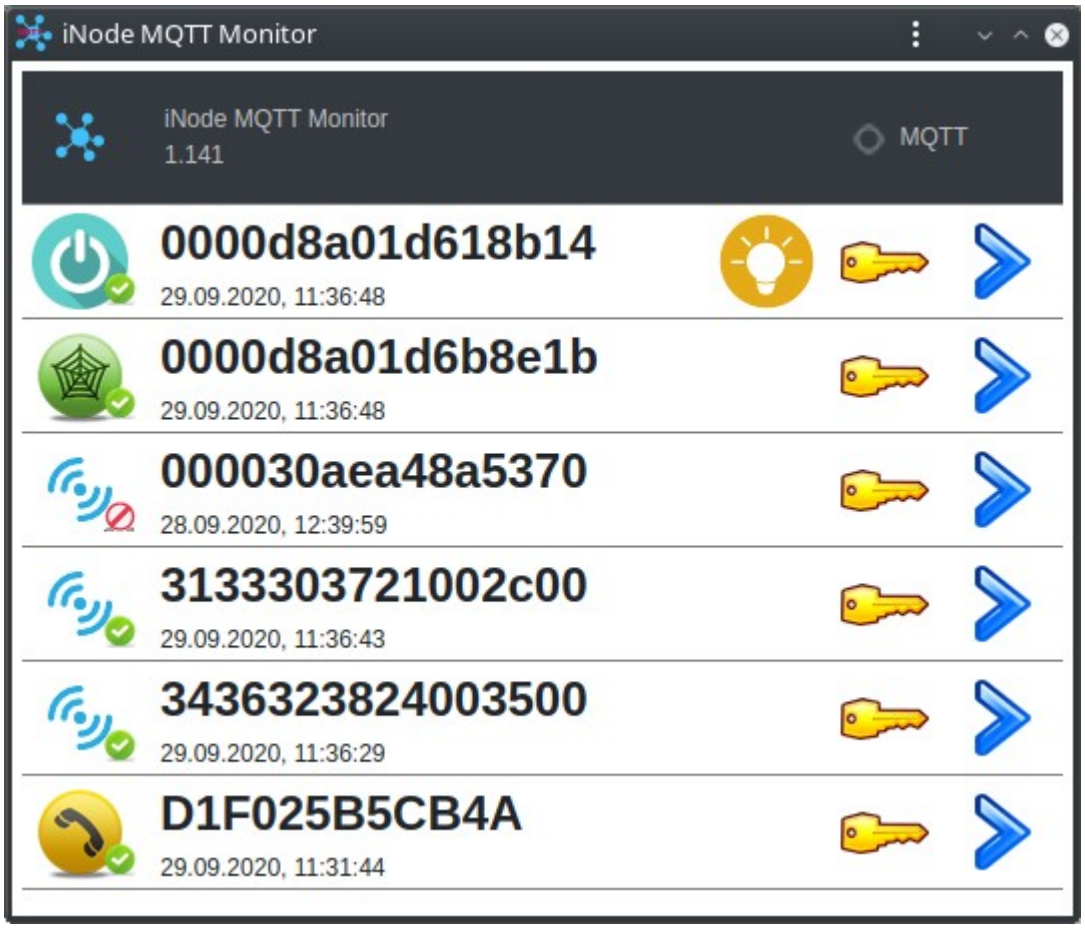
# *iNode MQTT Monitor*

***user manual***

## 1.1. MQTT MONITOR

The **iNode MQTT Monitor** application allows you to test communication between the device and the MQTT or HTTP / POST server. It also supports USB BT or LPWAN adapters and supports Web Bluetooth technology. Enables remote control of devices, e.g. the output in **iNode MCU Relay**. The **iNode MQTT Monitor** application is dedicated to the Google Chrome browser and works on Android, Windows, Linux operating systems, etc. After loading the application, it can be installed for easier launch later. An application icon will appear on the main screen.



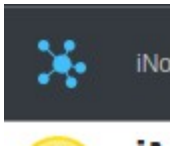

After starting the application, it shows devices that send data to the MQTT server iot.inode.pl. This is a free MQTT test server for users of the iNode products.

## Important !

ELSAT s.c. does not guarantee that the MQTT iot.inode.pl server will be available in the future and under what conditions. The user must be aware that data sent to this server may be received by others. For privacy, you should ensure that send on this server data to be encrypted - this is the default option in the **iNode LAN Central**, **iNode MCU Relay** or **iNode GSM MQTT LPWAN** device. The default password for encrypting them is different and randomly

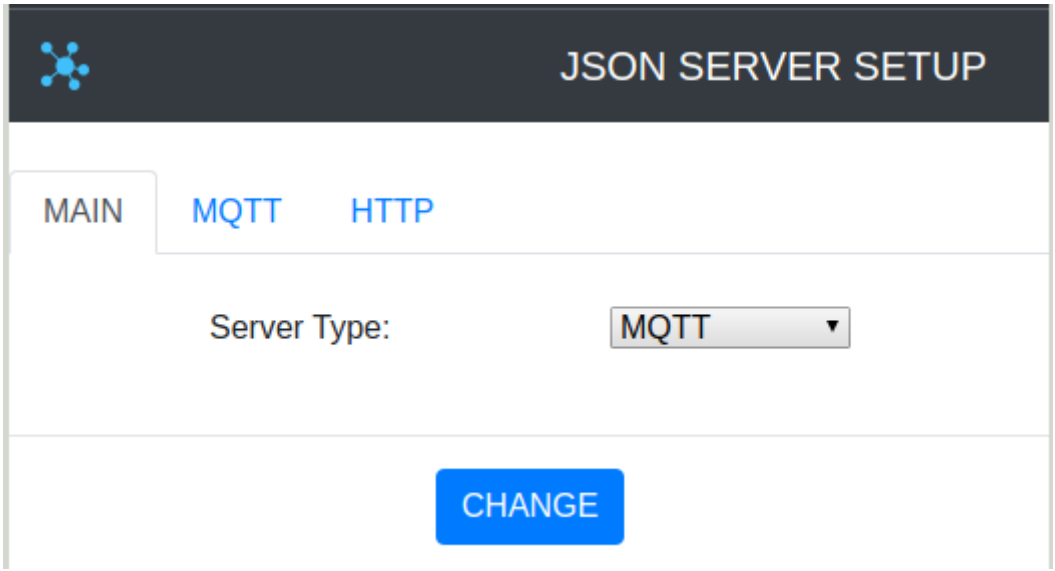
created on each device. The data on the server are not archived in any way, but they are publicly available, which results from the specifics of the MQTT server operation if access to it is not restricted by means of a username and password. ELSAT s.c. is not responsible for the content of this data in any way and does not interfere in any way – moderates it.

Configuration of the **iNode MQTT Monitor** application is done by clicking on the image in the upper left corner of the screen:

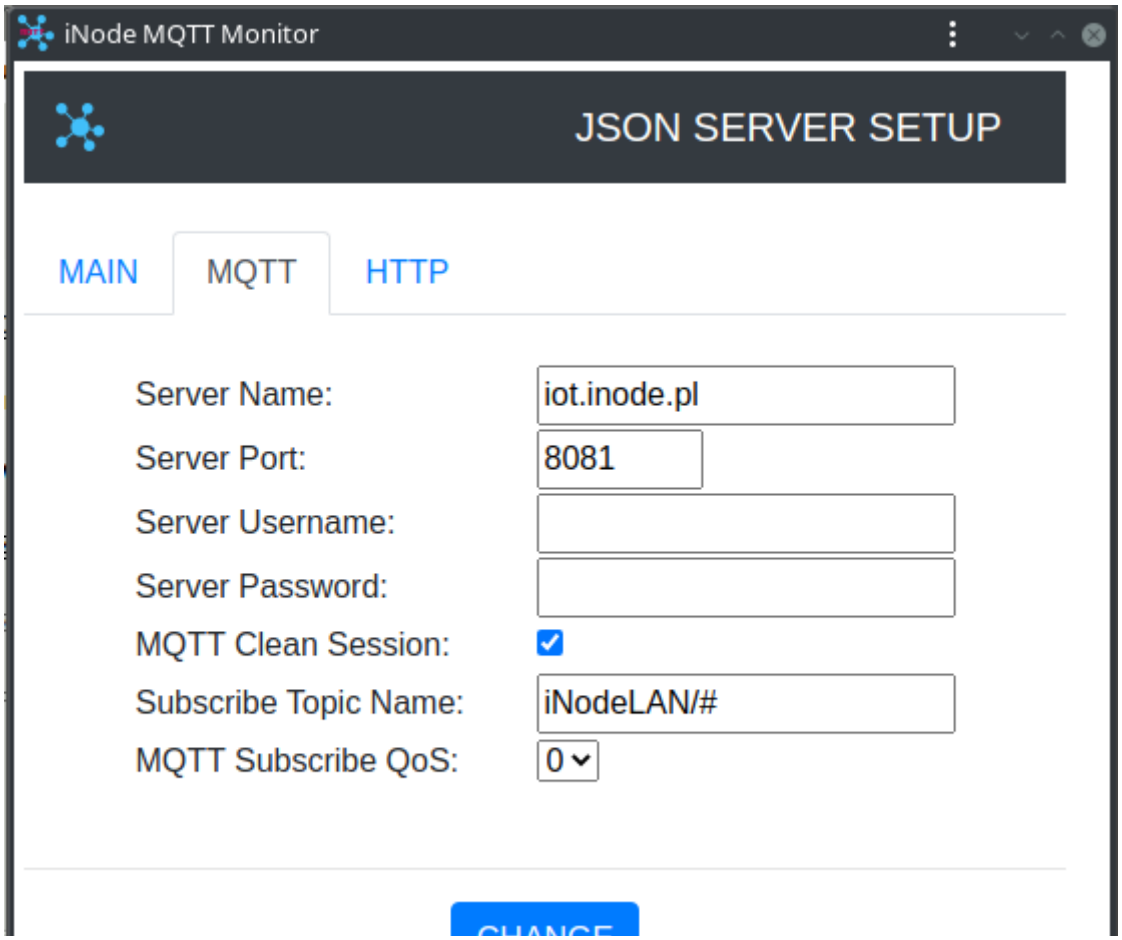


The following application screen will appear - **JSON SERVER SETUP**:

The ***MAIN*** tab allows you to select the type of server with which the application is to work. It can be HTTP, MQTT, USB or BLUETOOTH. The latter option is available from Google Chrome 79, however, so far it works only under Android. You may need to enable in **chrome://flags/#enable-experimental-web-platform-features** for USB or BLUETOOTH.

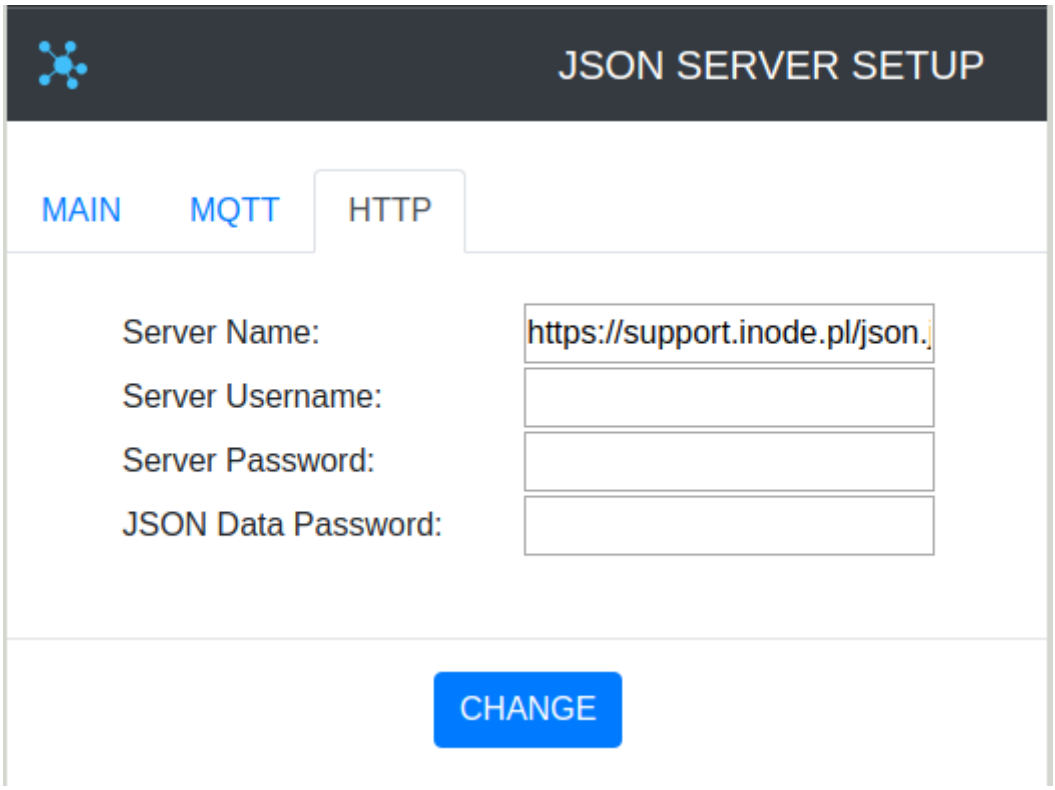


The ***MQTT*** tab allows you to enter the parameters of the MQTT server.



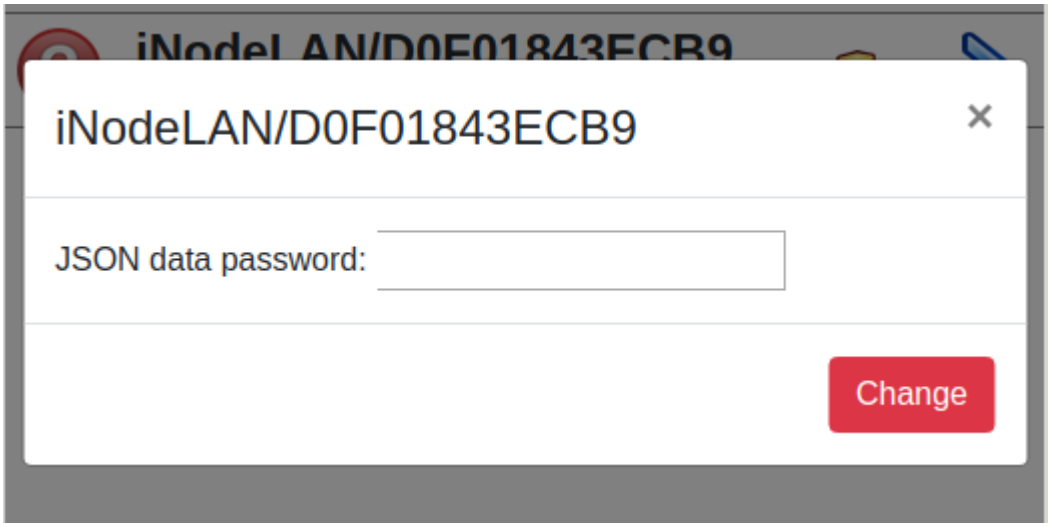

- ***Server Name*** – server name
- ***Server Port*** – the port at which the WebSocket service of the MQTT server is available
- ***Server Username*** – username if access to the MQTT server is restricted
- ***Server Password*** – password to access the MQTT server
- ***MQTT Clean Session*** - when the ***MQTT Clean Session*** flag is set, the client does not want a persistent session. If the client disconnects for any reason, all information and messages in the queue from the previous persistent session are lost.
- ***Subscribe Topic Name*** – it must be the same value as in the **iNode LoRa GSM MQTT** settings in the **PUBLISH – Topic** field or its fragment.
- ***MQTT Subscribe QoS:***
  - ***QoS 0*** - the client will not receive any confirmation from the server. Similarly, the message delivered to the client from the server does not have to be confirmed. This is the fastest way to post and receive messages, but also the one where you will most likely lose messages.
  - ***QoS 1*** - the client will receive a confirmation message from the server after it has been published. If the expected confirmation is not received within the specified time, the customer must retry the message. The message received by the client must also be confirmed in time, otherwise the server will deliver the message again.

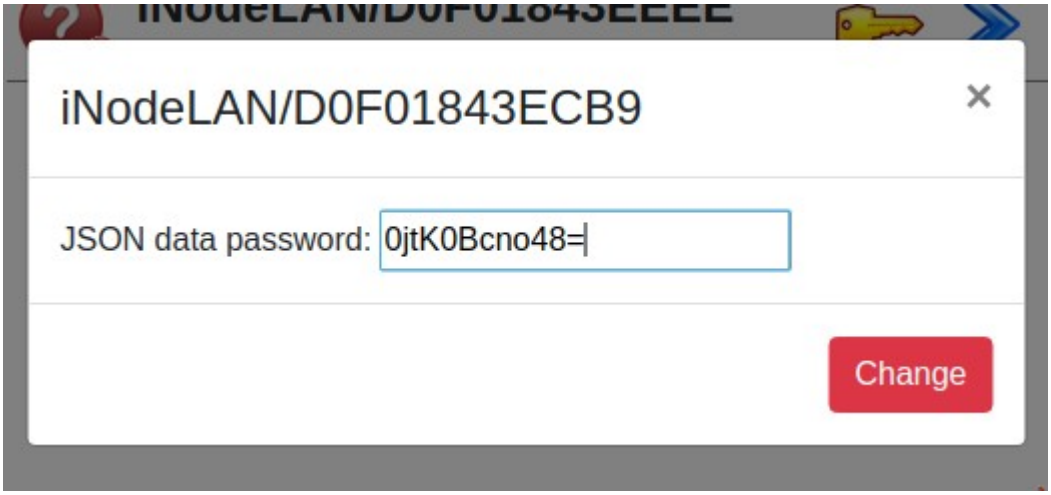
The ***HTTP*** tab allows you to specify HTTP server parameters.



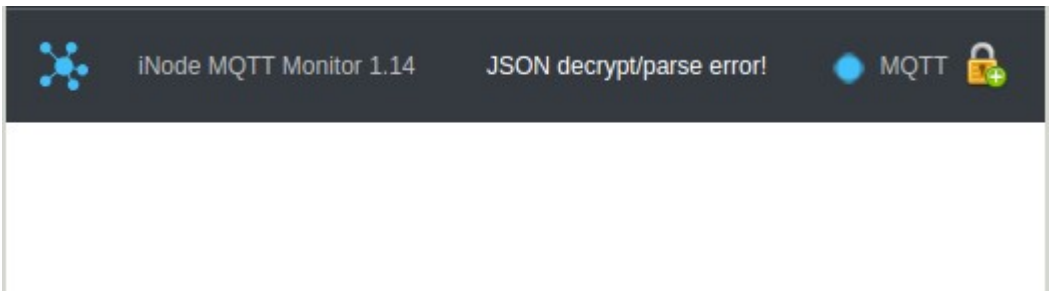

- ***Server Name*** – a full url containing the server name and path to the file with the given JSON
- ***Server Port*** – the port at which the HTTP server service is available
- ***Server Username*** – username if access to the HTTP server is restricted. Basic authorization type
- ***Server Password*** – password to access the HTTP server

If **iNode LoRa GSM MQTT** sends encrypted JSON data, then after selecting the picture with the key, enter the password to decode them and press the ***CHANGE*** button . It must be the same password as in the ***JSON*** tab in the ***KEY*** field.

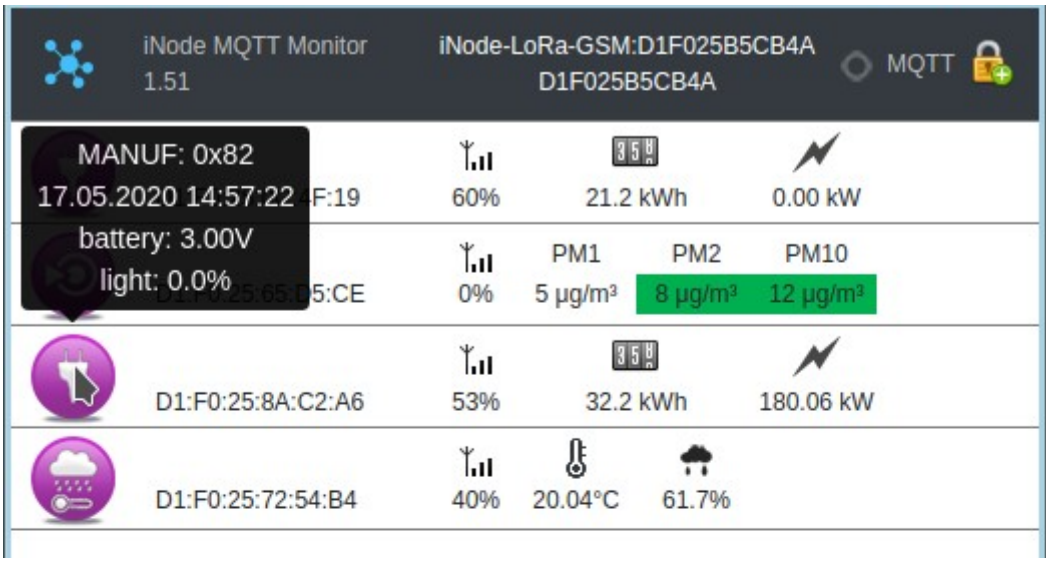





If the password is unset or wrong, after selecting the blue right arrow an error message will appear - ***JSON decrypt/parse error***. The fact that the data is encrypted is shown by a picture of the padlock in the upper right corner of the screen.




If the password was entered correctly, the application will display information about the devices from which they are sent by the given **iNode LoRa GSM MQTT**. By hovering the mouse or touching individual elements (smartphone) additional information about a given device will be displayed.



# 2. JSON data format

## 2.1. Decrypted JSON data:

In the first position of the JSON data being sent - the ***data*** table, there is information about **iNode LAN Central**.

- timestamp – timestamp
- type – name
- mac – mac address
- rtc – device time in seconds
- ethRx – number of frames received on the LAN
- ethTx – number of frames sent over the LAN
- bleRx- number of frames received by BLE
- bleTx - the number of frames sent by BLE (only if active scanning is enabled)
- workTime – device operation time in seconds
- txp – set transmission power
- rst – number of device resets
- temp – device temperature in degrees Celsius
- msg – number of JSON data shipments
- ack – number of confirmed data shipments
- tx_time – JSON data sending time in microseconds
- juf – security information
- period – period of sending JSON data
- manuf – device type code
- rstr – reason for the last reset

{"data": [{"timestamp": "2020-02-05T13:10:35Z","type": "iNode-LAN:43ECB9","mac": "D0F01843ECB9","ip": "10.10.6.47","rtc": 1580908235,"ethRx": 85,"ethTx": 1,"bleRx": 114,"bleTx": 0,"workTime": 17,"txp": 8,"rst": 7,"temp": 56,"msg": 2,"ack": 1,"tx_time": 168508,"juf": 128,"period": 15,"manuf": 244,"rstr": 20},{"timestamp": "2020-02-05T13:10:34Z","mac": "D0F01843F3D8","rssi": -64,"rawData": "0201061107694E6F646520426561636F6E0000000003FF0080020A08","rawResp": "0D09694E6F64652D343346334438"},{"timestamp": "2020-02-05T13:10:35Z","mac": "35FD6890709F","rssi": -64,"rawData": "1EFF06000109200240FFC4543224734A4054BEABBFCF2DE413BBB209EC8750","rawResp": ""},{"timestamp": "2020-02-05T13:10:35Z","mac": "1DB6F43813AF","rssi": -66,"rawData": "1EFF060001092002468330 7B82213C7F54DD5BBB25CE60CE90ED7FC8E15B97","rawResp": ""},{"timestamp": "2020-02-05T13:10:34Z","mac": "FE2BD474F9EC","rssi": -57,"rawData": "031941030201060303E7FE09FFF8F8ECF974D42BFE0409503131","rawResp": ""},{"timestamp": "2020-02-05T13:10:35Z","mac": "D0F01843DB0F","rssi": -57,"rawData": "02010603FFC088020AFE0D09694E6F64652D343344423046","rawResp": ""},{"timestamp": "2020-02-05T13:10:34Z","mac": "D0F01843F3DA","rssi": -79,"rawData": "02010619FFC09B01B000000000931800000F00CCF6B00428FEE4821F30","rawResp": "0D09694E6F64652D343346334441020AFE"},{"timestamp": "2020-02-05T13:10:33Z","mac": "D0F01843F15C","rssi": -77,"rawData": "0201060EFFA0820000C31300006440B100A0020A08","rawResp": "0D09694E6F64652D343346313543"},{"timestamp": "2020-02-05T13:10:33Z","mac": "D8A01D6B8E1A","rssi": -79,"rawData": "0F0843656E7472616C2D364238453141","rawResp": ""},{"timestamp": "2020-02-05T13:10:35Z","mac": "30F7724CCFF0","rssi": -83,"rawData": "1AFF4C00021550765CB7D9EA4E2199A4FA879613A49270D8B708CE","rawResp": ""},{"timestamp": "2020-02-05T13:10:34Z","mac": "D0F01843F3DB","rssi": -84,"rawData": "02010619FF909B01C000000000471836100F0090F6D67324F4862721D3","rawResp": "0D09694E6F64652D343346334442020AFE"},{"timestamp": "2020-02-05T13:10:34Z","mac": "D0F01843F1E4","rssi": -52,"rawData": "0201060EFF908200004D000000E803B20080020AFE","rawResp": "0D09694E6F64652D343346314534"},{"timestamp": "2020-02-05T13:10:34Z","mac": "D0CF5E03930E","rssi": -66,"rawData": "0201060EFFA08200009F000000E803A00000020A14","rawResp": "0D09694E6F64652D303339333045"},{"timestamp": "2020-02-05T13:10:34Z","mac": "D0F01843E2C5","rssi": -74,"rawData": "02010619FF129D01B00004643F3519D812390034FE30FD8292C68368FC","rawResp": "0D09694E6F64652D343345324335020A06"},{"timestamp": "2020-02-05T13:10:34Z","mac": "D0F01843F2FA","rssi": -72,"rawData": "0201060EFFA08200001000000E843B00000020AFE","rawResp":

"0D09694E6F64652D343346324641"},{"timestamp": "2020-02-05T13:10:34Z","mac": "D0F01843F3D5","rssi": -74,"rawData": "0201061107694E6F646520426561636F6E0000000003FF0080020AFE","rawResp": "0D09694E6F64652D343346334435"},{"timestamp": "2020-02-05T13:10:34Z","mac": "D0F01843F2D9","rssi": -70,"rawData": "02010619FF1A9500C00000B807F81F46001B001ADC26436A4EC8AAD4CF","rawResp": "0D09694E6F64652D343346324439020A08"},{"timestamp": "2020-02-05T13:10:34Z","mac": "D0CF5E039918","rssi": -74,"rawData": "02010619FF009BFF80000000009E1813130000A9FEC617BB7D98BD395D","rawResp": "0D09694E6F64652D303339393138"},{"timestamp": "2020-02-05T13:10:33Z","mac": "D0F01843F3DC","rssi": -79,"rawData": "02010619FF909101B00000077CA08000000F0090F605A530C80FB0D0F4","rawResp": "0D09694E6F64652D343346334443020AFE"},{"timestamp": "2020-02-05T13:10:35Z","mac": "D0F01843F150","rssi": -79,"rawData": "02010619FF149E00C000006D02A10000000900A24846D5481621C47BE9","rawResp": "0D09694E6F64652D343346313530020A08"},{"timestamp": "2020-02-05T13:10:33Z","mac": "D0F01843F458","rssi": -79,"rawData": "02010619FF129D01C000047C3FD818C80E0000163E538DEF07F9AF5D84","rawResp": "0D09694E6F64652D343346343538020AFE"},{"timestamp": "2020-02-05T13:10:34Z","mac": "0CF3EEA355A1","rssi": -74,"rawData": "02010417FF0C0300B8DC2023FEE4110FBE2000145E45041DBF000003030118","rawResp": ""}]}

## 2.2. Processed JSON data

## 2.3. Encrypted JSON data:

If the data from **iNode LAN Central** is encoded at the beginning of the JSON file, there is a ***key*** field. This is a temporary key used to encrypt JSON data. It is encrypted with the master key entered into **iNode LAN Central**. The data is encrypted with the ARCFour algorithm.

{"key": "33FE46D546832247CC91A8EA733D56E9","data": [Q}H/k_ {.mZu-
GzTwe&Zn&v1HTT}qEIm&qqO J)};)xmn
+eUCIA7!j*@]+/k'v0Ee3@^f[+">
yaKB\9:IBl]kA6hY(}w|SL_h)
j.xOLTS@@
T_O
Q1]}

## *2.4.* JSON data decryption

Below is an example function in JavaScript that decodes encrypted JSON data. The jsaes.js file can be downloaded from:
**https://support.inode.pl/apps/iNodeMqttMonitor/js/jsaes.js**

```
/*
 * RC4 symmetric cipher encryption/decryption
 * @license Public Domain
 * @param string key - secret key for encryption/decryption
 * @param string str - string to be encrypted/decrypted
 * @return string
 */

var rc4_s = [];
var rc4_i;
var rc4_j;

function rc4_init(rc4_key, rc4_key_length)
{
   var j = 0, x;
   for (var i = 0; i < 256; i++) {
      rc4_s[i] = i;
   }
   for (i = 0; i < 256; i++) {
      j = (j + rc4_s[i] + rc4_key[i % rc4_key_length]) % 256;
      x = rc4_s[i];
      rc4_s[i] = rc4_s[j];
      rc4_s[j] = x;
   }

   rc4_i=0;
   rc4_j=0;
}

function rc4_get_xor_byte()
{
   var x;

   rc4_i = (rc4_i + 1) % 256;
   rc4_j = (rc4_j + rc4_s[rc4_i]) % 256;
   x = rc4_s[rc4_i];
   rc4_s[rc4_i] = rc4_s[rc4_j];
   rc4_s[rc4_j] = x;
   return rc4_s[(rc4_s[rc4_i] + rc4_s[rc4_j]) % 256];
}

var JSON_USER_KEY = new Array(16);
var JSON_DECRYPT_KEY = new Array(16);

function searchCommentValue(sstr, key)
{
   var offset_start=sstr.search(key);

   if(offset_start>=0)
   {
      var rsstr=sstr.slice(offset_start+key.length);
      var offset_end=rsstr.search('"');
      return rsstr.substring(0,offset_end);
```

```
    }
    else
    {
       return "";
    }
}

function decodeJSON(json_raw, json_key)
{
    var img_dataView = new DataView(json_raw);
    var ik;
    var img_byte;
    var xor_byte=0;
    var ik_img_offset=0;

    for(var i = 0; i < 16; i++)
    {
       JSON_USER_KEY[i] = 0;
    }

    for(var i = 0; i < json_key.length; i+=2)
    {
       JSON_USER_KEY[15-i] = json_key.charCodeAt(i+1);
       JSON_USER_KEY[14-i] = json_key.charCodeAt(i);
    }

    var JSON_KEY=searchCommentValue(ab2str(json_raw),'{"key": "');

    if(JSON_KEY.length!=0)
    {
       AES_Init();

       var block = new Array(16);
       for(var i = 0; i < 16; i++)
          { block[15-i] = parseInt(JSON_KEY.substr(i*2,2), 16) };

       var key = new Array(16);
       for(var i = 0; i < 16; i++)
          { key[i] = JSON_USER_KEY[i]; }

       AES_ExpandKey(key);
       AES_Decrypt(block, key);

       for(var i = 0; i < 16; i++)
          { JSON_DECRYPT_KEY[15-i] = block[i] };

       AES_Done();

       rc4_init(JSON_DECRYPT_KEY,16);

       var json_data_start=ab2str(json_raw).search(',"data": ')+10;

        for(ik=json_data_start;ik<(img_dataView.byteLength-2);ik++)
       {
          img_byte=img_dataView.getUint8(ik);

          img_dataView.setUint8(ik,(img_byte^rc4_get_xor_byte())& 0xff);
       }
    }
```

```
    return json_raw;
}
```

## 2.5. BLE data decoding:

Data coding scheme in advertisement frame and response for active scan. Information about **AD Type** codes can be found in a Core_V4.0.pdf: Volume 3 Part C, Section 8. and at the page https://www.bluetooth.org/en-us/specification/assigned-numbers/generic-access-profile

advertisement frame:
***02010619FF12930110000017 00AB18951F485435BE5B809D6F571E40E8FE0000***

***020106***
***02*** -> length of the data field: 2 bytes
***0106*** -> data
***01*** -> 0x01 -> EIR Data Type = 0x01 -> «Flags»
***06*** -> 0x06 -> EIR Data = 0x06 -> LE General Discoverable Mode (bit 1), BR/EDR Not Supported (bit 2)

***19FF12930110000017 00AB18951F485435BE5B809D6F571E40E8***
***19*** -> length of the data field: 25 bytes
***FF12930110000017 00AB18951F485435BE5B809D6F571E40E8*** -> data (25 bytes)
***FF*** -> 0xFF -> EIR Data Type = 0xFF «Manufacturer Specific Data»
***12930110000017 00AB18951F485435BE5B809D6F571E40E8*** ->
***1293*** -> 0x9312 -> 0x93XX iNodeCareSensor #3 identifier; 0xXX1X version 1; 0xXXX2 since last memory readout lasts 24 hours;
***0110*** -> 0x1001 type -> bit 15 to bit 12 -> reserved, bit 11 to bit 0 -> sensor group address
***0000*** -> 0x0000 flags ->

```
SENSOR_ALARM_MOVE_ACCELEROMETER=1,
SENSOR_ALARM_LEVEL_ACCELEROMETER=2,
SENSOR_ALARM_LEVEL_TEMPERATURE=4,
SENSOR_ALARM_LEVEL_HUMIDITY=8,
SENSOR_ALARM_CONTACT_CHANGE=16,
SENSOR_ALARM_MOVE_STOPPED=32,
SENSOR_ALARM_MOVE_GTIMER=64,
SENSOR_ALARM_LEVEL_ACCELEROMETER_CHANGED=128,
SENSOR_ALARM_LEVEL_MAGNET_CHANGE=256,
SENSOR_ALARM_LEVEL_MAGNET_TIMER=512
```

***1700*** -> 0x0017 value1
/* motion sensor */
0x8000 sensor is in move (bit 15 =1)
bit 14 to 10:
X-axis (5 bit value as 2's complement number) -> 0x00= 0

bit 9 to 5:
Y-axis (5 bit value as 2's complement number) -> 0x00= 0
bit 4 to 0:
Z-axis (5 bit value as 2's complement number) -> 0x17= -9

***AB18*** -> 0x18AB value2
/* temperature sensor */
Temperature= ((175.72 * Temp_Code)/65536)-46.85 [°C]
Temp_Code = 0x18AB *4 = 0x62AC = 25260
Temperature = 20,879 °C

***951F*** -> 0x1F95 value3
/* humidity sensor */
%RH= ((125*RH_Code)/65536)-6 [%]
RH_Code = 0x1f95 *4 = 0x7e54 = 32340
%RH= 55,68 %

***485435BE*** -> 0x5448BE35 time (time stamp; number of seconds since 01.01.1970)

***5B80 9D6F 571E 40E8*** -> an AES128 digital signature of all data

response for an active scan:

***0D09694E6F64652D333536313441020A020000000000000000000000000000***

***0D09694E6F64652D333536313441***

***0D*** -> length of the data field: 13 bytes

***09694E6F64652D333536313441*** -> data

***09*** -> 0x09 -> EIR Data Type = 0x09 -> «Complete Local Name»

***694E6F64652D333536313441*** -> iNode-35614A

***020A02***

***02*** -> length of the data field: 2 bytes

***0A02*** -> data

***0A***-> 0x0A -> EIR Data Type = 0x0A -> «Tx Power Level»

***02*** -> 0x02 -> Tx Power Level = +2dBm