

# Guide for the Project Laboratory: Secure SoC for the Internet-of-Things

SoSe2022

## Contents

<b>1</b>	<b>Preliminary remarks</b>	<b>3</b>
1.1	Academic integrity . . . . .	3
1.2	Documentation . . . . .	3
1.2.1	Code . . . . .	3
1.2.2	Your Work - The lab report . . . . .	3
1.3	Getting help with tasks and problems . . . . .	3
<b>2</b>	<b>Lab Setup</b>	<b>4</b>
2.1	Set up a Virtual Machine (VM) with all tools needed for the lab . . . . .	4
2.1.1	Creating a new VM . . . . .	4
2.1.2	Installing Ubuntu . . . . .	4
2.1.3	After installation . . . . .	5
2.1.4	Install necessary packages . . . . .	5
2.1.5	Editor . . . . .	6
2.1.6	Clone the given Git Repository . . . . .	6
2.1.7	Setting up the connection to the SensorTag debuggers . . . . .	6
2.2	Using the RaspberryPi . . . . .	7
2.2.1	Usage of the RaspberryPi . . . . .	7
2.3	Flashing the SensorTags . . . . .	9
2.4	Set up the 6LoWPAN network . . . . .	11
<b>3</b>	<b>Lab Tasks</b>	<b>13</b>
3.1	Task 1: Attack existing communication . . . . .	13
3.2	Task 2: . . . . .	15
3.2.1	Task 2.1: Cloning the SensorTags . . . . .	15
3.2.2	Task 2.2: Periodically send Sensor Values . . . . .	16
3.2.3	Task 2.3: Alternative Behavior of the cloned SensorTags . . . . .	17
3.3	Task 3: Secure the MQTT communication . . . . .	18
3.4	Task 4: Control the SensorTag running the robot.elf . . . . .	20
3.5	Task 5: CoAP - A different protocol . . . . .	22

<b>4</b>	<b>Troubleshooting</b>	<b>24</b>
<b>5</b>	<b>Additional information</b>	<b>25</b>
5.1	Setup of the RaspberryPi . . . . .	25
5.1.1	Installation of RaspberryPi OS . . . . .	25
5.1.2	Software installed on the RaspberryPi . . . . .	25
5.1.3	Cloning contiki-ng release 4.5 and building tunslip6 . . . . .	26
5.1.4	Setup the Mosquitto Broker . . . . .	26

# 1 Preliminary remarks

## 1.1 Academic integrity

As writing code will be part of this lab, we have to set some rules about academic integrity. You are **only allowed to use code that you have written yourself** (besides the provided in the repository). If you use existing code snippets, you must at least reference the source using a comment in code and mention it in presentation. If you are in doubt, please ask the tutor.

## 1.2 Documentation

### 1.2.1 Code

Make sure that your code is documented using **meaningful comments**. It is mandatory to **use the git repository** to hand in your code after the lab, so use git from the beginning, as it saves your code from accidental code changes or deletions and gives you the possibility to revert your code to previous versions. In addition, it is a chance for the lecturer to see your contributions. If you have no experience with using git, you can find a comprehensive help at the [GitLab LRZ](#), where you also can find a **New to Git and GitLab?** section.

### 1.2.2 Your Work - The lab report

Make sure to write a **lab report** starting from the beginning. The lab report has to be created by each student individually. Use the supplied `lab_report_template.md` in `/LabNotes`.

The **purpose of the report** is the following:

- Identify the individual tasks done by **each** team member
- Set the focus of the oral exam to the task each team member has performed.

So it is in your own interest to have a detailed report, you don't have to write things like "Added three lines in fileA...". Please don't write emails to the lecturer with things like "How detailed should the report be?". To answer the question for yourself, think about what information you would need to grade the lab.

## 1.3 Getting help with tasks and problems

If any problem occurs during the lab, like for example a broken setup, at first do **not write an email** to the lecturer but rather try to contact the tutor first. Also make sure to check out the **troubleshooting section** at the end of this guide. Finally, please don't come to the office of the lecturer without an appointment. The tutor will arrange a meeting if needed. This does not mean that the lecturer is not available for questions it's rather to avoid questions that can be easily solved during a tutor session.

## 2 Lab Setup

This section describes how to set up the lab environment and how to use the provided SensorTags.

### 2.1 Set up a Virtual Machine (VM) with all tools needed for the lab

Set up a VM running Ubuntu 20.04 with the settings according to the following instructions.

First step is the installation of VirtualBox. All necessary information can be found at the official VirtualBox [Website](#). Install the VirtualBox is needed.

**Please note:** It is needed to install the *VirtualBox Extension Pack* in order to perform the lab. If you choose to install it please be sure that you **only use it for the work in this lab**. Otherwise, be sure to comply with the given [license](#).

#### 2.1.1 Creating a new VM

The following settings should be selected:

- Choose an appropriate name
- Type: Linux
- Version: Ubuntu 64Bit
- 2048 MB Memory or more
- create hard disk → VDI → dynamic allocation → choose at least 20 GB. Through the dynamic allocation the storage space needed on the computer will be smaller than 20 GB, if not the total space is used during installation and VM usage.

After creating the VM change the following under "Settings":

- System → Processor → change number of processor cores at least 2
- USB → You don't need the VirtualBox Extension Pack so just use USB 1.1 is sufficient.

#### 2.1.2 Installing Ubuntu

1. Download the Ubuntu 20.04 ISO-File from the official Ubuntu [website](#).
2. When starting the VM the first time select the previously downloaded ISO-File.
3. In the Ubuntu installer select the following settings:
  - Language recommendation: English
  - Install Ubuntu
  - Detect keyboard layout
  - Minimal installation (to save hard disk space)
  - Erase disk and install Ubuntu

When selecting a username and password make sure to save them for later. The password is needed later on.

While testing a bug occurred using the German keyboard layout. Some symbols were at the wrong keyboard position while entering username and password. Later on the keyboard

worked fine. This would not be a problem, if the entered password in the installer would not be hidden or the existence of a view password function. To guarantee a correctly set password, you can type the password for example in the username field and copy it to the password fields.

Choose login automatically for easier VM access (of course this would be a security concern on a normal machine!).

### 2.1.3 After installation

After the installation has completed successfully select reboot. When asked to remove installation medium and press return, just press return. VirtualBox removes the installation medium automatically. You might want to change the keyboard layout under *Settings* → *Region and Language*.

First step after the successful installation is to get the system up-to-date. Therefore, open a terminal by pressing Ctrl+Alt+T. You can add the icon to the favorites by Right-Click on the icon.

In the terminal enter the following commands:

```
sudo apt update  
sudo apt upgrade
```

After the commands have finished, reboot the system. This command can be used in the terminal:

```
sudo reboot
```

After the reboot run the following command to install build tools and compilers like make and gcc:

```
sudo apt install build-essential
```

Now it is possible to install the VirtualBox Guest Additions. This Guest Additions offer a better integration of the VM by allowing shared clipboards, resizable display size and more. Under Devices select insert Guest Additions. In the now open window press run and enter the password.

After the installation finished, eject the Guest Addition by right-clicking on the CD-icon and reboot the VM.

### 2.1.4 Install necessary packages

For the IoT-Lab the following packages are needed. They can be installed like the build-essentials by the following command. Multiple installs can be combined.

```
sudo apt install git gcc-arm-none-eabi srecord
```

- git : version control Git
- gcc-arm-none-eabi : gcc compiler for ARM processors
- srecord : needed for building contiki-ng projects

### 2.1.5 Editor

On Ubuntu 20.04 at least the following editors are preinstalled.

- gedit: standard graphical editor
- nano: command line editor, easy to use
- vim: command line editor, powerful, opposite of easy to use

These editors offer basic text editor functionality. If you prefer more features, install an editor or IDE of your choice. One example for a good compromise between lightweight editor and IDE is [Visual Studio Code](#).

### 2.1.6 Clone the given Git Repository

Next step is cloning the given Git Repo which contains all necessary data for the IoT-Lab by using the following command:

```
git clone <https Link found in the online Git Repository>
```

The command will create a folder with the repository name at the current path, so choose the path before running the command.

### 2.1.7 Setting up the connection to the SensorTag debuggers

For the connection to the debuggers two files, which can be found in the Git repository, have to be placed in the `/etc/udev/rules.d`. In the folder containing the files run the following commands.

```
sudo cp 72-ti-Sensortag.rules /etc/udev/rules.d  
sudo cp 71-ti-permissions.rules /etc/udev/rules.d
```

## 2.2 Using the RaspberryPi

Open a terminal on the RaspberryPi either using an ssh connection or a monitor, keyboard and mouse.

The received RaspberryPi should work out-of-the-box. But if you are interested in the setup process is done, the setup is described in 5. This can also be helpful if the given RaspberryPi is not working correctly.

### 2.2.1 Usage of the RaspberryPi

The RaspberryPi can be used in two ways. It can be used with a monitor, mouse and keyboard. Or it can be used with ssh connected to the VM using an ethernet connection.

**Starting the RaspberryPi:** An unplugged RaspberryPi will start automatically when powered with the power adapter.

**Login credentials:** The **username** of the given RaspberryPi is **pi**. The **password** is **SEC\_iot2020+**.

**Shutting down the RaspberryPi** After usage always make sure to shut down the RaspberryPi correctly. When only being connected via a terminal one of these two commands will shut down the RaspberryPi:

```
sudo shutdown -h 0
sudo poweroff
```

The RaspberryPi can be unplugged when the green LED is off all the time. The red LED will still be on. You can also check the activity LEDs of the network plug.

**Usage with monitor** The RaspberryPi can be used like a normal linux system. Open a terminal and get started.

**Usage with ssh** Connect the ethernet cable to the RaspberryPi and a free ethernet port of your computer. It is not necessary to set up a connection with the host system (your computer).

Then it is necessary to set up the ethernet connection with the VM. Shutdown the VM and go to settings: Under Network activate a second network adapter. Now choose bridged mode and select the ethernet adapter, which the cable from the RaspberryPi is connected to. The Bridged Mode is necessary as the VM needs complete access to the ethernet adapter as if it was part of the VM.

In the VM we now can set up the connection. On the right upper corner go to Settings → Network. Uncheck “Make available to other users“. At IPv4 and IPv6 choose link-local only.

After the connection has been successfully setup you can check the connection using the following command:

```
ping raspberrypi.local
```

If you want to compare this to a working output of ping, you can ping localhost using `ping localhost`.

Now a non-graphical ssh connection in the terminal of the VM can be started.

```
ssh pi@raspberrypi.local
```

When asked if you want to continue connecting, enter yes. When asked for a password enter the previously mentioned password. If get bored about retyping your password you can read up on [ssh with public key authentication](#).

The terminal with the successful ssh connection now runs all entered commands at the RaspberryPi. It can be identified by the `pi@raspberrypi.local` at the beginning of each line.

The ssh connection can be closed using the following command entered in the terminal with the ssh connection.

```
exit
```

As later on Wireshark running on the RaspberryPi is needed an ssh connection allowing X11 forwarding is needed. Using X11 forwarding it is possible to start a graphical application on the RaspberryPi with its graphical interface opening at the VM.

```
ssh -X pi@raspberrypi.local
```

Now we can for example start a graphical terminal at the RaspberryPi using the command:

```
lxterminal
```

Here we can use multiple tabs or windows. All running on the RaspberryPi.



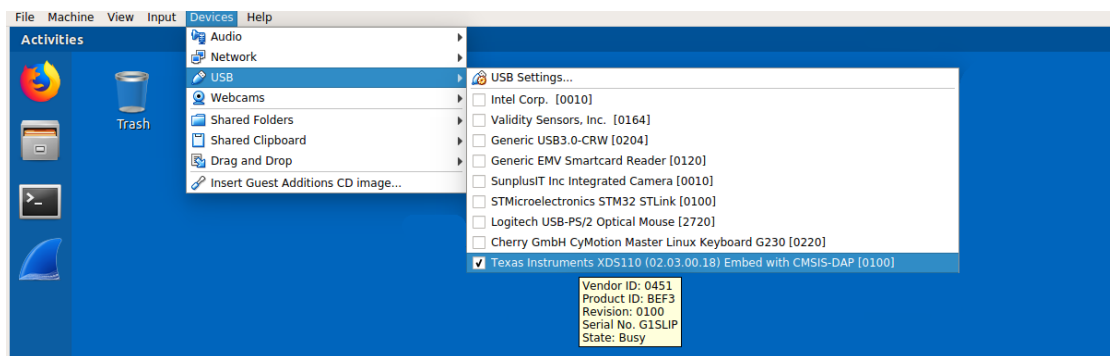
## 2.3 Flashing the SensorTags

In the `/binaries` folder you are given the four binaries `border-router.elf`, `sensniff.elf`, `mqtt-client-GXPUB.elf`, `mqtt-client-GXSUB.elf`. Here the X has to be substituted with your group number. Now flash the given binaries onto four different SensorTags using the debuggers. **You are only allowed to use the flash script provided in the git repository. As other tools may permanently lock and destroy the hardware.** Use the debugger with the label *SLIP* with the program `border-router.elf`, the debugger with the label *SNIFF* with the program `sensniff.elf`. These debuggers are already connected to SensorTags and **should not be disconnected. It is important to stick to this convention as otherwise the connection to the 6LowPAN Network is not working.**

**TI SensorTag** The documentation for the SensorTag can be found in `/Documents`. You are given the:

- Microcontroller datasheet
- Reference manual of the SensorTag
- Schematics
- Instruction for using GDB with the SensorTag

**Attaching Debugger with the SensorTag to the VM:** To attach a Debugger with the SensorTag to the VM plug it to a free USB-port. Then the Debugger should be selectable at `Devices` → `USB` at the menubar at the top of the running VM. You can identify the different debuggers through the serial number. In following picture the *G1SLIP* debugger is connected.

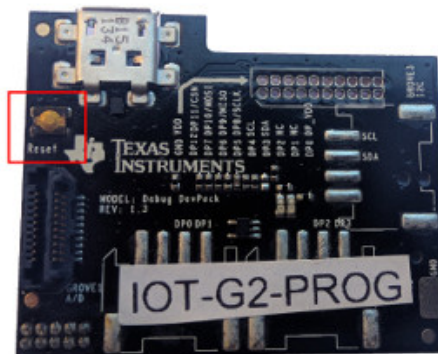


**Flashing a program onto the SensorTag:** To flash an `.elf` file onto the SensorTag, you have to connect it with a debugger to the VM and then use the programming script provided in `/program_device`

```
./flash_prog.sh <path to binary in .elf format>
```

**Additional information:**

- There is no chance to select between the connected SensorTags, so you have to **connect only the one you want to program**.
- The connectors of SensorTag and Debugger can easily be damaged. So do not separate them.
- After programming you have to **reset the SensorTag**. This can be done by pressing the reset button on the rear side of the debugger. As seen in the following picture. Or by simply unplugging the USB cable.



## 2.4 Set up the 6LoWPAN network

Set up the 6LoWPAN network as described in the following paragraphs.

**The Border-Router** The SensorTag connected to the Debugger SLIP should be already flashed with the border-router.elf. This SensorTag will set up the wireless SensorTag network.

To connect the 6LoWPAN network to the RaspberryPi connect the Debugger to one of the free USB-Ports of the RaspberryPi.

Using a terminal on the RaspberryPi (via ssh, ssh -X or a monitor) you can start the tunslip program. As long as the program is running the terminal is unusable. Make sure to not close the terminal window in which the tunslip is running, otherwise the connection is closed.

```
sudo ./tunslip -s /dev/SENSORTAG_SLIP fd00::1/64
```

👉 If there re multiple groups in the lab room please **only start a single border-router!**. We use the same 6LoWPAN channel for all groups and therefore multiple routers interfere with each other!

The following picture shows the successfull startup.

```
pi@raspberrypi:~$ sudo ./tunslip -s /dev/SENSORTAG_SLIP fd00::1/64
*****SLIP started on ``/dev/SENSORTAG_SLIP''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` mtu 1500 up
ifconfig tun0 add fd00::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST> mtu 1500
    inet 127.0.1.1 netmask 255.255.255.255 destination 127.0.1.1
    inet6 fe80::b8d9:f700:ec3e:ae67 prefixlen 64 scopeid 0x20<link>
    inet6 fd00::1 prefixlen 64 scopeid 0x0<global>
    inet6 fe80::1 prefixlen 64 scopeid 0x20<link>
    unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 500 (UNSPEC)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[INFO: BR      ] Waiting for prefix
*** Address:fd00::1 => fd00:0000:0000:0000
[INFO: BR      ] Waiting for prefix
[INFO: BR      ] Server IPv6 addresses:
[INFO: BR      ] fd00::212:4b00:afe:8b86
[INFO: BR      ] fe80::212:4b00:afe:8b86
```

If the border router is reachable can be checked using ping6:

```
ping6 <IPv6 address shown in tunsliip output beginning fd00::>
```

**View connected devices:** Call the IPv6 address from before in [] in the webbrowser on the RaspberryPi (using a monitor, the installed browser is *chromium-browser*) or use *wget* in the terminal. The brackets are needed, since we are trying to access a website through IPv6.

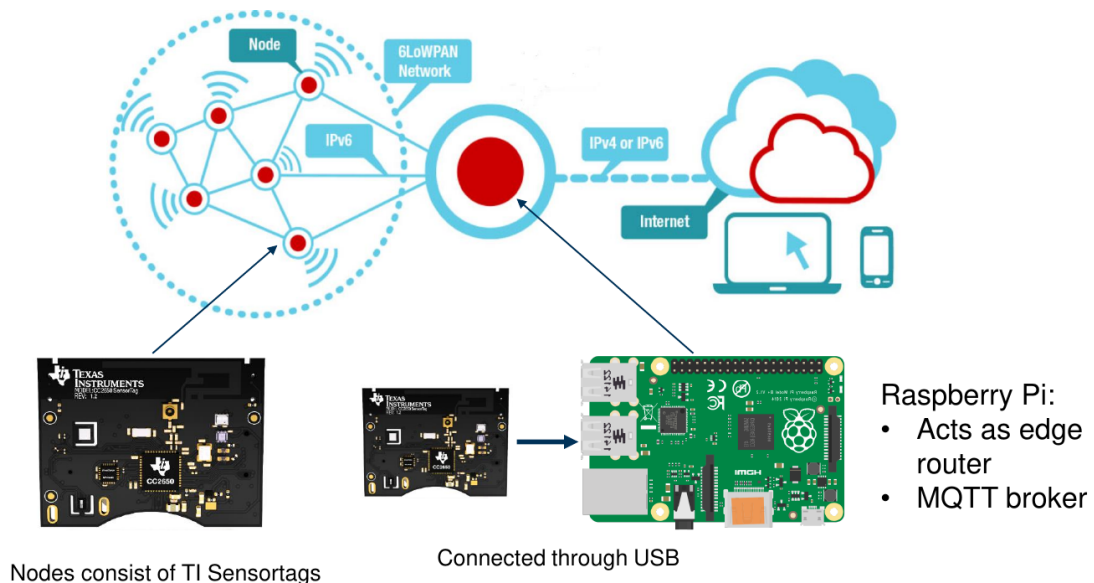
```
wget -6 "http://[<IPv6 address>]/"
```

This will store the information in a file, which name is printed.  
Now we can view the information with:

```
cat <FILENAME>
```

The connection to these connected devices can also be checked using *ping6* with the address beginning with *fd00::*.

### Overview of the infrastructure:

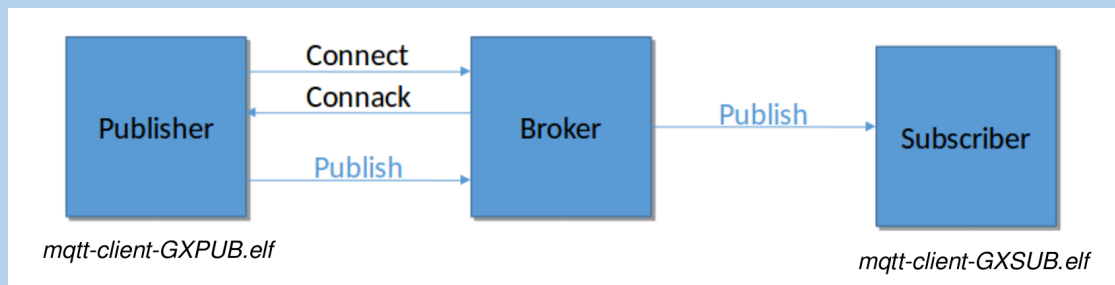


### 3 Lab Tasks

This section describes the tasks of the lab. You will find the given tasks written in the blue box. Please read the whole paragraph/section of the task first before starting to work on a given task. For the **deadlines** of the tasks check Moodle. Groups that do not deliver a working solution at the deadlines will be excluded from the lab.

#### 3.1 Task 1: Attack existing communication

You should have two SensorTags, which act as a MQTT publisher (`mqtt-client-GXPUB.elf`) and subscriber (`mqtt-client-GXSUB.elf`). Your task is to attack the unsecured communication by observing the wireless traffic using Wireshark. Refer to the second introduction to review the authentication with a MQTT broker and observe the following messages:



When you have obtained and saved your unique MQTT-credentials in a file, you are finished with this task. You also will need to know what events happen with the SensorTags and which message causes these events. You can test the credentials with the client programs of our MQTT broker called "mosquitto".

**LED on the SensorTag** The green LED on the SensorTag gives usefull information.

- very-fast-blinking: connecting to the border router.
- slower blinking: connecting to the MQTT broker.
- off-blinking-off: sending a mqtt message.

**Using Wireshark to capture packets** You are given a compiled program (`sensniff.elf`) for the SensorTag, that is able to capture packets on a chosen channel and transmit them through UART over USB to the RaspberryPi. You should already have flashed the SensorTag with the *SNIFF* label. Connect this SensorTag to the RaspberryPi now. In the folder `sensniff` in the home folder of pi there is a python script `sensniff.py`, which receives the packets and write them to a pipe `/tmp/sensniff`. This pipe can then be viewed with Wireshark.

The script can be started with this command in a terminal with an ssh connection:

```
python3 sensniff.py -d /dev/SENSORTAG_SNIFF
```

You have to make sure, that the script is sniffing on the **correct channel 26**. The channel can be changed by typing the required channel number followed by a confirmation through pressing enter.

In the home folder is a bash script `wireshark.sh`. It will start wireshark with the inclusion of the `/tmp/sensniff` pipe. The command executed by the bash script can be viewed by opening the file with a text editor. It can be started with:

```
./wireshark.sh
```

For a later task (do only when needed later): There is the possibility to show the decrypted DTLS messages therefore you have to set the correct pre-shared key in the Settings (in hex format): Edit → Preferences → Protocols → DTLS.

**MQTT-Broker** The broker that we are using is called **mosquitto** ([website](#)). The broker is already installed on the RaspberryPi. The authentication credentials are unique for each group. The documentation for the mosquitto broker can be found [here](#).

The broker comes with two client programs, that can publish and subscribe to certain topics on a running mosquitto broker. The clients are installed on the RaspberryPi. Since the broker is running on the RaspberryPi and binds all IP addresses you can use `localhost` as broker ip.

**Publish:** The `mosquitto_pub` client can be used to publish to a topic on the broker.

```
mosquitto_pub -h <broker ip> -u <username> -P <password> -t <topicname> -m <message>
```

**Subscribe:** The `mosquitto_sub` client can be used to subscribe to certain topics.

```
mosquitto_sub -h <broker ip> -u <username> -P <password> -t '#'
```

With `'-t'` you can specify the topic that you want to subscribe. The string `#` can be used to receive all messages.

## 3.2 Task 2:

### 3.2.1 Task 2.1: Cloning the SensorTags

After you successfully retrieved the authentication credentials and observed the MQTT-messages and events of the given SensorTags, you have to create your own publish- and subscribe-SensorTag. These SensorTags should exactly clone the given SensorTags. For example the SensorTags should send and receive the same MQTT-messages in the same intervall react to these MQTT-messages in the same way (e.g. hardware). It should be possible to use the given SensorTags and the cloned ones interchangeably. You can use the example project in `/contiki-ng/templates/mqtt-client-template` as a good starting point. **HINT:** You can also check the messages by using the MQTT-client installed on the RaspberryPi mentioned in 3.1. This is also helpful for checking, if your SensorTags are sending exactly the same messages.

**Contiki-NG** Throughout the lab each program builds upon a template, which is located at `/contiki-ng/templates/mqtt-client-template`. This template as well as compilation and debugging instructions are described in the next sections.

**Lab Template - mqtt-client-template** The template is a modified contiki-ng example project for the SensorTag. It contains an **MQTT client**, which publish and subscribe to a fixed broker IP. The template consists of these files:

- `project-conf.h`: Here you can set username, password, ...
- `mqtt-client.c`: Implementation of the MQTT client. The used MQTT implementation is located at `/contiki-ng/os/net/app-layer/mqtt`.
- `mqtt-client.h`

**Compiling Contiki-NG for the SensorTag** To compile any program for the SensorTag use the following command in the respective folder (unless otherwise specified).

```
make TARGET=cc26x0-cc13x0 BOARD=sensortag/cc2650
```

The program is then compiled as `<program>.elf` and can be found inside the `build/` folder. You can also fix the "TARGET" and "BOARD" variables inside the Makefile, so you can simply use a `make` for the compilation.

**Debugging** The debugger is creating a device when being connected to the RaspberryPi. The devices are `SENSORTAG_SNIFF`, `SENSORTAG_SLIP`, `SENSORTAG_PROG1` and `SENSORTAG_PROG2` corresponding to the debugger names. The devices can be found by using the following commands:

```
cd /dev  
ls
```

You can use `screen` or other programs to connect to the serial port of a certain device (e.g. `SENSORTAG_PROG1`):

```
screen /dev/SENSORTAG_PROG1 115200
```

Your debugging messages or data can be sent out through this serial port by calling `printf` ("This is my debug message") from all sources of your Contiki-NG program. Therefore, it is called `printf()`-debugging.

**The wiki:** For further information regarding Contiki-NG you can use their [online wiki](#). The **Programming Contiki-NG** part is especially interesting.


### 3.2.2 Task 2.2: Periodically send Sensor Values

Make a copy of your task 2.1 code.

Add necessary code to use the integrated sensors on the publisher SensorTag and periodically publish an additional MQTT message with the current sensor values of all (or most if there is a malfunctioning sensor) the available sensors under a topic of your choice.

The following tips could be helpful:

- The message could be a string of the values "Temperature: 32.1, humidity: <>, ...".
- You can find drivers (API) for the sensors in `arch/platform/cc26x0-cc13x0/sensortag`.
- Reading the different driver header and source files is helpful.
- You have to be able to demonstrate the change of behavior.
- If all or most sensors are not working, a likely cause is the wrong usage of the API.
- A topic, which is subscribable using a MQTT wildcard, can be helpful for the next task.

 As the hardware was used by several students before you, we have the problem that some or even all sensors of a Sensortag are broken. We made a test run with the respective tags and the Sensortag with the GX-PROG label should be working. Therefore, only **use the GX-PROG** sensortag as the **publisher**. If in doubt please ask the tutor, and we can issue a replacement if necessary.



### 3.2.3 Task 2.3: Alternative Behavior of the cloned SensorTags

Use your task 2.2 code.

Add necessary code to receive the published sensorvalue message and to change the behavior of the subscriber SensorTag based on one of the values of the sensors.

This is a chance for you to be creative.

👉 Remember there is a deadline for completing the tasks until now! Check on TUMOnline for the exact deadline date.

### 3.3 Task 3: Secure the MQTT communication

For this part you have to use a copy of the finished implementation of the task 2.3 and secure the communication between the publisher and the subscriber.

In a normal scenario this can be solved by using Transport Layer Security (TLS), which provides security for all TCP packets. Even though there are lightweight TLS implementations like [mbedTLS](#), the resources of the SensorTag are not sufficient to integrate this library. In our case, since the mbedTLS does not fit, we opt to **only** secure the payload (message data) of the MQTT publish messages. You should already be familiar with the way to send arbitrary payloads from the previous task. You can think of such a message in the following way:

Header	Payload
--------	---------

The security of the payload is provided by authenticated encryption with AES in the Galois/Counter Mode (GCM). This is a mode of operation for block ciphers (in our case AES), which provides in addition to *confidentiality* also *authenticity* and *integrity* of the data. Find the required documents and standards in the folder `/documents/AES`. You need to provide encryption and decryption for publisher and subscriber respectively.

To finish this task you have to:

1. Implement *AES-128*, which is used in AES-GCM:
  - (a) Develop your **own** implementation of the algorithm.
  - (b) Provide a performance evaluation of your implementation:
    - Memory consumption
    - Number of cycles
  - (c) Think of a way to handle the key schedule.
2. Implement the Galois/Counter Mode (GCM) with the AES-128 as a block cipher following the standards in `/documents/AES`.
  - (a) Develop your **own** implementation.
  - (b) Remember to implement the encryption (publisher) and decryption (subscriber).
3. Integrate your implementations into a working publisher/subscriber SensorTag. Use a copy of task 2.3 code as basis:
  - (a) Think of a way to create valid initialization vectors (IV):
    - Solve the synchronization of the IVs
    - Guarantee that IVs are unique
    - Justify your choice!
4. Use the integrated AES-128 hardware module and compare it against your implementation details (performance, ...). You find detail about the module in the reference manual in `/documents/Sensortag`.

**Take care:** In practice always use established and reviewed crypto-libraries. You should be really careful to use your own implementation because of:

- Side-channel vulnerabilities (see our other [lecture: Secure Implementation of Cryptographic Algorithms](#))
- Implementation errors that you did not recognize.
- There might be other restrictions on the usage of the algorithm that you did not follow.

**Adding own sources to the project** To add your own sources to the project you have to add them in the **Makefile**:

- In the Makefile add the name of your c files with:

```
PROJECT_SOURCEFILES +=
```

- Header files are automatically included when located in the top directory of a project (e.g. `mqtt-client-template/`).

### 3.4 Task 4: Control the SensorTag running the robot.elf

As you have implemented a working AES-GCM, you are now able to control the robot arm present in the lab room. In order to test the robot arm locally (outside of the lab room) we additionally provide a binary (*robot.elf*) that can be flashed to a SensorTag. It runs a mqtt-client similar to the one from *Task 2* and tries to replicate the behavior of the robot arm. Please perform the following steps:

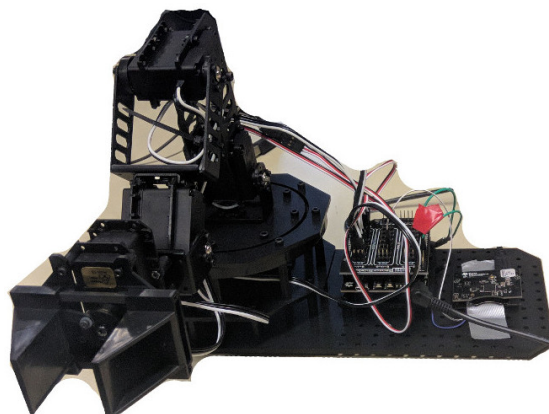
- Use the "*robot.elf*" or interact with the real robot arm in the lab room.
- An instruction for how to interact with the "*robot.elf*"/arm can be found in the file `robotControlX.txt`. It contains an AES-GCM encrypted ciphertext that is stored as a C-style string.
- After you successfully decrypted the file (this also verifies your AES-GCM implementation), please **store the resulting plaintext** in a file `robotControlSolution.txt` in your git repository. Also make sure to note in the file **whether the tag could be verified**.
- Now create a SensorTag that is periodically publishing a MQTT message according to the instructions.
- You will notice if the robot could be successfully activated through your message.

**Format of the message:**

TAG	IV	AAD	ENCRYPTED_DATA
-----	----	-----	----------------

For the length of the individual parts please refer the `robotControlX.txt` file.

**Robot arm:** When the Lab takes place locally at the university, there is a robot arm with an already connected SensorTag. This can be used to control the robot arm. The robot arm used is the RobotGeek Snapper Arduino Robotic Arm. It can be found online [here](#).



**Additional information:**

- Handle the robot arm like all other equipment **carefully**.
- If you power the robot arm for the first time, don't be surprised the arm starts to move very fast to get to its defined starting position.
- Unplug the robot arm after usage.

### 3.5 Task 5: CoAP - A different protocol

- First get familiar with the CoAP protocol. In the folder `/contiki-ng/templates/` you find a CoAP server template and a CoAP client template for the SensorTag, which work out-of-the-box. The CoAP server comes with example resources.
- Write different resources for the CoAP server:
  - A resource enabling or disabling a LED, when receiving *led=activate* or *led=deactivate* as POST request.
  - A resource responding with all sensor data, when receiving a GET request.
  - One periodic resource informing an observing client, when the value change of one sensor exceeds a threshold, by sending the sensor value.You can test your server implementation with the program `coap-client` installed on the RaspberryPi and the Wireshark output.
- Write a CoAP client. The client should fulfill following requirements:
  - Enables and disables the LED of the CoAP server recurrently.
  - Observes the periodic server resource.
  - React to observed changes by enabling the buzzer for a short time.
  - Instead of the template client, which consists of two different processes, it should have only one process.
- Secure the CoAP communication by using the integrated DTLS of contiki-ng. Verify if the DTLS encryption is working by using Wireshark.

**Communication with the COAP-Server:** On the RaspberryPi the program `coap-client` is installed (part of *libcoap*). It can be used with the following commands:

```
coap-client -m <mode either get/put/post> coap://<server IPv6 address>/<resource>
-e <can be used to include "text" as payload>
```

```
//The COAP protocol has a built-in mechanism to receive a listing of available re-
sources:
```

```
coap-client -m get coap://<IPv6 address>/.well-known/core
```

```
//For CoAP with DTLS:
```

```
coap-client -m <mode> coaps://<IPv6 address>/<resource> -u <username> -k <pre-
shared key>
```

**Securing the COAP communication:** Since COAP relies on UDP packets the communication can be secured with DTLS. This is already included in Contiki-ng and has to be activated during the compilation by supplying the following flags to make:

```
make MAKE_WITH_DTLS=1 ↵  
MAKE_COAP_DTLS_KEYSTORE= MAKE_COAP_DTLS_KEYSTORE_SIMPLE
```

Set your credentials in the *project-conf.h* file:

```
#define COAP_DTLS_PSK_DEFAULT_IDENTITY "user"  
#define COAP_DTLS_PSK_DEFAULT_KEY "password"
```

When requesting a resource by a CoAP client using *coaps://<IPv6 address>/<resource>* is necessary. If this is not working out-of-the-box, contact the tutor.

## 4 Troubleshooting

USB devices not showing up in VM menu	<ul style="list-style-type: none"><li>• Restart the computer running the VM.</li><li>• Check whether the extension packs are installed.</li><li>• Ask the tutor through the contact information on moodle.</li></ul>
tunslip6 not starting	<ul style="list-style-type: none"><li>• Check the USB connection to the RaspberryPi.</li><li>• Reset the SensorTag using the reset-button of the debugger.</li><li>• Are you using the SLIP-SensorTag?</li></ul>
SensorTags do not connect to the Border router (wireless)	<ul style="list-style-type: none"><li>• Has it stopped blinking very fast?</li><li>• Check if you flashed the correct SensorTag. If you connected multiple devices then there is no way to specify the one to program.</li></ul>
Debugging	<ul style="list-style-type: none"><li>• Use printf debugging to receive debugging messages.</li><li>• Often your problem is not caused by a very complicated issue, search for easy errors first.</li></ul>
Identify a SensorTag	<ul style="list-style-type: none"><li>• You can identify your SensorTag through its IP address. Check in the UART prints after a reset of the device and compare with the information provided by the router.</li></ul>



## 5 Additional information

### 5.1 Setup of the RaspberryPi

#### 5.1.1 Installation of RaspberryPi OS

The following steps are needed:

1. Flashing the SD-card with Raspberry Pi OS (32-bit) with desktop with help of the Raspberry Pi Imagers. It can be found at the official RaspberryPi [Website](#). If the SD card was used before, it can be helpful to erase all old partitions first using a tool like GParted.
2.
  - In the installer choose German. (The Setting is used for keyboard, wifi settings and more).
  - Select install updates.
  - Do not connect via Wifi but instead use a cable connection.
  - **Change the password to a secure password.**
3. After the installation finished set the language back to english. (Einstellungen -> Lokalisierung: English, GB, UTF-8)
4. Disable Wifi and Bluetooth as these are not needed and used.
5. Enable SSH in the config. (SSH uses the user password!)

#### 5.1.2 Software installed on the RaspberryPi

The following software needs to be installed:

- mosquitto : MQTT Broker
- mosquitto-clients : MQTT clients for testing
- coap-client: contains CoAP client with dtls functionality
- screen: usefull tool for debugging connected SensorTag
- wireshark: programm to monitor network traffic
- python3-serial: needed for sniffing script

**Wireshark** Wireshark is a tool to capture network traffic. In Combination with a sniffing-tool it will be used to capture 6LoWPAN traffic later on to attack the IoT-network.

Wireshark can be installed with:

```
sudo apt install wireshark
```

When installing a prompt will be shown in the terminal asking, if a non-superuser should be able to capture packets. Select yes.

After the installation the user account has to be added to the wireshark group to be able to make use of the non-superuser wireshark capturing.

```
sudo adduser $USER wireshark
```

To make the changes working reboot the VM. If the user is now part of the wireshark group can be checked by using the `groups` command.

The following settings under Edit → Preferences have to be changed.

- Under Protocols → 6LoWPAN: add `fd00::` to context 0.
- Under Protocols → IEEE802.15.4: select FCS-format TI CC24xx metadata.
- Under Protocols → MQTT: `show text as message` should be selected.

### coap-client

1. Install `autoconf`, `automake` and `libtool` using `apt`. These packages are needed for building the `coap-client`.
2. Clone the `libcoap` git repository, update submodules and checkout master. The repository can be found [here](#).
  - `git clone https://github.com/obgm/libcoap.git`
  - `git submodule update -init -recursive`
  - `git checkout master`
3. In the repository run the following commands:
  - `./autogen.sh`
  - `./configure --disable-tests --disable-documentation --enable-examples --with-tinydtls --disable-shared`
  - `make`
  - `sudo make install`

### 5.1.3 Cloning contiki-ng release 4.5 and building tunslip6

- Clone with `git clone https://github.com/contiki-ng/contiki-ng.git`
- Go in the repository folder and update the submodules  
`git submodule update --init --recursive`
- Set the release 4.5. `git checkout release/v4.5`
- In the folder `contiki-ng` → `tools` → `serial-io` run `make tunslip6`.
- Copy the executable to home folder with `cp tunslip6 /home/pi/tunslip`

### 5.1.4 Setup the Mosquitto Broker

The Mosquitto broker needs the following setup:

- Copy the given `authUser` file to `/etc/mosquitto`

```
sudo cp authUser /etc/mosquitto
```

- Copy the given local.conf file to /etc/mosquitto/conf.d

```
sudo cp local.conf /etc/mosquitto/conf.d
```