Securing IoT with the TEE

- Develop a smart parking system with Trustonic TEE and Samsung ARTIK platform

Oct 14th, 2016, TEE Conference Developer Workshop

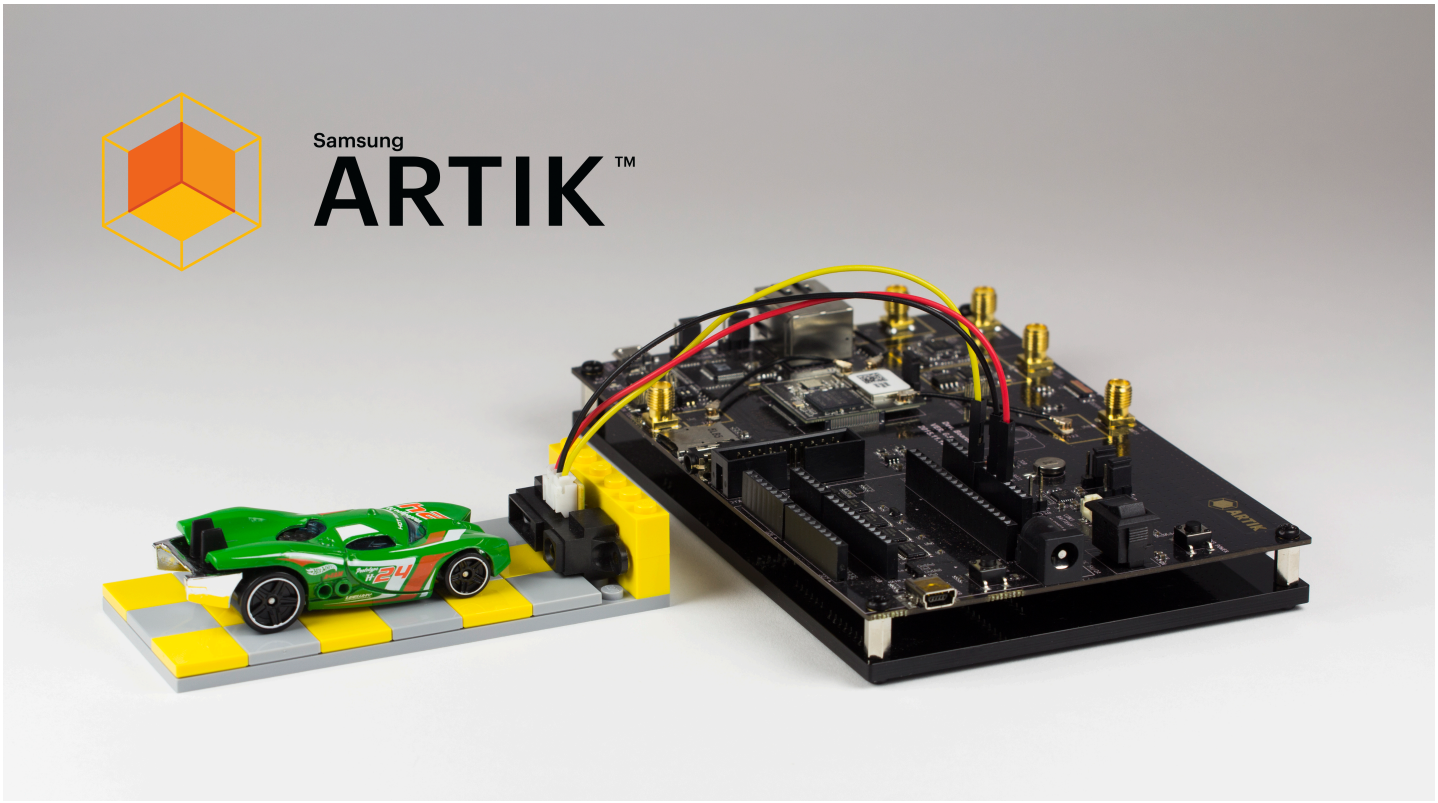


Table of Contents

Introduction

Welcome to the Trustonic/Samsung workshop! This workshop is your introduction to ARM TrustZone technology and how to secure your IoT application with Trustonic TEE on Samsung ARTIK platform. In this session, you will learn how to:

- Generate and protect an Elltiptic curve keypair using the Trustonics TEE
- Sign sensor dta using Elliptic Curve DS(ECDSA)
- Build program flow using Node-RED
- Collect sensor data from ARTIK GPIO
- Connect to ARTIK Cloud

Before you begin

1. Bring your own laptop. For Windows users, pre-install PuTTY.

   PuTTY (http://www.putty.org/) – SSH and Telnet client, for serial console access

2. Establish an ARTIK Cloud user portal account: We will stream data to ARTIK Cloud service. Create an account at ARTIK Cloud user portal (https://artik.cloud/).

Setup

1. Access ARTIK from your serial console. Instructions is at https://developer.artik.io/documentation/getting-started-beta/communicating-pc.html

2. Connect ARTIK to the network.

   2.1 Ethernet. Plug an Ethernet cable into your ARTIK Ethernet port, do a 'ping' test from your console and take note of your board IP address.

```
[root@localhost ~]# ping www.google.com
PING www.google.com (172.217.3.36) 56(84) bytes of data.
64 bytes from nuq04s18-in-f4.1e100.net (172.217.3.36): icmp_seq=1 ttl=52 time=8.83 ms
64 bytes from nuq04s18-in-f4.1e100.net (172.217.3.36): icmp_seq=2 ttl=52 time=59.9 ms
64 bytes from nuq04s18-in-f4.1e100.net (172.217.3.36): icmp_seq=3 ttl=52 time=17.2 ms
```

   (Ctrl-C to terminate)

```
[root@localhost ~]# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.0.0.2  netmask 255.255.255.0  broadcast 10.0.0.255
    inet6 2601:647:4e01:7b45:489d:21ff:fe85:6c27  prefixlen 64  scopeid 0x0<global>
    inet6 fe80::489d:21ff:fe85:6c27  prefixlen 64  scopeid 0x20<link>
    ether 4a:9d:21:85:6c:27  txqueuelen 1000  (Ethernet)
    RX packets 14769  bytes 20317291 (19.3 MiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 311790 (304.4 KiB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

   2.2 WiFi.(Optional) In place of Ethernet, set up WiFi on your ARTIK, and do the 'ping' test as noted above.

   https://developer.artik.io/documentation/developer-guide/configuring-wifi-on-artik-10.html - configuring-wifi-on-artik-5-and-10

Exercise 1: Monitor the state of your parking space (Node-RED and C programming language)

1. Wire up your distance sensor to your ARTIK board. ARTIK 5 has two analog pins exposed: J24 A0 and J24 A1. There are 3 wires on your distance sensor:

- 5V (red wire) connects to header J25 5V
- GND (black wire) connects to header J25 GND
- Signal (yellow wire) connects to header J24 A0

2. Read the analog GPIO pin using *sysfs*, which allows sensor data to be read from a system file. To get sensor data on J24 A0, copy the highlighted text below to your terminal and hit [Enter].

```
[root@localhost ~] # cat  /sys/devices/126c0000.adc/iio:device0/in_voltage0_raw
1413
```

Move your hand towards or away from the distance sensor, and repeat the command above by typing [Up] and then [Enter]. You should observe that the output value changes.

3. Compile SensorReading.c under /home directory. Launch SensorReading to test it reads sensor data correctly.

```
[root@localhost ~] # cd /root/
[root@localhost ~] # gcc SensorReading.c –o SensorReading
[root@localhost ~] # ./SensorReading
820
```

4. Start Node-RED in the background from your ARTIK serial console.

```
[root@localhost ~]# node-red &
…
Welcome to Node-RED
===================

12 Feb 13:53:43 - [info] Node-RED version: v0.13.1
12 Feb 13:53:43 - [info] Node.js  version: v0.10.36
…
12 Feb 13:53:48 - [info] Server now running at http://127.0.0.1:1880/
```

5. Open a browser on your laptop, and launch http://<your_board_ip_address>:1880/

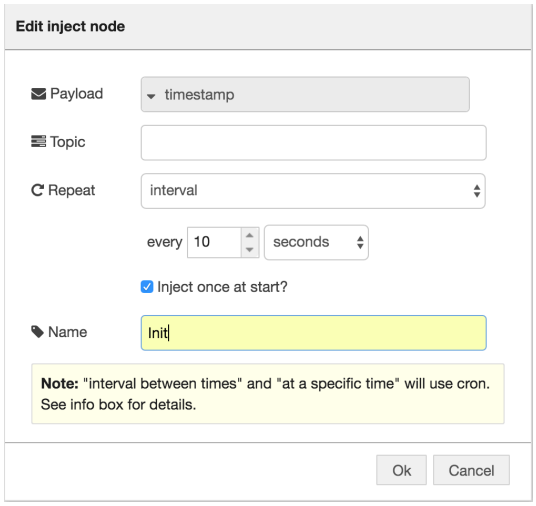
6. Design your first project flow.

6.1 Create a flow on the Node-RED canvas.

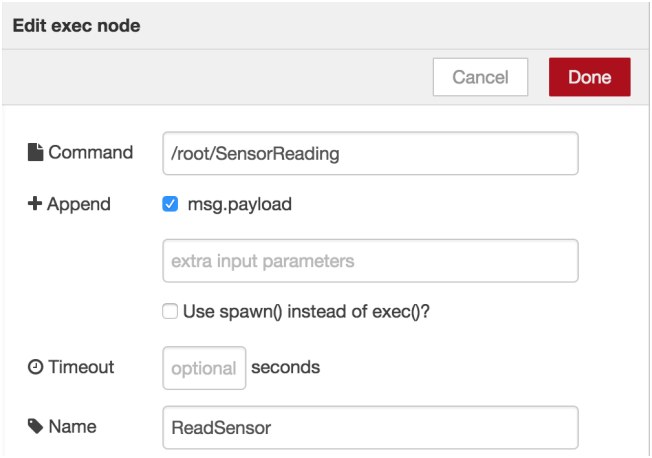
- Drag an “inject” input node from the node palette to the canvas (it initially shows “timestamp”).
- Drag an “exec” node to the right of the first node.
- Drag a “debug” output node to the right of the second node.
- Connect these 3 nodes by dragging a “wire” from the right side of the “inject” node to the left side of the “exec” node, then from the 1st output of “exec” node to the left side of the “debug” node.

6.2 Configure each node by double-clicking.

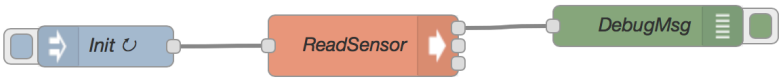
- “inject” node(shows as “timestamp”): Set Repeat to a 10s interval, enable ‘Inject once at start’ and rename the node to “Init”. Your flow will read the sensor data when it is first launched, and continue to read it every 10 seconds.



- “exec” node: Set Command as “/root/SensorReading”. Name as “ReadSensor”.



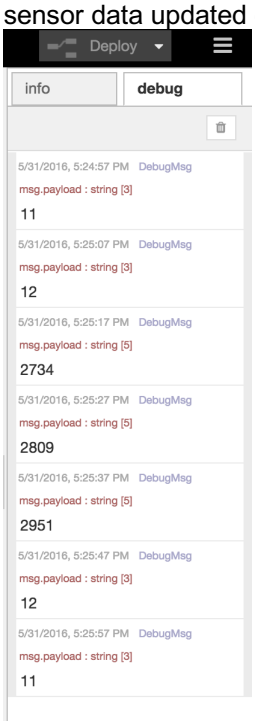
- "debug" node: Name as "DebugMsg". Here is our final flow.



6.3 Run the flow.

Click 

at the upper right corner to make your application live. You should be able to see your sensor data updated every 10 seconds in the Debug tab.



Exercise 2: Track Parking Lots Occupancy and stream the data to ARTIK Cloud(ARTIK Cloud, Node-RED, MongoDB)

In exercise 2, we will monitor if our parking space is occupied. ARTIK streams data to backend MongoDB, including parking space states and most importantly, if the parking space data comes from a secured source, so we can monitor real-time occupancy of our parking lots. We also use ARTIK Cloud for data visualization.

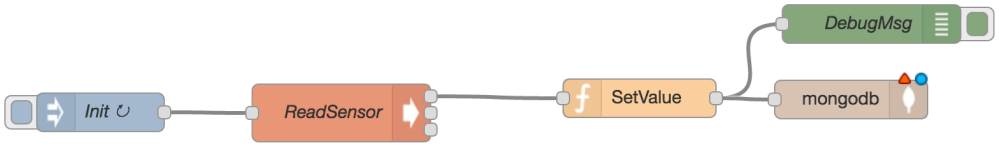
1. Use MongoDB to track the parking space occupancies.

We will mark our parking lot number, space number, space state and if the data comes from a secured source in our backend MongoDB.

1.1

- Drag one “function” node to the right of the ReadSensor node.
- Drag a “mongodb” **output** node 

to the rightmost.
- Connect the nodes as shown. We can also use “DebugMsg” node to verify the msg.payload output from “SetValue” function.



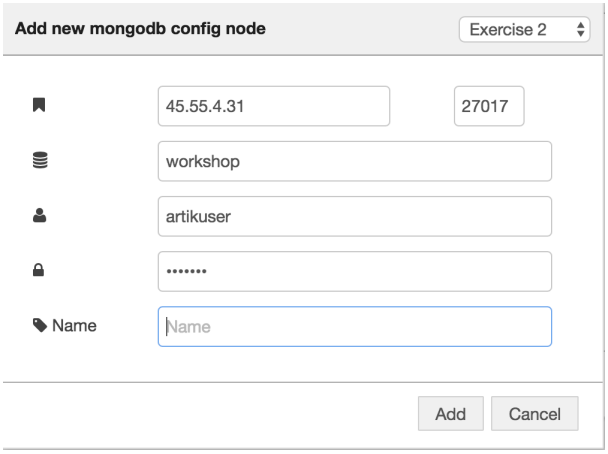
1.2 Double click each node to configure.

- In “function” node, define function as below, rename node to “SetValue”. Here we check the parking space state and format the msg.payload for MongoDB.

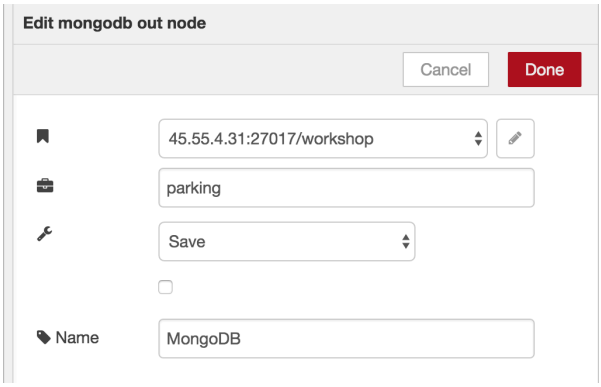
```
if (Number(msg.payload) >= 2000)
    state = "full";
else
    state = "empty";

msg.payload = {
    "lot" : 9,          //Replace "9" with your own parking lot number
    "space": 1,         //Replace "1" with your own parking space number
    "state": state,
    "secured": false
}
msg._id = "wei";  //Replace _id by using an unique name
return msg;
```

- “MongoDB” output node: We need to configure our remote MongoDB host as below.

Click  in the “Edit mongodb out node”dialog, “Add new mongodb config node” opens up. Enter “45.55.4.31” as host ip address, “27017” as port number, and “workshop” as database name. Use “artikuser” and “iot2016” as your username and password.



Click “Add” (or “Update” if changing existing parameters) to return to the “Edit mongodb out node” dialog. Enter “parking” as the collection name, “Save” as the operation. Rename the node to “MongoDB”.



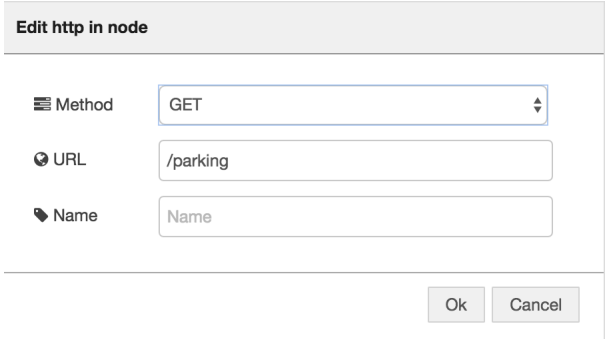
2. Display real-time parking occupancy on a web page.

2.1 Drag an “http” input node, a “mongodb in” node 

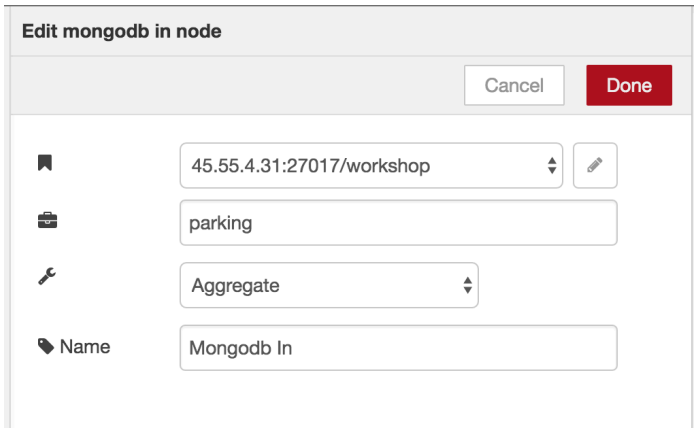
, a “template” node and a “http response” node to the canvas. Wire them up

Configure each node by double clicking:

- In “http” input node, we define our parking status URL to be “/parking”.

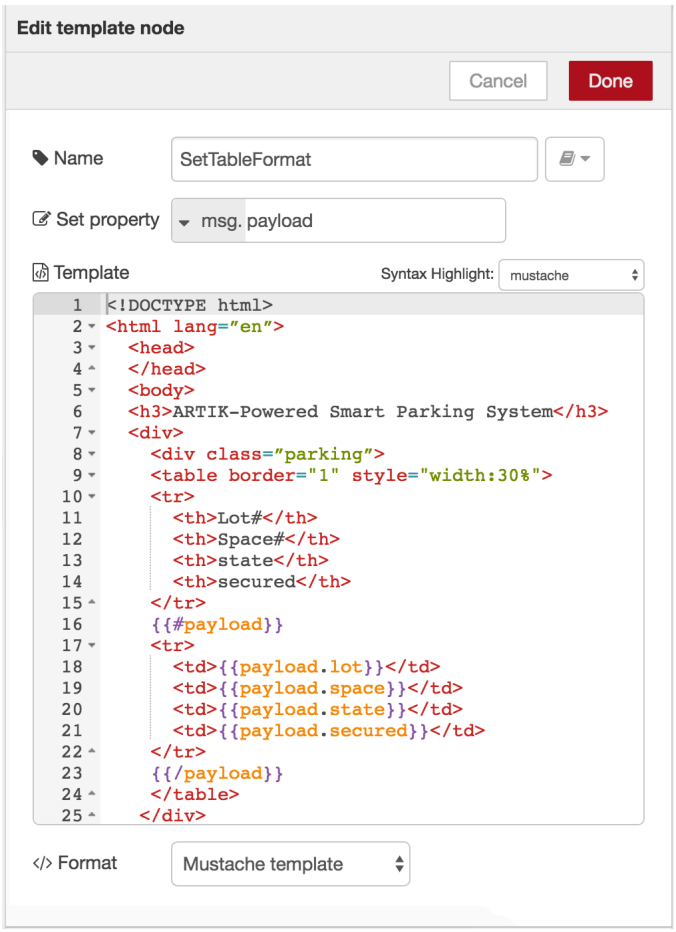


- In “Mongodb” in node, it should have picked up our settings from the MongoDB output node configuration above. Add “parking” as collection name, “Aggregate” as the operation name. Rename the node as “Mongodb In”.



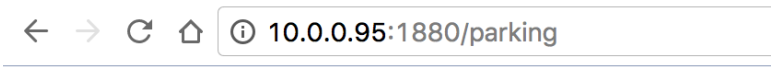
- In “template” node, here is the code you need to copy over. Rename to “SetTableFormat”.

```
<!DOCTYPE html>
<html lang=”en”>
 <head>
 </head>
 <body>
  <h3>ARTIK-Powered Smart Parking System</h3>
  <div>
    <div class=”parking”>
      <table border="1" style="width:30%">
      <tr>
        <th>Lot#</th>
        <th>Space#</th>
        <th>Status</th>
        <th>Secured</th>
      </tr>
      {{#payload}}
      <tr>
        <td>{{payload.lot}}</td>
        <td>{{payload.space}}</td>
        <td>{{payload.state}}</td>
        <td>{{payload.secured}}</td>
      </tr>
      {{/payload}}
      </table>
    </div>
  </div>
 </body>
</html>
```



Here is what our final flow looks like:

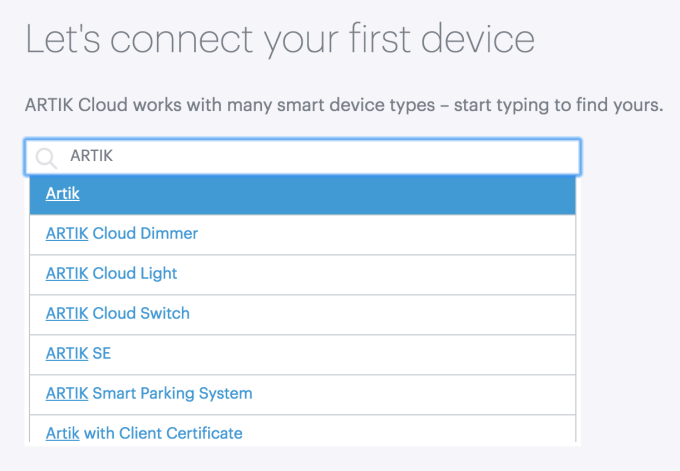
Now, deploy your changes. Open a browser and launch <your_ARTIK_IP_address>:1880/parking, you will be able to see the real-time occupancy of our parking lots.



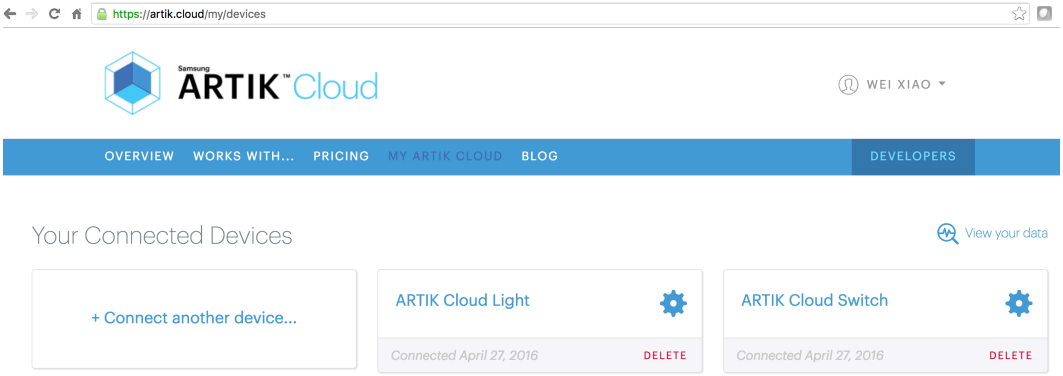
ARTIK-Powered Smart Parking System

Lot#	Space#	state	secured
9	1	empty	false
9	2	full	false
9	3	full	false

3. As the next step, we will stream our parking space states to ARTIK Cloud. Log into ARTIK Cloud user portal https://artik.cloud.
   3.1 If this is your first device, you will be re-directed to the screen below:

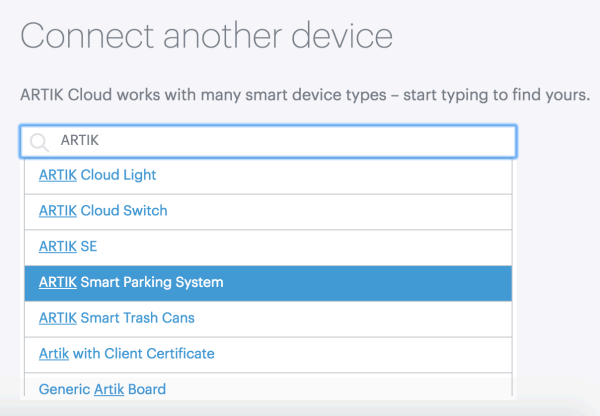


Otherwise, Under “MY ARTIK CLOUD”/”DEVICES”, you will see all you connected devices.

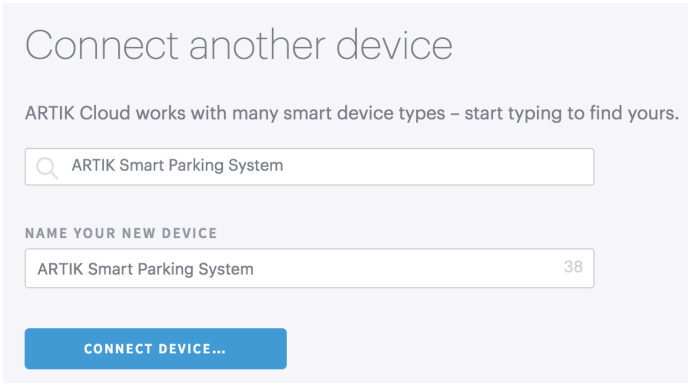


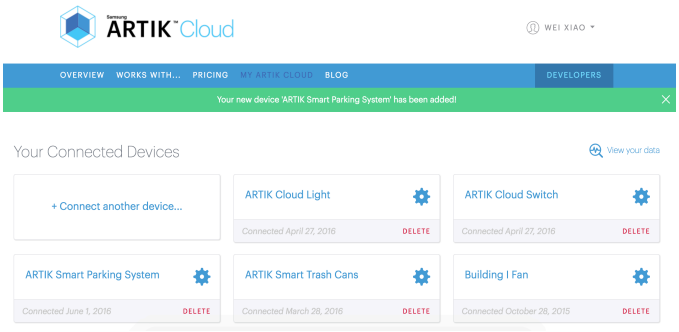
Click “Connect another device…” link.

Search for “ARTIK Smart Parking System” and select it. “ARTIK Smart Parking System” is a public device type we created for the workshop, which includes “lot”(a number), “space”(a number), “state”(“empty” or “full”)

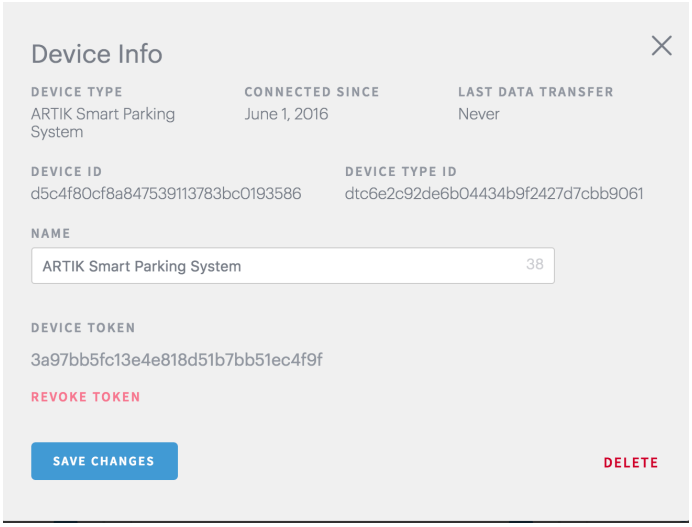


3.2 Use the default device name “ARTIK Smart Parking System”, then click the “CONNECT DEVICE…” button. A new device will be created.





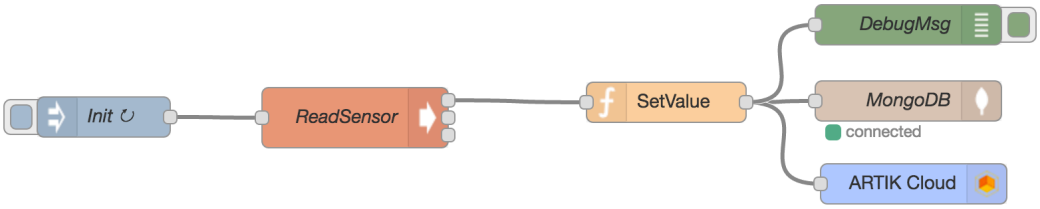
3.3 Click the Gear icon next to your newly created device, you will see the Device Info popup, which shows your Device Type, Device ID, Device name etc. details. Click the “GENERATE DEVICE TOKEN…” link to generate a device token.



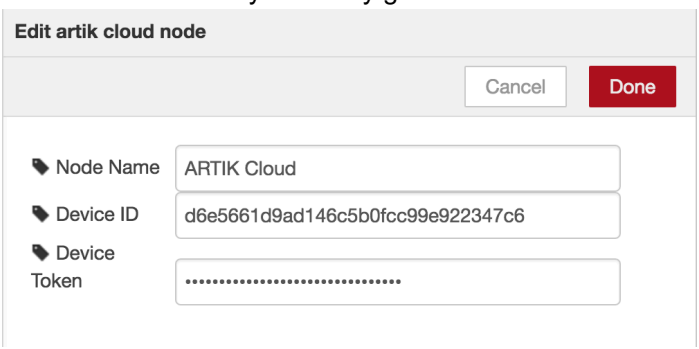
We need to use the **Device ID** (not “DEVICE TYPE ID”) and **Device Token** to associate our parking space with ARTIK Cloud. Make a copy or leave the dialog open.

3.4 Drag an ARTIK Cloud “output” node 

to the canvas, and put it under the “MongoDB” node. Here is what the flow looks like.



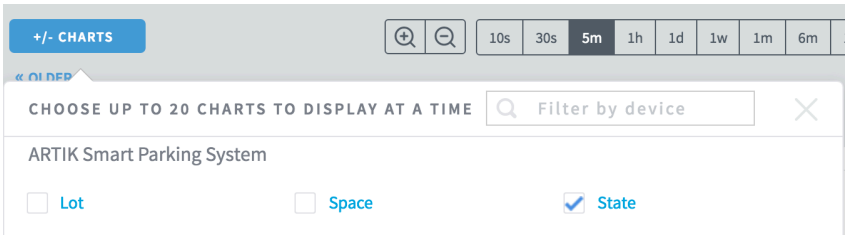
Double click to enter your newly generated Device ID and token. Rename to “ARTIK Cloud”.



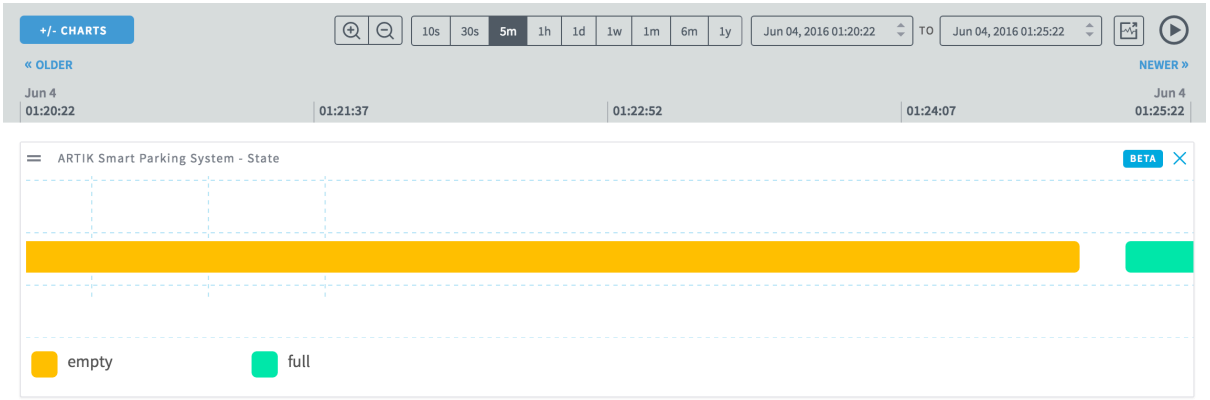
Click 

at the upper right corner to make your application live.

Go to ARTIK Cloud user portal, click the “+/- CHARTS” button, and enable to view the “State” of your parking space.



You will see the streamed parking space states in your portal:



Exercise 3: Build a trusted application

In exercise 3, we will monitor if our parking space data comes from a secured source

1. Run build script to compile our trusted application

```
[root@localhost ~]#cd /root/tools/t-sdk-r11.1/Samples/DataSign/
[root@localhost Code]#./buildall-artik.sh
```

2. Execute compiled trusted application *caDataSign* from command line.

2.1 Generate the 256bit ECC key pair with: *caDataSign --genkey*

```
[root@localhost ~]#cd /root/tools/t-sdk-r11.1/Samples/DataSign/CADataSign/Locals/Code/
[root@localhost Code]#./caDataSign –genkey | grep -E -v 'I/|D/|E/'
```

2.2 Get our public key: *caDataSign --getpub*

```
 [root@localhost Code]#   ./caDataSign --getpub | grep -E -v 'I/|D/|E/'

B28888F90ADBE22F4114E136C666F817BCEAAD53AC54D59E3717C5EB5C27663E05CEC5CA868A8724
ADB0D041A6EE932CAE222980EEE5411AA31B2203505D34C8
```

2.3 Sign the sensor data: *caDataSign –sign <analog_pin_sysfs>* OR *caDataSign –signvalue <sensor_value>*

```
 [root@localhost Code]#./caDataSign --sign /sys/devices/126c0000.adc/iio:device0/in_voltage0_raw |
grep -E -v 'I/|D/|E/'
1E960D13D6FB3FB05E0D9763EABA8E08520AF0528B6E6B247ECE4F8FADBDB809F212A4800767C75B
58BD0768A3C8AE7D8A76D8C4CF9AA4B3AF6C8540F3330A17

OR
[root@localhost Code]#   ./caDataSign --signvalue 1234 | grep -E -v 'I/|D/|E/'
FA8C4A9C728C801AB44DA8B330B1CE25E6A9F296C82B790F17225874D1178858B217760CD86625D52
873650A38E885AC7AC8389B80822E21BB173D0D6F5307EA
```

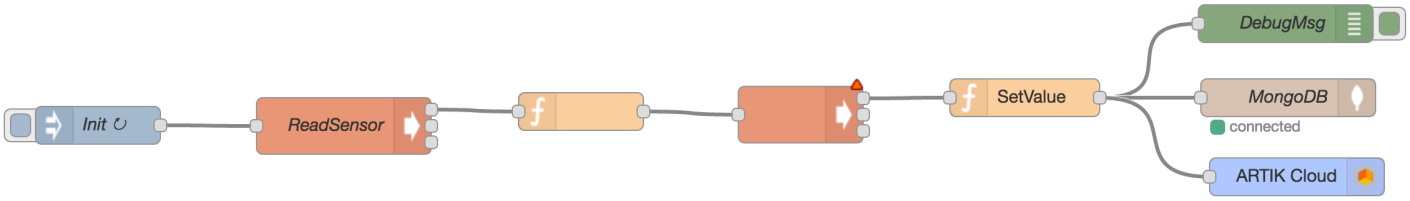
2.4 In the final step, we can verify the signature or any other signatures from another ARTIK with: *caDataSign –verify <sensor_value> <signature><public_key>*, where *<signature>* is the output generated in step 2.3 above and *<public_key>* is the output generated in step 2.2.

```
 [root@localhost Code]#./caDataSign --verify 1234
1E960D13D6FB3FB05E0D9763EABA8E08520AF0528B6E6B247ECE4F8FADBDB809F212A4800767C75
B58BD0768A3C8AE7D8A76D8C4CF9AA4B3AF6C8540F3330A17
B28888F90ADBE22F4114E136C666F817BCEAAD53AC54D59E3717C5EB5C27663E05CEC5CA868A872
4ADB0D041A6EE932CAE222980EEE5411AA31B2203505D34C8 | grep -E -v 'I/|D/|E/'
Signature verified
```

3. Now, we will integrate our trusted application into our Node-RED flow:

3.1 Insert a “function” node and an “exec” node between “ReadSensor” node and “SetValue” node.

- In “function” node, we save sensor reading into a global variable, and set msg.payload as the parameter for obtaining public key.

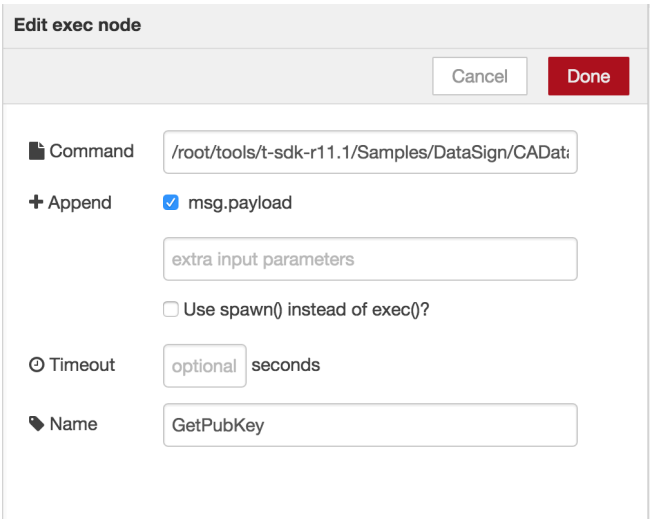


- In “function” node, we define our function as below. Rename to “Func1”.

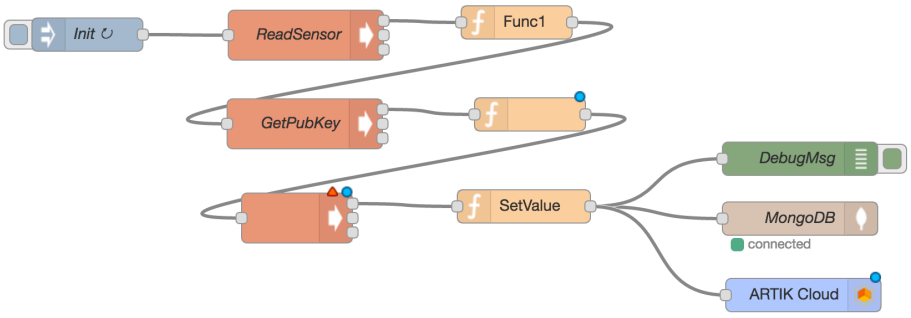
```
var str = msg.payload;
str = str.replace(/(\r\n|\n|\r)/gm,"");
global.set('distance', str);

var parameter = '--getpub ';
parameter = parameter.concat(" | grep -E -v 'I/|D/|E/'");
console.log(parameter);
msg.payload = parameter;

return msg;
```

- In “exec” node, we run trusted application with msg.payload as its parameter. This will obtain public key.

  Rename to “GetPubKey”.



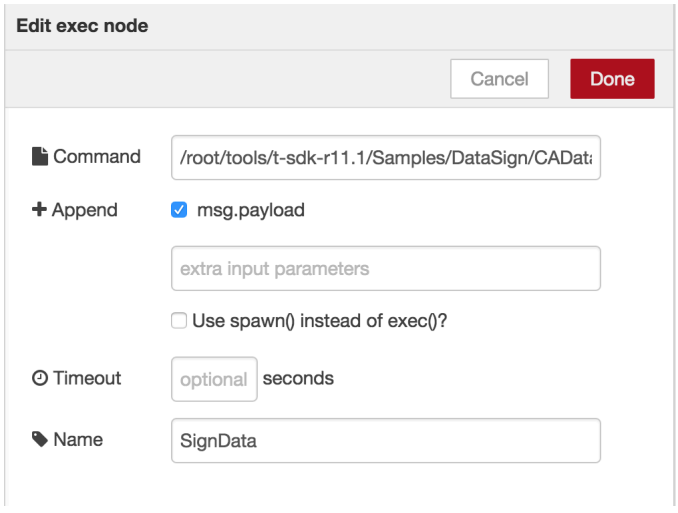
3.2 We will follow the same steps to sign the sensor reading data. Insert a “function” node and an “exec” node between “GetPubKey” node and “SetValue” node. In “function” node, we save our public key into a global variable, and set msg.payload as the parameter for signing the sensor data.



- Double click “function” node, and define our function as below. Rename to “Func2”.

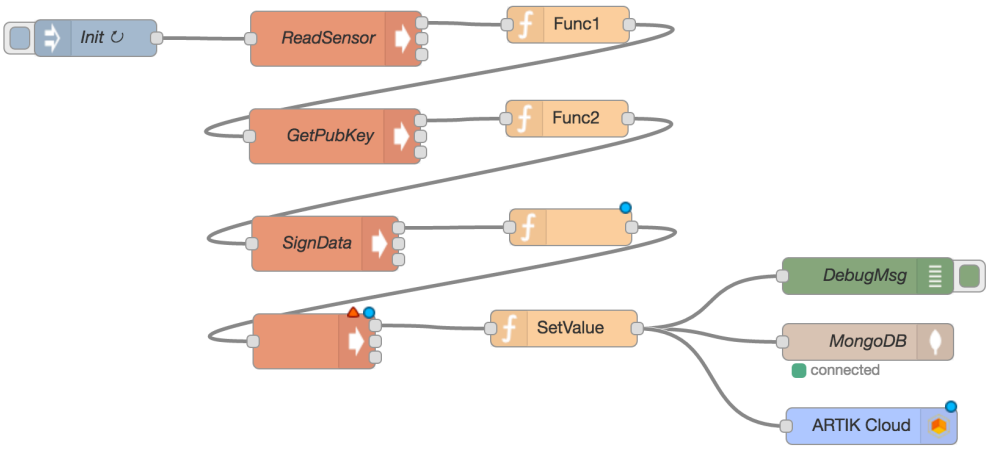
```
var str = msg.payload;
str = str.replace(/(\r\n|\n|\r)/gm,"");
global.set('pubkey', str);

var parameter = '--signvalue ';
parameter = parameter.concat(global.get('distance'));
parameter = parameter.concat(" | grep -E -v 'I/|D/|E/'");
console.log(parameter);
msg.payload = parameter;

return msg;
```

- In “exec” node, we run trusted application with msg.payload as its parameter. This will sign the sensor reading data. Rename to “SignData”.



3.3 Now we will verify the signed data. Again, insert a “function” node and an “exec” node between “SignData” node and “SetValue” node.

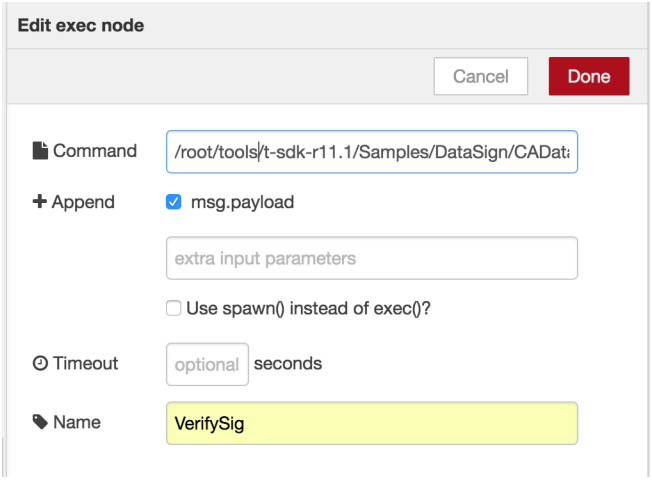
- In “function” node, we save signed data into a global variable, and set msg.payload as the parameter for verifying the signature.



- In “function” node, we define our function as below. Rename to “Func3”.

```
var str = msg.payload;
str = str.replace(/(\r\n|\n|\r)/gm,"");
global.set('signeddata', str);

var parameter = '--verify ';
parameter = parameter.concat(global.get('distance'));
parameter = parameter.concat(' ');
parameter = parameter.concat(global.get('signeddata'));
parameter = parameter.concat(' ');
parameter = parameter.concat(global.get('pubkey'));
console.log(parameter);
msg.payload = parameter;
return msg;
```

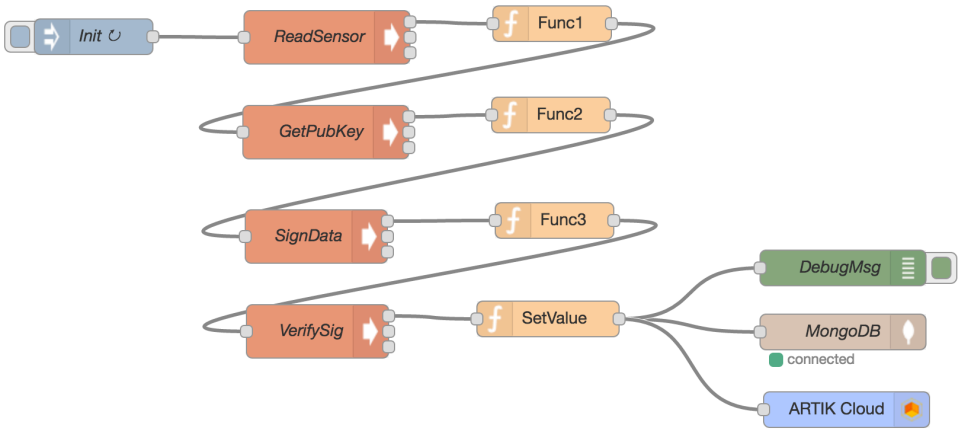
- In “exec” node, we run trusted application with msg.payload as its parameter. Rename to “VerifySig”.



3.4 We need to make few changes in “SetValue” function.

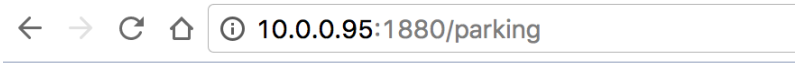
```
var str = msg.payload;
var secured = false;

if (str.search("Signature verified") != -1)
   secured = true;
else if (str.search("Signature invalid") != -1)
   secured = false;

if (Number(global.get('distance')) >= 2000)
  state = "full";
else
  state = "empty";

msg.payload = {
    "lot" : 9,        //Replace "9" with your own parking lot number
    "space": 1,       //Replace "1" with your own parking space number
    "state": state,
    "secured": secured
}
msg._id = "wei";  //Replace _id by using an unique name return msg;
return msg;
```

Here is what our final flow looks like:



3.5 Click 

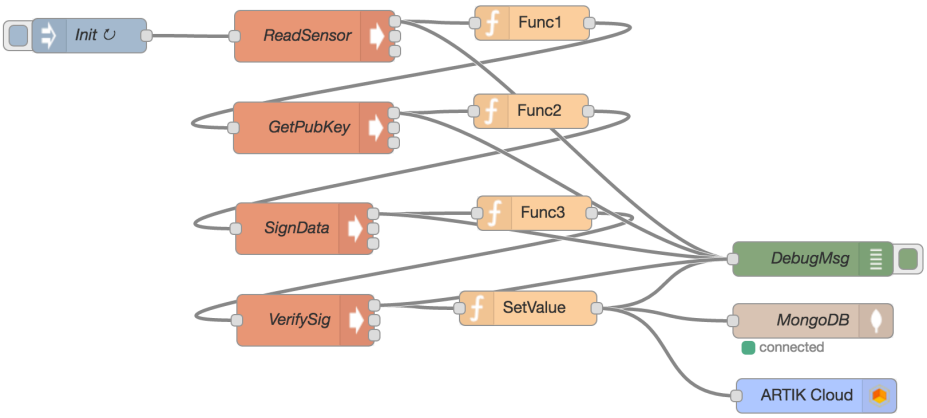
at the upper right corner to make your application live. In our /parking page, we will be able to see real-time secured data.



ARTIK-Powered Smart Parking System

Lot#	Space#	Status	Secured
9	1	empty	true
9	2	full	false
9	3	full	false

In order to get more debugging info, you can also wire the stout of each function to the DebugMsg node.



And you will be able to get the output on Debug panel.