
      "type": "string",
      "pattern": "^[0-9a-f]{32}$"
    }
  },
  "required": [ "Response" ]
}
```

In response to a `AuthenticateResponse` ST-1506 sends a `Notify` with the outcome of the response.

- message parsing applies (see Section 8)
- *Success*: The authentication was successful
- *MissingInReplyToId*: `InReplyToId` was missing
- *InvalidInReplyToId*: `InReplyToId` was invalid or does not match the `AuthenticateRequest`
- *InvalidChallengeResponse*: The content of the payload field *Response* is not proper
- *AuthenticationDenied*: The response was incorrect

7.7.5 OutputRequest

Direction:	ST1506 to Client
Response:	`OutputResponse` (Section 7.7.6) if `ExpectResponse = true`;
	None if `ExpectResponse = false`

On behalf of the ST-1506 the client shall be able to display message and ask the user to confirm or cancel. If the ST-1506 requests such an action it sends an `OutputRequest` message to the connected client.

The following fields are included in the OutputRequest:

- `Message`: Contains the message to be presented to the user
- `MessageType`: Contains a context to the user:
  - `PinResult` indicates that the message contains the outcome of a PIN operation
  - `PinConfirm` indicates that the message contains a query to the user to confirm a PIN operation.
- `MessageCode` indicates how the message shall be display
  - `Info`: it is an informal message (e.g.: Pin operation was successful)
  - `Warning`: it is a warning message (e.g.: Pin operation failed)

- `Timeout` indicates the timeout after which request is considered to timeout in seconds.
- `OkButton` indicates that a Ok/Confirm button shall be displayed and that `OkButton` is a valid response within the `OutputResponse`.
- `CancelButton` indicates that a Ok/Confirm button shall be displayed and that `CancelButton` is a valid response within the `OutputResponse`.
- `ExpectResponse` indicates if the client shall send a response to the ST-1506. If this value is `false` the ST-1506 considers the request as complete after sending.
- `Slot` depicts the slot the pin operation targets. In protocol version "1.0" the field is always present. But it is left as optional since future extension might allow omitting the field.
- `Atr` depicts the ATR of the ICC in the targeted slot. In protocol version "1.0" the field is always present. But it is left as optional since future extension might allow omitting the field.

```
{
  "type": "object",
  "properties": {
    "Slot": {
      "type": "string",
      "pattern": "^Slot[1234]$"
    },
    "Atr": {
      "type": "string",
      "pattern": "^([0-9a-f][0-9a-f])+$"
    },
    "Message": {
      "type": "string"
    },
    "MessageType": {
      "type": "string",
      "pattern": "^(PinConfirm|PinResult)$"
    },
    "MessageCode": {
      "type": "string",
      "pattern": "^(Info|Warning)$"
    },
    "Timeout": {
      "type": "int"
    },
    "OkButton": {
      "type": "bool"
    },
    "CancelButton": {
      "type": "bool"
    },
    "ExpectResponse": {
      "type": "bool"
    }
  },
  "required": [
    "Message",
    "MessageType",
    "MessageCode",
    "Timeout",
    "OkButton",
    "CancelButton",
    "ExpectResponse"
  ]
}
```

7.7.6 OutputResponse

Direction: Client to ST1506

Response: `Notify`

The client responds with a `Code` depicting the response of the user.

The following fields are included in the OutputResponse:

- `Code`: Contains one of `Error`, `Timeout`, `OkButton`, `CancelButton`.

```
{
  "type": "object",
  "properties": {
    "Code": {
      "type": "string",
      "pattern": "^(Timeout|OkButton|CancelButton|Error)$"
    }
  },
  "required": ["Code"]
}
```

In response to a `OutputResponse` ST-1506 sends a `Notify` with the outcome of the response.

- message parsing applies (see Section 8)
- *Success*: The authentication was successful
- *MissingInReplyToId*: `InReplyToId` was missing
- *InvalidInReplyToId*: `InReplyToId` was invalid or does not match the `OutputRequest`
- *InvalidOutputCode*: The code was `OkButton` or `CancelButton`, while not requested in the corresponding request

7.7.7 InputPinRequest

Direction: ST1506 to Client

Response: `InputPinResponse` (Section 7.7.8)

On behalf of the ST-1506 the client shall be able to request a PIN entry from the user. If the ST-1506 requests such an action it sends an `InputPinRequest` message to the connected client.

The following fields are included in the InputPinRequest:

- `Prompt`: Contains a string with the prompt to display
- `Message`: Contains a message to display
- `MessageType`: Contains the context of the operation
  - `Validate` indicates a pin verification operation
- `MinLen`: Indicate the minimal expected PIN length
- `MaxLen`: Indicate the maximal expected PIN length
- `TimeoutFirst`: Indicate the timeout in seconds until the user is expected to enter the first digit
- `TimeoutOther`: Indicate the timeout in seconds until the user is expected to enter the subsequent digits
- `TimeoutAll`: Indicate the overall timeout of the request in seconds
- `OkButton`: Indicate if the client shall present a ok/confirm button (if this field is set to false the pin shall be sent when user entered `MaxLen` digits)

- `CancelButton`: Indicate if the client shall present a cancel button
- `Slot` depicts the slot the pin operation targets
- `Atr` depicts the ATR of the ICC in the targeted slot

Note: `MessageType` allows only the option `Validate` since PIN modification operations are **never** requested via the *Remote PIN* API.

Note: The ST-1506 sends a `Cancel` message if the user removes the card while asked for a PIN.

```
{
  "type": "object",
    "properties": {
    "Slot": {
      "type": "string",
      "pattern": "^Slot[1234]$"
    },
    "Atr": {
      "type": "string",
      "pattern": "^([0-9a-f][0-9a-f])+$"
    },
    "Prompt": {
      "type": "string"
    },
    "Message": {
      "type": "string"
    },
    "MessageType": {
      "type": "string",
      "pattern": "^Validate$"
    },
    "MinLen": {
      "type": "int",
      "minimum": 0
    },
    "MaxLen": {
      "type": "int",
      "minimum": 0
    },
    "TimeoutFirst": {
      "type": "int",
      "minimum": 0
    },
    "TimeoutOther": {
      "type": "int",
      "minimum": 0
    },
    "TimeoutAll": {
      "type": "int",
      "minimum": 0
    },
    "OkButton": {
      "type": "bool"
    },
    "CancelButton": {
      "type": "bool"
    }
  },
```

```
  "required": [
    "Slot",
    "Atr",
    "Prompt",
    "Message",
    "MessageType",
    "MinLen",
    "MaxLen",
    "TimeoutFirst",
    "TimeoutOther",
    "TimeoutAll",
    "OkButton",
    "CancelButton"
  ]
}
```

7.7.8 InputPinResponse

Direction: Client to ST1506

Response: `Notify`

The client asks the user to enter a PIN to be sent to the terminal. In case of a timeout a partial entered PIN shall be sent along with the code `Timeout` if (and only if) the partially entered PIN length is between `MinLen` and `MaxLen`.

The field `Pin` shall only contain ASCII digits.

The following fields are included in the InputPinResponse:

- `Code`: Contains one of `Error`, `Timeout`, `Pin`, `CancelButton`.
- `Pin`: Contains the PIN. This field
  - **must** be present if the code is `Pin`.
  - **may** be present if the code is `Timeout`.
  - **must not** be preset if the code is `Error` or `CancelButton`.
  - **if preset** the field must contain a string containing only digits.
  - **if preset** the field must contain a string with at list `MinLen` digits.
  - **if preset** the field must contain a string with at most `MaxLen` digits.

```
{
  "type": "object",
  "properties": {
    "Code": {
      "type": "string",
      "pattern": "^(Timeout|Pin|CancelButton|Error)$"
    },
    "Pin": {
      "type": "string",
      "pattern": "^[0-9]{0,64}$"
    }
  },
  "required": ["Code"]
}
```

In response to a `InputPinResponse` ST-1506 sends a `Notify` with the outcome of the response.

- message parsing applies (see Section 8)
- *Success*: The authentication was successful
- *MissingInReplyToId*: `InReplyToId` was missing

- *InvalidInReplyToId*: `InReplyToId` was invalid or does not match the `OutputRequest`
- *MissingPin*: `Code` was `Pin` but `Pin` field is missing
- *InvalidPin*: The `Pin` field contains invalid characters
- *InvalidPinLength*: The `Pin` field is too short or too long
- *UnexpectedPin*: A `Pin` is preset while the code was neither `Pin` nor `Timeout`

Note: There is no code for incorrectly responding with code `CancelButton` since in protocol version 1.0 the field `CancelButton` is always true.

---

8 Message processing (Remote SMCB API)

All messages received via the Remote SMCB API are validated. The validation consists of the following stages (performed in this order):

1. JSON validation
2. Format validation
3. Header validation
4. Session identification
5. Payload contents validation
6. Process message according to the request description

8.1 JSON validation

Received messages are parsed using JSON grammar. If the message is invalid or incomplete JSON, it will be dropped. A notification (Code: *ParserError*) is sent.

8.2 Format validation

Once JSON validation is passed, the validity of the message is checked. It is checked that:

- the message is an object,
- the object contains the three fields: “Header”, “PayloadType” and “Payload”,
- the three envelope fields are of the correct type (object, string and object respectively),
- the contents of the header fields have the correct type,
- the payload type is known,
- all required fields in the payload are present,
- all present fields in the payload are of the correct type

If this validation stage fails the message is dropped and a notification (Code: *ParserError*) is sent.

8.3 Header validation

Once format validation is passed, it is checked that the message header contains correctly formatted content. It is checked that:

- the header contains a valid “MessageId” (the pattern is correct),
- if header contains a `SessionId` it must be valid (the pattern is correct),
- the header contains a valid “InReplyToId” (the pattern is correct)

If this validation stage fails the message is dropped and a notification with the respective code (*InvalidMessageId*, *InvalidSessionId*, *InvalidInReplyToId* *MissingInReplyToId*) is sent.

8.4 Session identification

If the response requires authentication the `SessionId` field must be present and match the Authentication `SessionId`.

If this validation stage fails the message is dropped and a notification (Code: *AuthenticationRequired*) is sent.

8.5 Payload contents validation

The contents of the message are validated. This is specific for each message. If the validation fails a notification message with the respective code is sent.

8.6 Patterns used in validation

The following patterns/rules are used for message validation:

- `[0-9a-f]{32}` : session ID, message ID (also reply to ID), challenge response.
- `[0-9]{1,64}` : PIN