Remootio Websocket API Version 1.0

1) Introduction

The Remootio Websocket API lets you integrate your Remootio device with existing home automation systems. Using it you can simply write your own feature exensions for Remootio. The API is open, and we also provide open-source client-side libraries for it. We also encourage our users to share their code with the community. There is a range of client-side libraries and examples made available by Remootio as well:

- A website to test the Remootio API. (Javascript in the browser)
- Node.js Remootio API client
- Python (coming soon)

The API is a Websocket API run on your local home LAN network. The Remootio device has to be conencted to your home Wi-Fi for this feature. Once the API is enabled the Remootio device will run a websocket server on it's port 8080 where the API client can connect to.

2) Enabling the API

To enable the API in the Remootio. Swipe down to the Device software tab of a device (you must have the master key for the device), and open the Remootio Webscoket API settings. The API can be enabled in two modes:

- Enable API with logging
- Enable API without logging

If the API is enabled without logging, the Remootio device will notify the API client about when the gate or garage door's status has changed (a status sensor needs to be installed for this). If the API is enabled with logging the Remootio device will also notify the API client about a wide range of other events such as when and which key operated the gate, when some settings were changed and so on...

After enabling the API the app will display the credentials needed for API access:

- The API Secret Key - a 64 character long hexstring (e.g. "12b3f03211c384736b8a1906635f4abc90074e680138a689caf03485a971efb3")
- The API Auth Key - another 64 character long hexstring (e.g. "74ca13b56b3c898670a67e8f36f8b8a61340738c82617ba1398ae7ca62f1670a")
- The device's IP address on the local network (e.g. 192.168.1.115)

For the best user experience it is recommended to assign an fixed IP address to your Remootio device using your home Router.

3) Connecting to the API

After the API is enabled, it can be connected at ws://:8080 (e.g. ws://192.168.1.115). As it's a ws:// conection and not a secure wss:// connection the API implements encryption of frames containing sensitive information. These frames are called ENCRYPTED frames (see Basic Fraems). Moreover the API client needs to authenticate itself to the Remootio device otherwise the connection will be closed. (More about the authentication process is described in chapter 6)

4) Frame constructions

The API expects frames (frame = payload of the websocket messages in this context) that are stringified JSON objects. The replies to these frames are also stringified JSON objects. There are two main frame constructions in the API. A very basic frame looks like the following:

```
{"type":"<frame_type_string>"}
```

And a frame that carries encrypted information looks like:

```
{
    "type":"ENCRYPTED",
    "data":{
        "iv":"<base64_encoded_string>",
        "payload":"<base64_encoded_string>",
    },
    "mac":"<base64_encoded_string>"
}
```

ENCRYPTED frames basically wrap messages denoted as UNENCRYPTED_PAYLOAD into ENCRYPTED frames by applying cryptography to the unencrypted message (UNENCRYPTED_PAYLOAD).

In the following chapters both of these main constructions will be used. The first construction will be referenced as BASIC frame and the second construction as ENCRYPTED frame.

5) Basic frames

ERROR frame

Use: The Remootio device sends error frames to the API client to indicate various errors Direction: API client → Remootio device

```
{
    "type":"ERROR",
    "errorMessage":"<error_message_string>"
}
```

The "<error_message_string>" can be any of the following:

<error_message_string>	Meaning

<error_message_string>	Meaning
"json error"	The Remootio device couldn't parse the frame as a JSON object.
"input error"	The frame received by Remootio is a valid JSON object, but the values in it do not correspond to any valid Remootio API frame.
"internal error"	The Remootio device encountered an internal error.
"connection timeout"	If Remootio receives no incoming frame from the API client for 120 seconds it sends the following error frame and then closes the websocket connection
"authentication timeout"	Once a websocket connection is established with the Remootio Websocket API, the API client has to go through an authentication flow (More details about the Authentication can be found in in chapter 6). If this doesn't happen Remootio closes any unauthenticated session after 30 seconds, and sends this error frame.
"already authenticated"	If Remootio returns this error frame if it receives an AUTH frame when the session is already authenticated.
"authentication error"	If there is an error with the authentication flow Remootio returns this error frame. Such errors are typically caused by invalid Api Secret Key and Api Auth Key settings, non-Remootio-API-compliant encryption algorithm implementations (if not using the example client-side libraries), or actionId errors (actionId is a frame counter for actions the Remootio device receives from the API client)

PING frame

Use: The API client sends this to keep the connection alive. It is recommended to send one PING frame to the Remootio device every 60-90 seconds and also check for the PONG response to detect a broken connection. Direction: API client → Remootio device

```
{"type":"PING"}
```

Response: Remootio device → API client

```
{"type":"PONG"}
```

HELLO frame

Use: The API client can send this frame to check the version of the Websocket API running on the Remootio device. Direction: API client → Remootio device

```
{"type":"HELLO"}
```

Response: Remootio device → API client

```
{
    "type":"SERVER_HELLO",
    "apiVersion":1,
    "message":"This is the Remootio Websocket API"
}
```

---

6) AUTH frame and the authentication flow

A new session is created every time the API client connects to the Websocket API of the Remootio device. Every session is unauthenticated by default. The API client however needs to authenticate each session to be able to send and receive ENCRYPTED frames containing sensitive information. (Such as a sending a command to open the gate or receiving log information about which key has opened it and when). In the described authentication flow below data for a real device are used, so the data can be used to verify implmenetation of the fram encryption and decryiption algorithms:

The credentials for the example data below are: API Secret Key: EFD0E4BF75D49BDD4F5CD5492D55C92FE96040E9CD74BED9F19ACA2658EA0FA9 API Auth Key: 7B456E7AE95E55F714E2270983C33360514DAD96C93AE1990AFE35FD5BF00A72

Step 1 - the API client sends an AUTH frame to Remootio

Use: The API client sends the AUTH frame to start the authentication flow. Direction: API client → Remootio device

```
{"type":"AUTH"}
```

Step 2 - Remootio sends an ENCRYPTED frame to the API client

Direction: Remootio device → API client This message looks like the one below. The payload of this ENCRYPTED frame is encrypted using the API Secret Key, and the MAC (Message Authentication Code) is calculated using Api Auth Key.

```
{
    "type":"ENCRYPTED",
    "data":{
        "iv":"4kbmkg6iU29Zlpi3NCDM4g==",

"payload":"ZTQwhEWXMV2ZxkzDJiJWyCD52FF88pha8lJbpD2KYk5B6TGQvBaTJlA7apd+lO3
8mu44NA7heNVZOc6B6jVwqvdqMSrEdV33KgaHMZY7yNXBq4aP3+Z2ai4TJ8Smgnj6Z77J4qeT6
MqBbr0FTLYkEg=="
    },
    "mac":"qko4r2/Eucwh8FqJIXucKn/w/ftR9+vs05E8A1/y++Q="
}
```

Detailed explanation on how to construct an ENCRYPTED frame from an UNENCRYPTED_PAYLOAD and how to extract the UNENCRYPTED_PAYLOAD of an ENCRYPTED frame are shown in chapter 7.

The client should verify that the message was not tampered with by verifying the MAC. To do this, the API client should calculate a HMAC-SHA256 using API Auth Key as the key over the content of the "data" field, then compare this calculated MAC with the one in the ENCRYPTED frame's "mac" field. The value in the "mac" field the is base64 encoded form of the HMAC-SHA256 MAC calculated by Remootio.

If the MAC matches, the client can be sure that the message has not beed tampered with. Now the encrypted payload can be decrypted. The payload can be decrypted using AES-CBC using API Secret Key as the encryption key the value of the data.iv field as the IV (initialization vector), and of course the value of the data.payload field as the ciphertext. Both of data.iv and data.payload are base64 encoded, so decode them first if the library used for the AES operation does not support base64 encoded inputs. After decrypting the data.payload the padding needs to be removed (PKCS7 padding is used). Removing the padding and interpreting the resulting UNENCRYPTED_PAYLOAD as ASCII (Latin1) string it will look like:

```
{
    "challenge":{
        "sessionKey":"yzEI7RWCjYDEwFrgc5YrmWo82kXEjFNStbtN+wFM2Qk=",
        "initialActionId":808411243
    }
}
```

This is the authentication challenge from Remootio. The sessionKey is another encryption key (similar to API Auth Key). In an authenticated session sessionKey (denoted as API Session Key) is used for encrypting and decrypting the payload of the ENCRYPTED frames. The API Auth Key is only used for encrypting the authentication challenge seen above. The initialActionId is a counter value for actions (commands) the API client can send to Remootio. A command/action for example is to open the gate or garage door. Each command the API client sends to Remootio must contain an acionId that is the last action id (denoted as lastActionId) incremented by one (and truncated to 31bits) (this is to add additional defense against replay attacks). In the authentication challenge the initialActionId is used as the lastActionId for the new session. The client can finish authentication by sending any valid action to the Remootio device. It is recommended to send a QUERY action (more about actions later on) as it has no direct impact on the device. The QUERY action prompts Remootio to send a response containing the current status of the gate or garage door (open/closed).

Step 3 - the API client sends any valid action to Remootio (preferably a QUERY action)

As actions will be discussed later in detail, here we briefly describe what has to be done. Every action sent from the API client to Remootio is encapsulated in an ENCRYPTED frame (as this is considered to involve sensitive information). To send the QUERY action that completes the authentication flow successfully the following UNENCRYPTED PAYLOAD has to be sent:

```
{
    "action":{
        "type":"QUERY",
        "id":808411243
    }
}
```

Note how the "id" field is id = initialActionId+1 = lastActionId+1. If the action id is not an increment of the previous action at any time the session will be closed by Remootio with an "authentication error" error frame. The actual formula to calculate the next actionId is:

```
id = (lastActionId + 1) % 0x7FFFFFFF //The modulo division is because the
actionId is truncated to 31 bits
```

The UNENCRYPTED_PAYLOAD shown above needs to be packaged into an ENCRYPTED frame using the session encryption key (API Session Key) received in the authentication challenge above. Detailed explanation on how to construct an ENCRYPTED frame from an UNENCRYPTED_PAYLOAD and how to extract the UNENCRYPTED_PAYLOAD of an ENCRYPTED frame are shown in chapter 7. The brief steps for this are the following: Pad the UNENCRYPTED_PAYLOAD using PKCS7, generae a cryptographically secure random IV, then encrypt it using AES-CBC using API Session Key as the encryption key. Base64 encode the iv and the ciphertext, and create the JSON string for the HMAC calculation:

```
{"iv":"vz3r424R6v9XFchkkgWQTw==","payload":"L6eTyvyY/q4I7oDAfdeDyz17x0vMUq
mqvnCYl73zG2UxnYpIKVIQ0DooAWxcm3WT"}
```

Calculate the HMAC-SHA256 over this string using the API Auth Key as the key. Then encode the calculated MAC using base64 encoding to construct the final ENCRYPTED frame:

```
{
    "type":"ENCRYPTED",
    "data":{
        "iv":"vz3r424R6v9XFchkkgWQTw==",

"payload":"L6eTyvyY/q4I7oDAfdeDyz17x0vMUqmqvnCYl73zG2UxnYpIKVIQ0DooAWxcm3W
T"
    },
    "mac":"legB+2ZnikMtX54VpkPVc8P7o17s61y1JqGDvFrxbts="
}
```

At this point the authentication flow is finished from the API client's perspective. The Remootio device will send a response in an ENCRYPTED frame to this query action. If that arrives successfully, the API client can be sure that the session is authenticated. For the completeness of this example here is the ENCRYPTED message that contains the response to the QUERY action:

```
{
    "type":"ENCRYPTED",
    "data":{
        "iv":"S7Mt0PR3MCADhHOPqhJPLA==",

"payload":"pSw+jH9iR3/nOO2+78EpQct3w+vJGKku+8ynSaYra6WsU4dHQJfMg1KNJkooVb1
/WYhT28NyGznEHEKt97SYTMG15KjWcQUuqRSlpGD3JzWi/5LG+JPvIg3ptivsFrRZR3wzHAtZI
6CekFujm8dhjeK/o6w+daK4FdvVh78pVigX6tBuNHEjoRQfUL9TRS9W"
    },
    "mac":"cD4IpRARmeWoUjkL4Kh40uhOMbs7P9prP497qZUapwQ="
}
```

And the UNENCRYPTED_PAYLOAD is:

```
{
    "response":{
        "type":"QUERY",
        "id":808411244,
        "success":true,
        "state":"no sensor",
        "t100ms":8985,
        "relayTriggered":false,
        "errorCode":""
    }
}
```

If Remootio sends an "authentication error" ERROR frame instead it means that the authentication failed.

```
{
    "type":"ERROR",
    "errorMessage":"authentication error"
}
```

6) Messages that require an authenticated session

All the messages that require an authenticated session are encapsulated in ENCRYPTED frames that are encrypted using API Session Key. These messages are all placed in the UNENCRYPTED_PAYLOAD of the ENCRYPTED frames. There are two main type of message exchange flows in the authenticated session:

- The API client sends actions to control Remootio, and receives responses to these actions.

- Remootio sends messages about important events to the API client (the scope of these messages depend on whether the API is enabled with or without logging). Remootio keeps track of the most recent 100 events that have happened so even if the API client is disconnected for some time, Remootio will send the events not already sent upon reconnection (and re-authentication).

In the next two chapters all the actions, responses and events will be described using the UNENCRYPTED_PAYLOAD for all of the messages only. The respective encrypted frames (as they depend on the encryption keys and do not pertain any significant information to the actions themselves) will not be shown.

7) Actions and the respective responses

QUERY action

Use: The API client sends this action to get the current state of the gate or garage door (open/closed). The UNENCRYPTED_PAYLOAD of the action is shown below (the action id also needs to be calculated id=lastActionId%0x7FFFFFFF) Direction: API client → Remootio device

```
{
    "action":{
        "type":"QUERY",
        "id":1017222789
    }
}
```

The UNENCRYPTED_PAYLOAD of the response ENCRYPTED frame is shown below: Response: Remootio device → API client

```
{
    "response":{
        "type":"QUERY",
        "id":1017222789,
        "success":true,
        "state":"closed",
        "t100ms":3354,
        "relayTriggered":false,
        "errorCode":""
    }
}
```

The field "type" hows which action the response is related to. The field "id" is the same as the id sent with the action above. The field "success" hows if the action was successful or not. It can be true/false. The field "state" shows the current state of the gate or garage door. It can be "open"/"closed"/"no sensor" The field "t100ms" shows the time passed since the last restart of the Remootio device in 100ms units. So t100ms=10 would mean that the device started up 1 seconds ago. The field "relayTriggered" shows if Remootio's output relay was triggered during the action or not (it's always false for the QUERY action) The field "errorCode" contains an error message if "success" is false

---

TRIGGER action

Use: The API client sends this action to trigger the control output of the Remootio device and thus operate the gate or garage door. The UNENCRYPTED_PAYLOAD of the action is shown below (the action id also needs to be calculated id=lastActionId%0x7FFFFFFF) Direction: API client → Remootio device

```
{
    "action":{
        "type":"TRIGGER",
        "id":1936062911
    }
}
```

The UNENCRYPTED_PAYLOAD of the response ENCRYPTED frame is shown below: Response: Remootio device → API client

```
{
    "response":{
        "type":"TRIGGER",
        "id":1936062911,
        "success":true,
        "state":"no sensor",
        "t100ms":11136,
        "relayTriggered":true,
        "errorCode":""
    }
}
```

If Remootio's control output is already being triggered and is busy, the response would contain "success":false and "errorCode":"ERR_RELAY_BUSY". The fields have the same meanings as previously.

OPEN action

Use: The API client sends this action to open the gate or the garage door. This will trigger Remootio's control output only if the gate or garage door status is "closed". The UNENCRYPTED_PAYLOAD of the action is shown below (the action id also needs to be calculated id=lastActionId%0x7FFFFFFF) Direction: API client → Remootio device

```
{
    "action":{
        "type":"OPEN",
        "id":1936062921
    }
}
```

The UNENCRYPTED_PAYLOAD of the response ENCRYPTED frame is shown below: Response: Remootio device → API client

```
{
    "response":{
        "type":"OPEN",
        "id":1936062921,
        "success":true,
        "state":"closed",
        "t100ms":16231,
        "relayTriggered":true,
        "errorCode":""
    }
}
```

In this example, the gate state was "closed", so the OPEN action did trigger Remootio's control output. If the state would be "open" the response would contain "success":true and "relayTriggered":false. If no status sensor is installed the response would contain "success":false and "errorCode":"ERR_NO_SENSOR". If Remootio's control output is already being triggered and is busy, the response would contain "success":false and "errorCode":"ERR_RELAY_BUSY".

The fields have the same meanings as previously.

CLOSE action

Use: The API client sends this action to close the gate or the garage door. This will trigger Remootio's control output only if the gate or garage door status is "open". The UNENCRYPTED_PAYLOAD of the action is shown below (the action id also needs to be calculated id=lastActionId%0x7FFFFFFF) Direction: API client → Remootio device

```
{
    "action":{
        "type":"CLOSE",
        "id":1936062931
    }
}
```

The UNENCRYPTED_PAYLOAD of the response ENCRYPTED frame is shown below: Response: Remootio device → API client

```
{
    "response":{
        "type":"CLOSE",
        "id":1936062931,
        "success":true,
        "state":"closed",
        "t100ms":19251,
        "relayTriggered":false,
        "errorCode":""
    }
}
```

In this example, the gate state was "closed", so the CLOSE action did not trigger Remootio's control output. If the state would be "open" the response would contain "success":true and "relayTriggered":true. If no status sensor is installed the response would contain "success":false and errorCode":"ERR_NO_SENSOR". If Remootio's control output is already being triggered and is busy, the response would contain "success":false and "errorCode":"ERR_RELAY_BUSY".

The fields have the same meanings as previously.

RESTART action

Use: The API client sends this action to restart the Remootio device. The UNENCRYPTED_PAYLOAD of the action is shown below (the action id also needs to be calculated id=lastActionId%0x7FFFFFFF) Direction: API client → Remootio device

```
    {
        "action":{
            "type":"RESTART",
            "id":66789
        }
    }
```

The UNENCRYPTED_PAYLOAD of the response ENCRYPTED frame is shown below: Response: Remootio device → API client

```
    {
        "response":{
            "type":"RESTART",
            "id":66789,
            "success":true,
            "state":"closed",
            "t100ms":54138,
            "relayTriggered":false,
            "errorCode":""
        }
    }
```

After the response message the Remootio device restarts, so the websocket connection to the API is closed. The client has to reconnect and re-authenticate the new session.

The field "relayTriggered" is always false for the RESTART action The fields have the same meanings as previously.

---

7) Events from the API

StateChange event

Use: Remootio sends the following event if the status of the gate or garage door has changed (from "open" to "closed" or from "closed" to "open"). This is the only event that is sent if the API is enabled without logging. It is also sent if the API is enabled with logging. Direction: Remootio device → API client

```
{
    "event":{
        "cnt":72,
        "type":"StateChange",
        "state":"open",
        "t100ms":18342
    }
}
```

The "type" field shows what kind of event it is The "state" field shows the state of the gate or garage door when the event happened. So for the StateChange event it contains the new state the gate or garage door just entered into. The field "t100ms" shows the time passed since the last restart of the Remootio device in 100ms units. So t100ms=10 would mean that the device started up 1 seconds ago. The "cnt" field is a counter field for events. Remootio increments this counter value by one for each event. The counter is reset to 0 if the device restarts.

RelayTrigger event

Use: Remootio sends the following event if it any key has operated the Remootio device (triggered the control output). Direction: Remootio device → API client

```
{
    "event":{
        "cnt":311,
        "type":"RelayTrigger",
        "state":"closed",
        "t100ms":12346,
        "data":{
            "keyNr":5,
            "keyType":"unique key",
            "via":"wifi"
        }
    }
}
```

The "keyNr" shows which key has operated the device (the number of each key is found in the Remootio app). The "keyType" field shows what kind of key it is. It can be:

- "master key" - the key of the Remootio device's owner who has set it up
- "unique key" - the unique keys the owner has shared with other people
- "guest key" - the guest key that can be shared with an unlimited amount of people
- "api key" - "keyType" is "api key" if the event was triggered by this websocket API client
- "smart home" - if the event was triggered by the smart home key used for Alexa, Google Home etc...

The "via" field shows what connection method was used to trigger this event. It can be:

- "bluetooth" - event was triggered via Bluetooth
- "wifi" - event was triggered via Wi-Fi
- "internet" - event was triggered via the Internet (EasyConnect)
- "autoopen" - event was triggered by the auto-open feature
- "unknown" - this should normally not happen at all

The other fields have the same meanings as previously.

Connected event

Use: Remootio sends the following event if any key has connected to the Remootio device. Direction: Remootio device → API client

```
{
    "event":{
        "cnt":412,
        "type":"Connected",
        "state":"closed",
        "t100ms":12426,
        "data":{
            "keyNr":0,
            "keyType":"master key",
            "via":"bluetooth"
        }
    }
}
```

The fields have the same meanings as previously.

LeftOpen event

Use: Remootio sends the following event if the gate or garage door has been left open for some time. Direction: Remootio device → API client

```
{
    "event":{
        "cnt":743,
        "type":"LeftOpen",
        "state":"open",
        "t100ms":22441,
        "data":{
            "timeOpen100ms":3000
        }
    }
}
```

The field "timeOpen100ms" shows how long the gate has been left open in multiples of 100 ms, so timeOpen100ms=3000 means that the gate or garage door has been left open for 300 seconds (5 minutes).

The other fields have the same meanings as previously.

KeyManagement event

Use: Remootio sends the following event if the access rights or notification settings for any key have been changed. Direction: Remootio device → API client

```
{
    "KeyManagement":{
        "cnt":324,
        "type":"KeyManagement",
        "state":"open",
        "t100ms":1434,
        "data":{
            "keyNr": 15,
            "keyType":"unique key",
            "bluetooth":true,
            "wifi":true,
            "internet":false,
            "notification":true,
            "isRemoved":false,
        }
    }
}
```

The field "bluetooth" shows if the Bluetooth access rights for the key are enabled (after the key management event). The field "wifi" shows if the Wi-Fi access rights for the key are enabled (after the key management event). The field "internet" shows if the Internet access rights for the key are enabled (after the key management event). The field "notification" shows if notifications are sent when the key operates the gate or garage door or not. The field "isRemoved" shows if the key has just been removed or not. If "isRemoved" is true then "bluetooth", "wifi", "internet", "notification" will all be false.

The other fields have the same meanings as previously.

Restart event

Use: Remootio sends the following event if it was restarted. This is only sent if the API is enabled with logging. Direction: Remootio device → API client

```
{
    "event":{
        "cnt":0,
        "type":"Restart",
        "state":"closed",
        "t100ms":16
    }
}
```

The fields have the same meanings as previously. The counter is reset to 0 if the device restarts, so the restart event has cnt=0

ManualButtonPushed event

Use: Remootio sends the following event if the manual button was pushed. This is only sent if the API is enabled with logging. Direction: Remootio device → API client

```
{
    "event":{
        "cnt":1342,
        "type":"ManualButtonPushed",
        "state":"closed",
        "t100ms":1632
    }
}
```

The fields have the same meanings as previously.

ManualButtonEnabled event

Use: Remootio sends the following event if the manual button was enabled. This is only sent if the API is enabled with logging. Direction: Remootio device → API client

```
{
    "event":{
        "cnt":324
        "type":"ManualButtonEnabled",
        "state":"closed",
        "t100ms":12321
    }
}
```

The fields have the same meanings as previously.

ManualButtonDisabled event

Use: Remootio sends the following event if the manual button was disabled. This is only sent if the API is enabled with logging. Direction: Remootio device → API client

```
    {
        "event":{
            "cnt":43263
            "type":"ManualButtonDisabled",
            "state":"closed",
            "t100ms":324113
        }
    }
```

The fields have the same meanings as previously.

DoorbellPushed event

Use: Remootio sends the following event if the doorbell was pushed. This is only sent if the API is enabled with logging. Direction: Remootio device → API client

```
    {
        "event":{
            "cnt":1343,
            "type":"DoorbellPushed",
            "state":"closed",
            "t100ms":1783
        }
    }
```

The fields have the same meanings as previously.

DoorbellEnabled event

Use: Remootio sends the following event if the doorbell was enabled. This is only sent if the API is enabled with logging. Direction: Remootio device → API client

```
{
    "event":{
        "cnt":6542,
        "type":"DoorbellEnabled",
        "state":"closed",
        "t100ms":1789
    }
}
```

The fields have the same meanings as previously.

DoorbellDisabled event

Use: Remootio sends the following event if the doorbell was disabled. This is only sent if the API is enabled with logging. Direction: Remootio device → API client

```
{
    "event":{
        "cnt":3432,
        "type":"DoorbellDisabled",
        "state":"closed",
        "t100ms":94727
    }
}
```

The fields have the same meanings as previously.

SensorEnabled event

Use: Remootio sends the following event if the status sensor was enabled. This is only sent if the API is enabled with logging. Direction: Remootio device → API client

```
{
    "event":{
        "cnt":1393,
        "type":"SensorEnabled",
        "state":"closed",
        "t100ms":17711
    }
}
```

The fields have the same meanings as previously.

SensorFlipped event

Use: Remootio sends the following event if the logic of the status sensor was flipped. This is only sent if the API is enabled with logging. Direction: Remootio device → API client

```
{
    "event":{
        "cnt":1411,
        "type":"SensorFlipped",
        "state":"closed",
        "t100ms":19712
    }
}
```

The fields have the same meanings as previously.

SensorDisabled event

Use: Remootio sends the following event if the status sensor was disabled. This is only sent if the API is enabled with logging. Direction: Remootio device → API client

```
{
    "event":{
        "cnt":1434,
        "type":"SensorDisabled",
        "state":"closed",
        "t100ms":19798
    }
}
```

The fields have the same meanings as previously.

7) Encrypting and decrypting ENCRYPTED frames

In this chapter psedo-code implementations (with javascript flavor) are shown for the ecryption and decryption of ENCRYPTED frames. Non-pseudo code implementations can be found in the client side libraries and examples.

The encryption keys for the examples below are the following: API Secret Key: EFD0E4BF75D49BDD4F5CD5492D55C92FE96040E9CD74BED9F19ACA2658EA0FA9 API Auth Key: 7B456E7AE95E55F714E2270983C33360514DAD96C93AE1990AFE35FD5BF00A72 API Session Key: yzEI7RWCjYDEwFrgc5YrmWo82kXEjFNStbtN+wFM2Qk=

Creating an ENCRYPED frame from UNENCRYPTED_PAYLOAD

This process creates an ENCRYPTED FRAME from an UNENCRYPTED_PAYLOAD The input is an UNENCRYPTED_PAYLOAD:

```
{
    "action":{
        "type":"QUERY",
        "id":808411243
    }
}
```

The frist step if to remove any formatting from the UNENCRYPTED_PAYLOAD JSON string (if any)

```
 var unencryptedPayload = '{"action":{"type":"QUERY","id":808411243}}'
```

The second step is to add PKCS7 padding to the UNENCRYPTED_PAYLOAD (some crypto libraries can do this with their encrypt operation)

```
 unencryptedPayload = StringToByteArray(unencryptedPayload,'Latin1')
//also convert unencryptedPayload to byte array if necessary using Latin1
(ASCII) encoding
 var paddedUnencryptedPayload = AddPKCS7Padding(unencryptedPayload)
```

Then a cryptographically strong random IV (of 16 bytes) has to be generated and the paddedUnencryptedPayload has to be encrypted with AES-CBC using API Session Key as the encryption key.

```
 var APISessionKey =
Base64.decode('yzEI7RWCjYDEwFrgc5YrmWo82kXEjFNStbtN+wFM2Qk=') //Convert
the API session key from base64 encoded string to byte array

 var iv = generateCryptographicallySecureRandomBytes(16); //Generate a 16
long byte array of cryptographically secure random numbers
 var ivBase64Encoded = Base64.encode(iv);
 console.log(ivBase64Encoded)
 //prints:
 //vz3r424R6v9XFchkkgWQTw==

 var encryptedPayload = AES_CBC_encrypt(
    paddedUnencryptedPayload, //plaintext
    iv, //IV
    APISessionKey, //encryption key
 )

 var payloadBase64Encoded = Base64.encode(encryptedPayload)
 console.log(payloadBase64Encoded)
 //prints
 //L6eTyvyY/q4I7oDAfdeDyz17x0vMUqmqvnCYl73zG2UxnYpIKVIQ0DooAWxcm3WT
```

After this the value of the ENCRYPTED frame's "data" field has to be constructed for MAC (Message Authentication Code) calculation

```
    //macBase - the data to calculate the MAC on
    //The order if "iv" and "payload" are very important ("iv" first) -
otherwise the calculated MAC will be different and Remootio won't accept
it
    var macBase = JSON.stringify({
        iv: ivBase64Encoded
        payload: payloadBase64Encoded
    })
    console.log(macBase)
    //prints:

//{"iv":"vz3r424R6v9XFchkkgWQTw==","payload":"L6eTyvyY/q4I7oDAfdeDyz17x0vM
UqmqvnCYl73zG2UxnYpIKVIQ0DooAWxcm3WT"}
```

The MAC to calculate is a HMAC-SHA256 using API Auth Key

```
    var APIAuthKey =
hexString.decode('7B456E7AE95E55F714E2270983C33360514DAD96C93AE1990AFE35FD
5BF00A72') //convert the hexstring to byte array if needed

    var mac = HMAC-SHA256(
        macBase, //The data to calculate HMAC on
        APIAuthKey, //The key for HMAC
    )

    var macBase64Encoded = Base64.encode(mac)
    console.log(macBase64Encoded)
    //prints
    //legB+2ZnikMtX54VpkPVc8P7o17s61y1JqGDvFrxbts=
```

The last step is to construct the ENCRYPTED frame

```
    var encryptedFrame = JSON.stringify({
        type:"ENCRYPTED",
        data:{
            iv: ivBase64Encoded,
            payload: payloadBase64Encoded
        },
        mac: macBase64Encoded
    })

    console.log(encryptedFrame)
    //prints
    //{"type":"ENCRYPTED","data":
{"iv":"vz3r424R6v9XFchkkgWQTw==","payload":"L6eTyvyY/q4I7oDAfdeDyz17x0vMUq
mqvnCYl73zG2UxnYpIKVIQ0DooAWxcm3WT"},mac":"legB+2ZnikMtX54VpkPVc8P7o17s61y
1JqGDvFrxbts="}
```

Decrypting the UNENCRYPTED_PAYLOAD from an ENCRYPTED frame

This process decrypts and parses the UNENCRYPTED_PAYLOAD of an ENCRYPTED frame

The input data is an encrypted frame:

```
var encryptedFrame = '{"type":"ENCRYPTED","data":
{"iv":"S7Mt0PR3MCADhHOPqhJPLA==","payload":"pSw+jH9iR3/nOO2+78EpQct3w+vJGK
ku+8ynSaYra6WsU4dHQJfMg1KNJkooVb1/WYhT28NyGznEHEKt97SYTMG15KjWcQUuqRSlpGD3
JzWi/5LG+JPvIg3ptivsFrRZR3wzHAtZI6CekFujm8dhjeK/o6w+daK4FdvVh78pVigX6tBuNH
EjoRQfUL9TRS9W"},"mac":"cD4IpRARmeWoUjkL4Kh40uhOMbs7P9prP497qZUapwQ="}'
```

First of all the MAC should be verified

```
var encryptedFrameObj = JSON.parse(encryptedFrame) //parse the ENCRYPTED
frame

//macBase is the data the MAC has to be calculated on
var macBase = JSON.stringify(encryptedFrameObj.data)
console.log(macBase)
//prints
//{"iv":"S7Mt0PR3MCADhHOPqhJPLA==","payload":"pSw+jH9iR3/nOO2+78EpQct3w+vJ
GKku+8ynSaYra6WsU4dHQJfMg1KNJkooVb1/WYhT28NyGznEHEKt97SYTMG15KjWcQUuqRSlpG
D3JzWi/5LG+JPvIg3ptivsFrRZR3wzHAtZI6CekFujm8dhjeK/o6w+daK4FdvVh78pVigX6tBu
NHEjoRQfUL9TRS9W"}

//load the API auth key
var APIAuthKey =
hexString.decode('7B456E7AE95E55F714E2270983C33360514DAD96C93AE1990AFE35FD
5BF00A72') //convert the hexstring to byte array if needed

var mac = HMAC-SHA256(
    macBase, //The data to calculate HMAC on
    APIAuthKey, //The key for HMAC
)
//mac should be a byte array (convert it to byte array if it's not)

var receivedMac = Base64.decode(encryptedFrameObj.mac) //convert the
received mac to byte array

//Compare the calculated MAC with the one in encryptedFrameObj
var isMACOk = true;
for (var i = 0; i < 32 ; i++){
    if (mac[i] != receivedMac[i]){
        isMACOk = false;
    }
}
```

The next step is to decrypt the payload and remove the padding afterwards. If the session is already authenticated API Session Key is used for decryption. If The API client just sent the AUTH frame, and tries to decrypt the ENCRYPTED message containing the authentication challenge API Secret Key has to be used instead.

```
    var iv = Base64.decode(encryptedFrameObj.data.iv) //Convert iv to byte array if needed
    var ciphertext = Base64.decode(encryptedFrameObj.data.payload) //Convert payload to byte array if needed

    if (Session_Authenticated == true){ //Check if the session is authenticated
        //If the session is authenticated, the encryption key is the API Session Key
        encryptionKey = Base64.decode('yzEI7RWCjYDEwFrgc5YrmWo82kXEjFNStbtN+wFM2Qk=') //Convert the API session key from base64 encoded string to byte array
    }else {
        //Else the encryption key is the API Secret Key
        var encryptionKey = hexString.decode('EFD0E4BF75D49BDD4F5CD5492D55C92FE96040E9CD74BED9F19ACA2658EA0FA9') //convert the hexstring to byte array if needed
    }

    //Decrypt the ciphertext
    var decryptedPaddedPayload = AES_CBC_decrypt(
        ciphertext, //ciphertext
        iv, //IV
        encryptionKey, //encryption key
    )

    //Remove PKCS7 padding
    var isPaddingOk = IsPKCS7PaddingOk(decryptedPaddedPayload); //Check if the padding is valid
    var decryptedPayload = RemovePKCS7Padding(decryptedPaddedPayload) //Remove the padding

    //Convert the decryptedPayload into a string using Latin1 (ASCII encoding)
    var unencryptedPayloadString = ByteArrayToString(decryptedPayload,'Latin1')
    console.log(unencryptedPayloadString)
    //prints:
    //{"response":{"type":"QUERY","id":808411244,"success":true,"state":"no sensor","t100ms":8985,"relayTriggered":false,"errorCode":""}}

    if (isPaddingOk == true && isMACOk==true){
        //The decryption was successful, and the decrypted data is
unencryptedPayloadString
    }else{
        //Error
    }
```