CRYPTOGRAPHIC SMARTMETERING

Deliverable DT3.2.1.

Version Tracking

Version	Description	Author	Date
0.1	Initial draft.	FM & SJ & WM & TG	10/02/2023
0.2	Improvements	FM & TG	27/02/2023
0.3	Review & improvements	AD	27/02/2023
0.4	Finalization & Conclusion	SJ & FM	28/02/2023
0.5	Further review in comment	TS	1/03/2023
0.6	Review part 2	AD & TG	2/03/2023

1 Introduction

Climate change and the associated energy transition are major challenges which also affect the way power is produced and how it is traded.
Central power production plants are supplemented e.g. by decentralized production of private photovoltaic systems.
To cope with the fluctuating production of renewable power systems, flexibility of consumers and sophisticated energy monitoring will be necessary.
The energy monitoring acts as an enabler in multiple ways: it visualizes the energy consumption of specific devices and how and when those devices are used. This opens the possibility to lower the total energy consumption or to shift the peak energy consumption time. Prosumers also have the option to match their energy demand with their energy production, which increases the consumption of in-house produced energy, puts less load on the power grid and lowers cost for bought-in (i.e. out-of-house produced) energy [1].

For these reasons energy monitoring plays a critical role in a modern and decentralized energy market. This project aims to deliver a platform not only for energy monitoring but also for using that information for trading energy in a safe way based on blockchain technology. As part of this project, we will also develop cryptographically secured smart metering devices. The security will ensure that measurements sent by the metering devices can not be manipulated during transmission to a server (a centralized system where the data is stored). Figure 1 shows the concept of the decentralized metering solution which will be further examined in the following chapters.

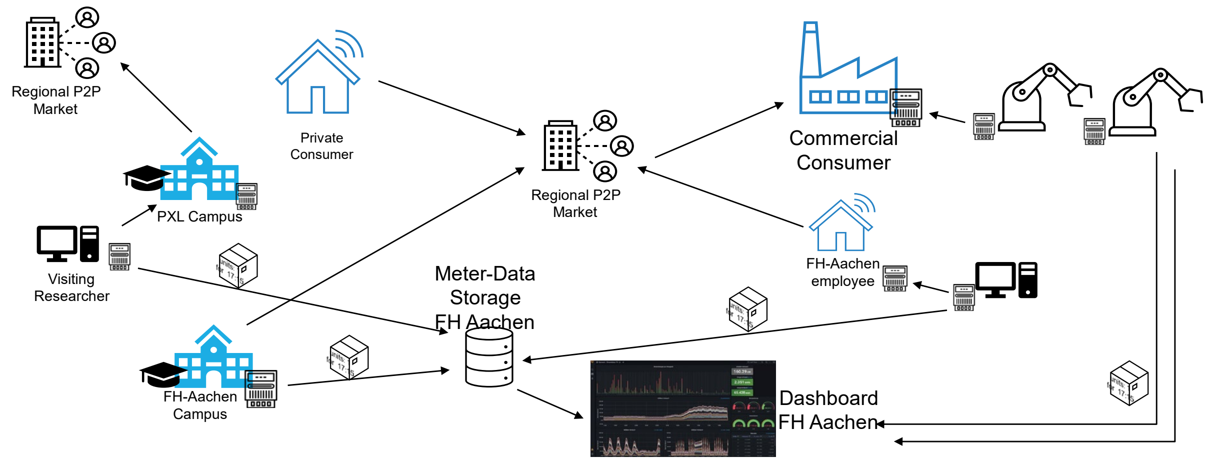


Figure 1. Concept for the decentralized campus metering solution

The system consists of multiple stakeholders e.g., the FH-Aachen Campus Jülich, the PXL campus, and some private households and companies where smart cryptometers have been installed. The measurements of these stakeholders are sent to a database on FH-Aachen's servers, which are visualized through a Grafana dashboard [2]. Besides the visualization of the data, another goal of this project is to show how a regional Peer-to-

Peer (P2P) Market could be implemented in a decentralized energy market concept. Therefore, an app will be developed which has the goal to trade energy locally between prosumers. Prosumers generating more power than they consume can sell the power to other prosumers, whose consumption is higher than their production. To follow the decentralized idea, blockchain technology will be used for trading energy. By using a blockchain, the data will be stored safely, transparently and will be distributed while being highly available. Trading will be done through a cryptocurrency. Blockchain technology with a cryptocurrency could make trading through an intermediary in form of a central authority, e.g. a grid operator, obsolete. The blockchain uses a consensus algorithm to keep a valid network state so all network participants can trust the blockchain [3].

As illustrated in Figure 1, this enables many small and decentralized energy markets consisting of prosumers which are a part of the European or global energy market. However, trading energy requires an energy monitoring of the production and consumption of power which will be explained in the following chapter.

2 Data Collection

To create a decentralized smart metering system, multiple components are needed. Those can be split from the macroscopic view into administrated servers and metering devices. This solution also enables to store data from other existing measurements from different data stores.

Figure 2 shows an overview of the created structure.
On the left, modern home metering devices which are installed as a plug or with clamps are shown. These devices can send the measured values through a message broker, MQTT (Message Queue Telemetry Transport)[1].
As a translator Telegraf is used to read the data from MQTT and write it into a time series database called InfluxDB.
The data stored in InfluxDB can be displayed in Grafana, a tool to create dashboards. There, the final resulting Dashboard can be adjusted as required.

[1] https://mosquitto.org/

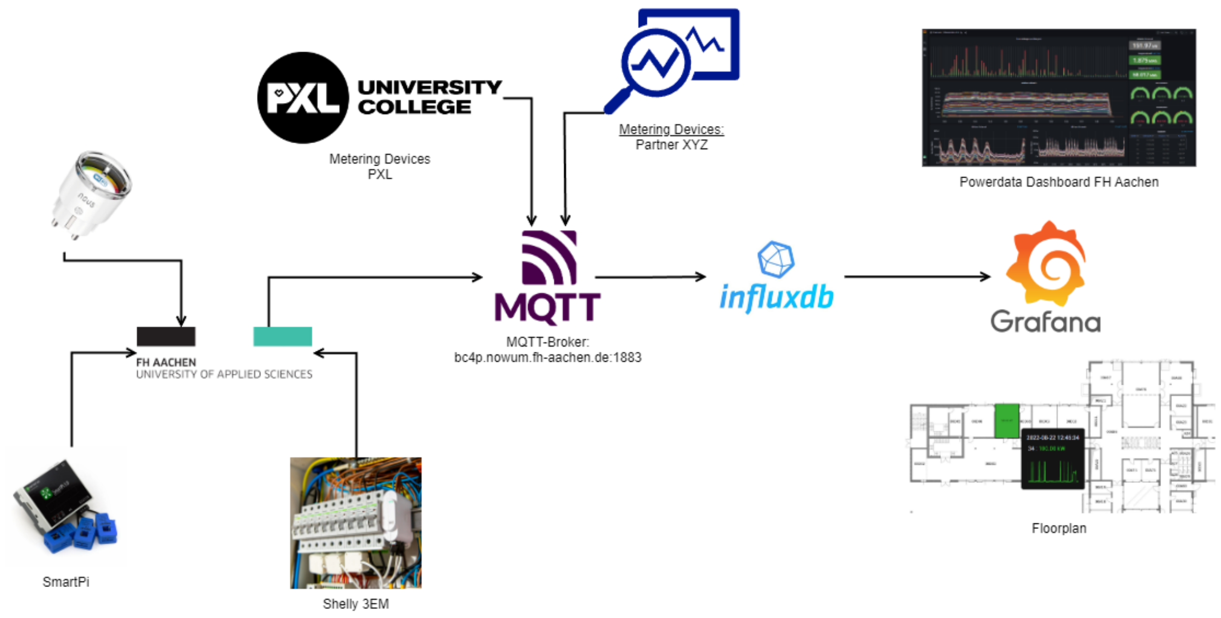

UNIVERSITY COLLEGE
Metering Devices PXL
Metering Devices: Partner XYZ
Powerdata Dashboard FH Aachen
MQTT
MQTT-Broker: bc4p.nowum.fh-aachen.de:1883
influxdb
Grafana
FH AACHEN UNIVERSITY OF APPLIED SCIENCES
Floorplan
SmartPi
Shelly 3EM

Figure 2. Overview of used components for the technical execution

2.1 Metering Devices

As metering devices, various available smartmeters were tested.
They can be classified into two categories:

- SmartPlugs,
- Non-intrusive energy meter devices.

In the following chapter these devices, advantages and disadvantages will be presented.

2.1.1 Smart Plugs

Multiple plugs are available from different manufacturers and can be bought on the internet. Manufacturers include the company Shelly, Atom and Nous, to name a few of the popular choices. We tested those devices and calibrated them to compare them with similar plugs. Other devices are mostly compatible too, as most smart home plugs are using an Espressif 8266 (ESP8266) microcontroller internally.

Figure 3 ESP8266 microcontroller

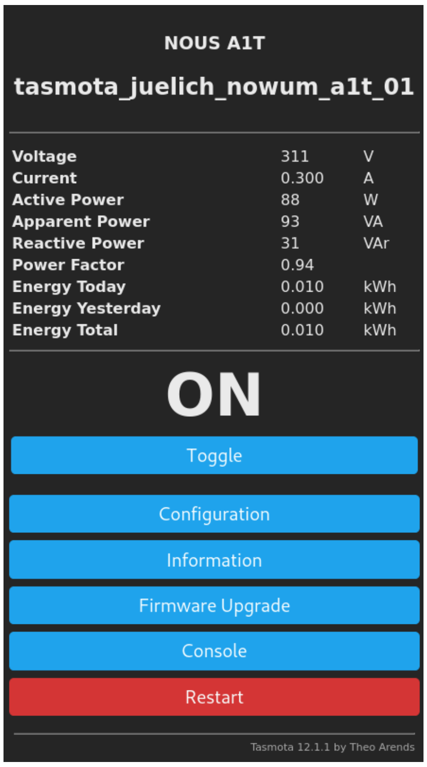
These devices can be fully flashed (installed, see chapter 2.1.4) with the open-source Tasmota firmware. This firmware can be adapted to make custom changes to the source code to extend the functionality. For example, meter value signatures can be implemented. This is something we did in this project.

The installation procedure differs among the devices. Some devices are pre-installed by the OEM (Original Equipment Manufacturer) with a different firmware and must be flashed with Tasmota first. Furthermore, some devices must be calibrated before use, which needs additional technical calibration equipment.

An overview of devices and their properties is shown in Table 1:

Device	Needs calibration	Installation	Accuracy	Max power
Nous A1	Yes	Preinstalled	●●	3680W
Shelly Plug S	No	OTA API	●●●	2500W
Athom Plug	No	preinstalled	●●●	3680W
Shelly Plug	No	OTA API	●●●	3500W
Nous A5T (3-plugs)	Yes	Preinstalled	●●	3680W
Gosund SP1	Yes	UART	●●	3650W
Gosund EP2	Yes	UART	●●	2500W
Tasmota OEM EU smart Plug Wifi	No	Preinstalled	●●	3680W
SONOFF S26R2 10A	No	UART	●●●	2200W
SONOFF S26R2 16A	No	UART	●●●	3680W
CloudFree EU Smart	Yes	Tasmota	●●	3680W
DeLock Power Sock	Yes	Tasmota	●●	3680W

Table 1. Properties of the different evaluated smart plug devices


NOUS A1T

tasmota_juelich_nowum_a1t_01

```
Voltage            311     V
Current            0.300   A
Active Power       88      W
Apparent Power     93      VA
Reactive Power     31      VAr
Power Factor       0.94
Energy Today       0.010   kWh
Energy Yesterday   0.000   kWh
Energy Total       0.010   kWh
```

Figure 4. Tasmota web interface

The installation and calibration process will be described in the following chapter.
To flash the Shelly Devices with Tasmota, the device needs a connection to the internet. If privacy is a concern, using a local webserver to receive device updates works too. If the device is reachable on the same network, the user of the device has to enter an update URL in the web browser, which can be copied from the public Repository[2].
Afterwards, an update to the latest Tasmota firmware can be performed. The calibration of this device will be kept during this process and works very well in general.

2.1.2 Calibration

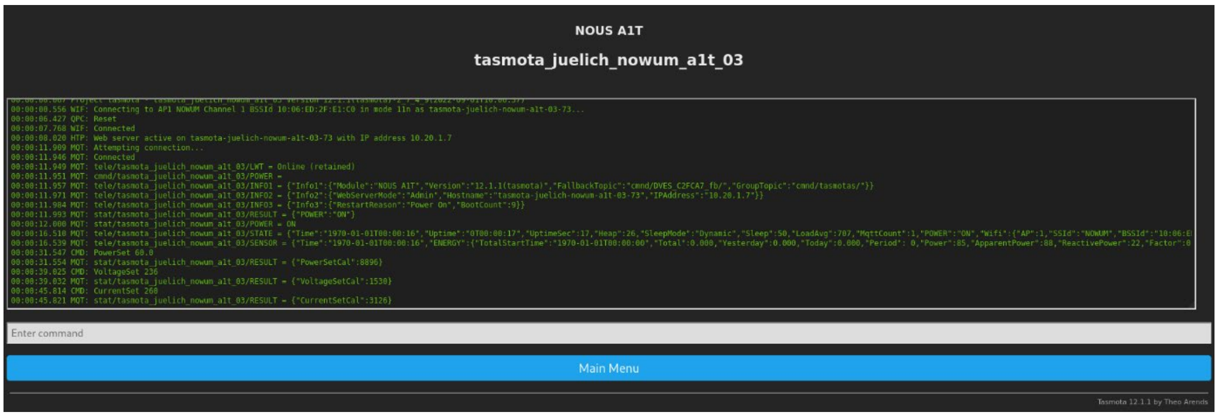
For other devices, especially the Nous with preinstalled Tasmota firmware, a calibration procedure is needed, which will be described in detail. By default, the voltage measured by these devices can differ enormously from the actual voltage. From our experience the range can vary between 95 V and 311 V as seen in figure 4. For

[2] https://github.com/arendst/mgos-to-tasmota

accurate measuring results it is recommended to calibrate power, voltage and current. Before calibration the Tasmota WiFi has to be configured. Therefore, one can enter the WiFi SSID and the password in the user interface. When doing this for multiple devices in a row it might be more convenient to do this process through the web API call which is: http://192.168.4.1/wi?s1=SSID&p1=WIFI-PASSWORD&save=

To calibrate the device, it is recommended to connect a pure ohmic consumer with a known power like a 60 W light bulb, which uses 100 % active power. While measuring the load with an already calibrated device (e.g. a multimeter) one must enter the correct values for the load by typing the following commands into the Tasmota web console:

PowerSet 60.0 (in Watt)
VoltageSet 235 (in Volt)
CurrentSet 260 (in mA - milliampere)



```
00:00:00.007 Project tasmota - tasmota_juelich_nowum_a1t_03 Version 12.1.1(tasmota)-2_7_4_9(2022-09-01T10:00:37)
00:00:00.556 WIF: Connecting to AP1 NOWUM Channel 1 BSSId 10:06:ED:2F:E1:C0 in mode 11n as tasmota-juelich-nowum-a1t-03-73...
00:00:06.427 QPC: Reset
00:00:07.768 WIF: Connected
00:00:08.020 HTP: Web server active on tasmota-juelich-nowum-a1t-03-73 with IP address 10.20.1.7
00:00:11.909 MQT: Attempting connection...
00:00:11.946 MQT: Connected
00:00:11.949 MQT: tele/tasmota_juelich_nowum_a1t_03/LWT = Online (retained)
00:00:11.951 MQT: cmnd/tasmota_juelich_nowum_a1t_03/POWER =
00:00:11.957 MQT: tele/tasmota_juelich_nowum_a1t_03/INFO1 = {"Info1":{"Module":"NOUS A1T","Version":"12.1.1(tasmota)","FallbackTopic":"cmnd/DVES_C2FCA7_fb/","GroupTopic":"cmnd/tasmotas/"}}
00:00:11.971 MQT: tele/tasmota_juelich_nowum_a1t_03/INFO2 = {"Info2":{"WebServerMode":"Admin","Hostname":"tasmota-juelich-nowum-a1t-03-73","IPAddress":"10.20.1.7"}}
00:00:11.984 MQT: tele/tasmota_juelich_nowum_a1t_03/INFO3 = {"Info3":{"RestartReason":"Power On","BootCount":9}}
00:00:11.993 MQT: stat/tasmota_juelich_nowum_a1t_03/RESULT = {"POWER":"ON"}
00:00:12.000 MQT: stat/tasmota_juelich_nowum_a1t_03/POWER = ON
00:00:16.510 MQT: tele/tasmota_juelich_nowum_a1t_03/STATE = {"Time":"1970-01-01T00:00:16","Uptime":"0T00:00:17","UptimeSec":17,"Heap":26,"SleepMode":"Dynamic","Sleep":50,"LoadAvg":707,"MqttCount":1,"POWER":"ON","Wifi":{"AP":1,"SSId":"NOWUM","BSSId":"10:06:E
00:00:16.539 MQT: tele/tasmota_juelich_nowum_a1t_03/SENSOR = {"Time":"1970-01-01T00:00:16","ENERGY":{"TotalStartTime":"1970-01-01T00:00:00","Total":0.000,"Yesterday":0.000,"Today":0.000,"Period": 0,"Power":85,"ApparentPower":88,"ReactivePower":22,"Factor":0
00:00:31.547 CMD: PowerSet 60.0
00:00:31.554 MQT: stat/tasmota_juelich_nowum_a1t_03/RESULT = {"PowerSetCal":8896}
00:00:39.025 CMD: VoltageSet 236
00:00:39.032 MQT: stat/tasmota_juelich_nowum_a1t_03/RESULT = {"VoltageSetCal":1530}
00:00:45.814 CMD: CurrentSet 260
00:00:45.821 MQT: stat/tasmota_juelich_nowum_a1t_03/RESULT = {"CurrentSetCal":3126}
```

Figure 5. Console View with Calibration Commands entered

Note that while the power and the voltage are entered in Watts and Volts, the value of the current must be entered in mA instead of Ampère.
If done correctly, the "Power Factor" should be 1.00 for a light bulb due to its pure ohmic property. This step is needed to get correct values for the resulting measurements of apparent, active and reactive power from the device. An exemplary view of the calibration console is shown in Figure . Note that in contrast to the calibration of the voltage, the calibration of the power factor phi can only happen with a load connected to the device. Figure 6 shows an exemplary test setup for the calibration process.
Details on the calibration can be found in the documentation of the project[3]

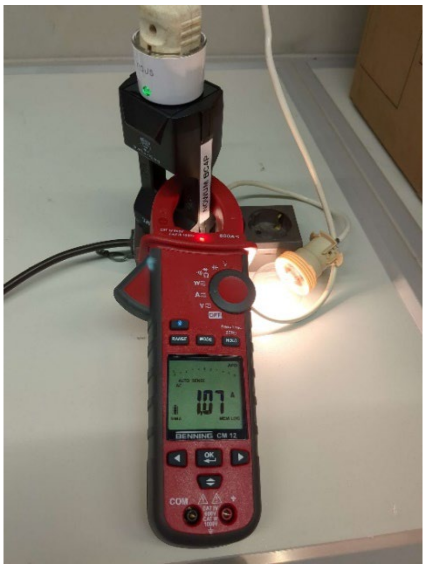
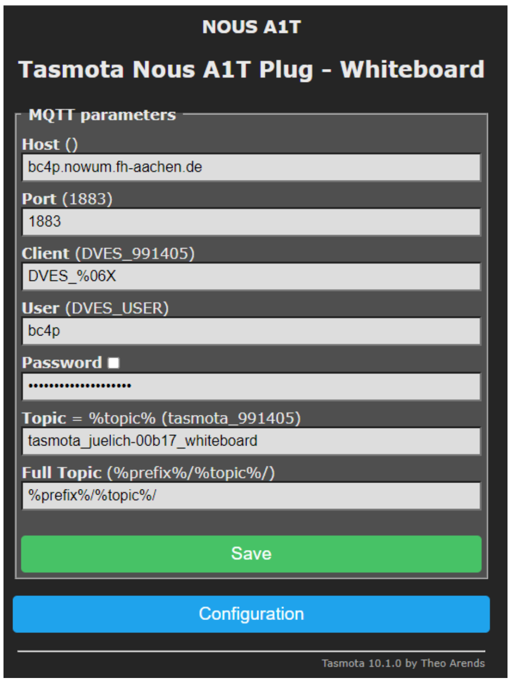


Figure 6. Image of the calibration process usage a power clamp and a known 60W light bulb

[3] https://tasmota.github.io/docs/Power-Monitoring-Calibration/

2.1.3 Connection to the BC4P Data Server with MQTT broker


NOUS A1T

Tasmota Nous A1T Plug - Whiteboard

MQTT parameters

Host (): bc4p.nowum.fh-aachen.de

Port (1883): 1883

Client (DVES_991405): DVES_%06X

User (DVES_USER): bc4p

Password ■: ••••••••••••••••••••

Topic = %topic% (tasmota_991405): tasmota_juelich-00b17_whiteboard

Full Topic (%prefix%/%topic%/): %prefix%/%topic%/

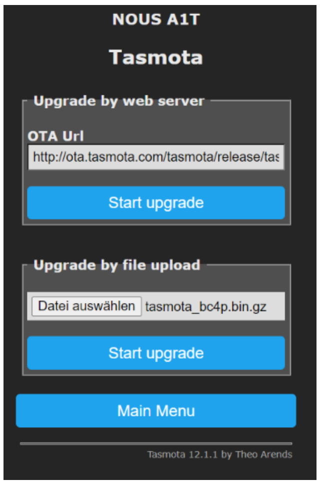
Figure 7. MQTT Settings on the Tasmota web interface

To be able to send data to a server one can set up MQTT in a compatible metering device. In Tasmota this can be done manually through the menu (Configuration → Configure MQTT):

For the BC4P project a monitoring server at FH-Aachen was installed, where the metering data can be sent to. Known device types will be automatically added to the monitoring system after MQTT setup. The host can be set to "bc4p.nowum.fh-aachen.de". The port remains the default one "1883". After entering the user "bc4p" and the password, one needs to give the device a Topic, which defines the name of the metering device within the system. This name will also be used as a display name for the device in the Grafana Dashboard. Thus, a representative name should be chosen for the Topic, such as "metertype_location-room_device". An example can be seen in Figure 7. The fields "Client" and "Full Topic" can stay at their default value. This step can also be done by using an API call like:
http://10.20.1.8/mq?mh=bc4p.nowum.fh-aachen.de&ml=1883&mc=DVES_%2506X&mu=MQTT_USERNAME&mp=MQTT_PASSWORD&mt=MQTT_NAME&mf=%25prefix%25%2F%25topic%25%2F&save=

2.1.4 Flashing custom firmware

Within the BC4P project a modification was made to the Tasmota software. It makes sure the data is being signed by the device before it is sent over MQTT. With this newly implemented functionality the data can be validated on the server before storing it in the database. The modifications will be presented in detail in Chapter . This custom firmware can be installed on the Tasmota devices in a process which is called "flashing". With Tasmota the devices can be flashed over the air (OTA), which means that the process can be done through Wi-Fi without physically connecting the device to a computer. From the main menu one simply can go to the "Firmware Upgrade" section.
Here the custom BC4P firmware, which can be downloaded from the BC4P GitHub page[4], can be selected from a file and the flash process can be started by clicking "Start upgrade". With this the data that gets sent through the network will be encrypted and signed.


NOUS A1T

Tasmota

Upgrade by web server

OTA Url: http://ota.tasmota.com/tasmota/release/tas

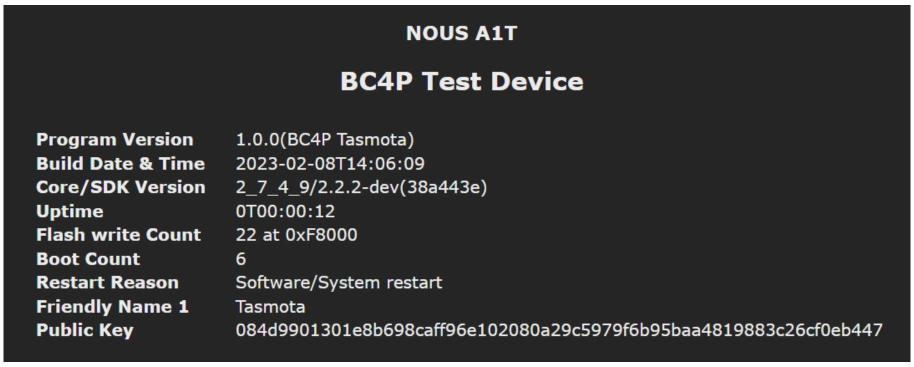
Upgrade by file upload

Datei auswählen: tasmota_bc4p.bin.gz

Figure 8. Flashing the Tasmota device OTA with the binary

[4] https://github.com/bc4p/Tasmota

To verify if the flashing process was successful, one can go to the "Information" section and check the "Program Version", which should include "BC4P" in the name. Also, the newly added field "Public Key" can be found here. With this key the server can verify that the data that it is receiving is valid and has not been tampered with by a third party. It needs to be shared to the server after setup. More details regarding encryption and signature using public and private keys will follow in Chapter 3.1. Additionally, the version information at the bottom of the menu should have changed to a version name including the tag "BC4P".

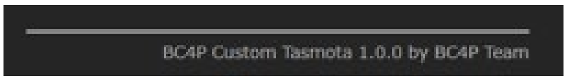


NOUS A1T

BC4P Test Device

Program Version	1.0.0(BC4P Tasmota)
Build Date & Time	2023-02-08T14:06:09
Core/SDK Version	2_7_4_9/2.2.2-dev(38a443e)
Uptime	0T00:00:12
Flash write Count	22 at 0xF8000
Boot Count	6
Restart Reason	Software/System restart
Friendly Name 1	Tasmota
Public Key	084d9901301e8b698caff96e102080a29c5979f6b95baa4819883c26cf0eb447

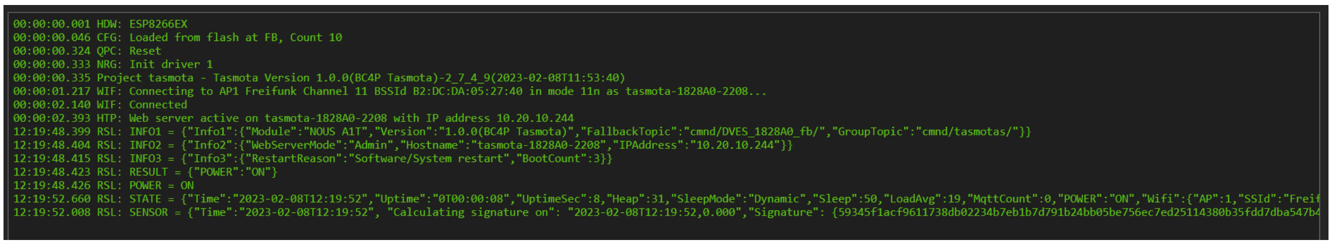
Figure 9 Location of the public key on the Tasmota web interface



BC4P Custom Tasmota 1.0.0 by BC4P Team

Figure 10: Custom Version text at bottom of menu

Figure 11 shows the signature inside a MQTT message that is being sent by one of your Smart-Crypto-Meters.



```
00:00:00.001 HDW: ESP8266EX
00:00:00.046 CFG: Loaded from flash at FB, Count 10
00:00:00.324 QPC: Reset
00:00:00.333 NRG: Init driver 1
00:00:00.335 Project tasmota - Tasmota Version 1.0.0(BC4P Tasmota)-2_7_4_9(2023-02-08T11:53:40)
00:00:01.217 WIF: Connecting to AP1 Freifunk Channel 11 BSSId B2:DC:DA:05:27:40 in mode 11n as tasmota-1828A0-2208...
00:00:02.140 WIF: Connected
00:00:02.393 HTP: Web server active on tasmota-1828A0-2208 with IP address 10.20.10.244
12:19:48.399 RSL: INFO1 = {"Info1":{"Module":"NOUS A1T","Version":"1.0.0(BC4P Tasmota)","FallbackTopic":"cmnd/DVES_1828A0_fb/","GroupTopic":"cmnd/tasmotas/"}}
12:19:48.404 RSL: INFO2 = {"Info2":{"WebServerMode":"Admin","Hostname":"tasmota-1828A0-2208","IPAddress":"10.20.10.244"}}
12:19:48.415 RSL: INFO3 = {"Info3":{"RestartReason":"Software/System restart","BootCount":3}}
12:19:48.423 RSL: RESULT = {"POWER":"ON"}
12:19:48.426 RSL: POWER = ON
12:19:52.660 RSL: STATE = {"Time":"2023-02-08T12:19:52","Uptime":"0T00:00:08","UptimeSec":8,"Heap":31,"SleepMode":"Dynamic","Sleep":50,"LoadAvg":19,"MqttCount":0,"POWER":"ON","Wifi":{"AP":1,"SSId":"Freif
12:19:52.008 RSL: SENSOR = {"Time":"2023-02-08T12:19:52", "Calculating signature on": "2023-02-08T12:19:52,0.000","Signature": {59345f1acf9611738db02234b7eb1b7d791b24bb05be756ec7ed25114380b35fdd7dba547b4
```

Figure 11: MQTT message including the signature

Besides the plug devices, there are the larger NIEM devices used to monitor whole distribution connections. We describe this in the following chapter.

2.1.5 NIEM Devices

The non-intrusive metering devices are quite rare, as the installed location is often not easily accessible and sometimes needs to be installed by an electrician because it is connected directly near the main fuses.
After the installation one has a complete overview of the total energy usage of all devices connected to the installation – for example the whole building.
Using non-intrusive metering methods has the advantage of installing without rewiring the installation or creating downtime.
The disadvantage on the other hand is, that the current is not measured directly but through the inductive measurement from the energy clamps.
The accuracy of the clamps is, for example from the Shelly 3EM given as ±1% (2 - 120 A), ±2% (1 - 2 A), ±5% (0 - 1 A), which is good enough for our application in this research project.

Figure 12. Enerserve SmartPi - non-intrusive metering

Available devices include the SmartPi 3 from Enerserve.eu, the Shelly 3EM, a self-built device using a microcontroller and a current clamp and radio-based meter using PowerRadar PAN10 and PAN12 sensors.

The options are described in Table 2:

Device	**Shelly 3EM**	**SmartPi 3.0**	**PowerRadar Sensors**	**DIY Smartmeter**
Needs calibration	No	Yes	No	Yes
Needs accurate voltage	Yes, per phase	No (but did not work without)	No	Fixed value
Cost	~ 100€	~ 300€	~ 140+ gateway	~ 50€
Custom Firmware	Tasmota support	Open-Source, Raspbian based	Proprietary	Arduino-based
WPA3 support	No	No (did break)	No	No
MQTT configuration	No configurable topic	Yes	MQTT not supported	Yes
Configuration	Easy	Advanced	Advanced	Advanced

Table 2. Comparison of different NIEM devices

In Table 2 one can see that the SmartPi 3.0 is said to support using a fixed voltage instead of measuring the voltage from its source, which makes this truly non-invasive, yet reduces the accuracy as the voltage changes are not measured. This is sufficient for most use cases without billing. On the other hand, the Shelly 3EM does also support Tasmota making it a good candidate for the use with a custom firmware for the cryptographically secure smart-metering.
Furthermore, it comes pre-calibrated and does not need the described calibration procedure before commissioning.

The setup routine for the Shelly 3EM is as follows:
When this device is correctly wired and connected to the grid, it opens its own Wi-Fi network with a SSID like Shellyem3-123456789. Now we can join this network with a laptop/smartphone, open the default web interface at http://192.168.33.1 in the browser and configure SSID and password of the network the Shelly shall operate in. Now the Shelly will show its future IP on the website, that should be written down before the device is rebooted. If it doesn't show its IP, a network scanner (e.g., Angry IP Scanner) can be used to find it. After the reboot the Shelly can be found under its new IP address in the newly connected Wi-Fi network. Shelly supports MQTT by default, and the MQTT broker settings can be entered from the menu. However, the device name for the MQTT communication cannot be changed from the menu like in Tasmota. The MQTT setup in the Shelly web interface can be done through the settings tab. For details see manufacturer's manual[5]

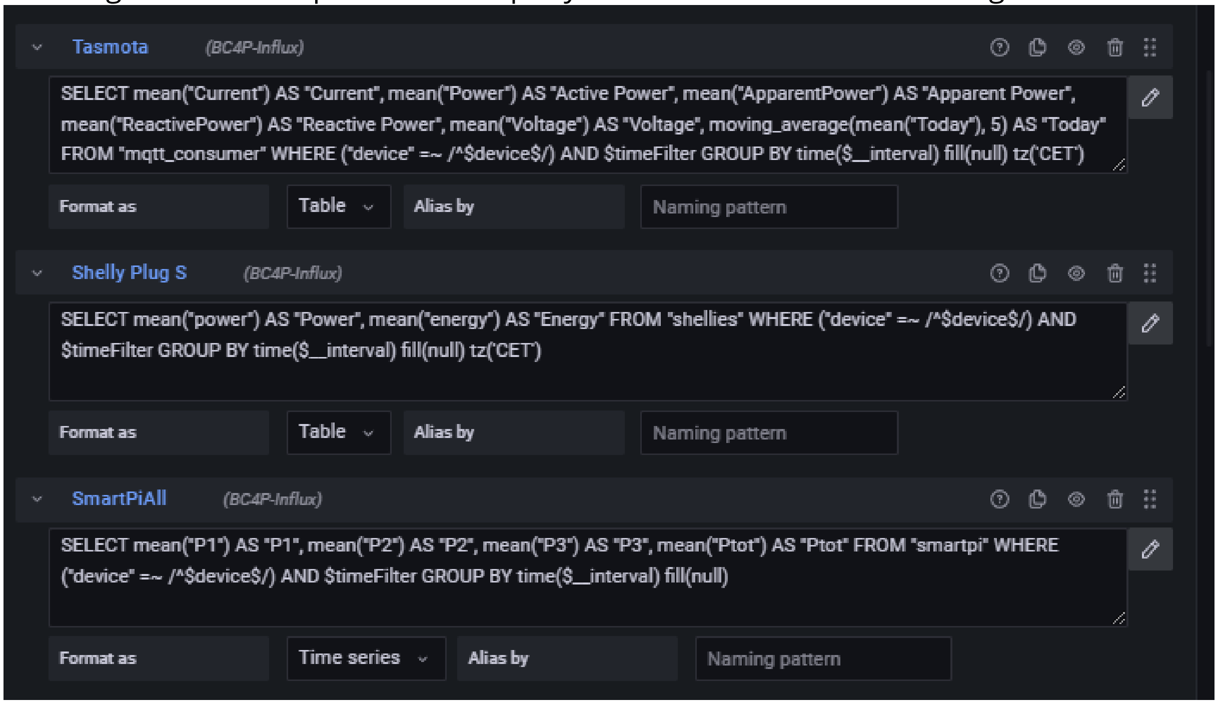
The SmartPi comes with a SD-Card which contains a pre-installed operating system based on Raspbian (Linux).
First the SmartPi needs to be connected to the power supply so the initial boot can be started. Then the device can be connected to a laptop via a LAN cable. Usually, the SmartPi should automatically set up a connection to the laptop using DHCP and the SmartPi should be reached through the static IP 169.254.3.10 via ethernet. However, sometimes this does not work, so after connecting the ethernet cable one can give the Laptop/PC a static IP from the address range of the SmartPi e.g. 169.254.3.20/24, leave the gateway empty.
Now the SmartPi can be accessed via SSH, linux-terminal: "ssh smartpi@169.254.3.10", the password is smart4pi. Under Windows the same can be done with the tool PuTTY.
The config mode of the device can be entered by typing "sudo raspi-config" into the console. From there the SSID and password of the WiFi that should be used can be set under "System Options" → "Wireless LAN".
After that one can reboot the SmartPi with the command "sudo reboot".
With the LAN connection still established open the SmartPi interface in the browser under http://169.254.3.10. From the menu with the three dashes (burger menu) one can go into the settings and log in with user "smartpi" and password "smart4pi. Here the MQTT settings can be adjusted using the credentials described earlier.
In the settings menu one can also change the name of the device and choose if the voltage should be a measured value or a fixed value, which can be set to a value (e.g. 230V). However, during our tests, the SmartPi was only able to measure accurately while

[5] https://shelly.cloud/documents/user_guide/shelly_3em.pdf

measuring the voltage actively. Using a fixed value for the voltage instead (e.g. measured by a multimeter beforehand) resulted in unreasonable inaccurate power measurement values (e.g. 1 W instead of 60 W actual power).
As the last step save your changes.
To find out the WLAN IP you have two options. One is to connect again via Ethernet/LAN and ssh to the SmartPi and use "ip a" to find the Wi-Fi IP of the SmartPi at its correct network adapter. The other is to scan the address space of the Wi-Fi on the laptop with the IP scanner of choice.

2.2 Data Storage and visualization

After setting up the Wi-Fi and MQTT connection the metering devices send data to the server. On the server, the data received from the different devices is translated and stored into an InfluxDB by Telegraf. Therefore, for different device types (e.g. Tasmota, Shelly, SmartPi) rules for the translation must be defined accordingly. This makes sure the data gets written into the database as expected.
To access the data from the InfluxDB, the Grafana Dashboard is used. Here multiple queries can be done for each panel. This means we can show data from different metering device types, which send their data using different formatting and naming of variables, in one diagram. An example of how to query data in Grafana is shown in Figure 13.



```
SELECT mean("Current") AS "Current", mean("Power") AS "Active Power", mean("ApparentPower") AS "Apparent Power", mean("ReactivePower") AS "Reactive Power", mean("Voltage") AS "Voltage", moving_average(mean("Today"), 5) AS "Today" FROM "mqtt_consumer" WHERE ("device" =~ /^$device$/) AND $timeFilter GROUP BY time($__interval) fill(null) tz('CET')
```

Format as: Table — Alias by: Naming pattern

```
SELECT mean("power") AS "Power", mean("energy") AS "Energy" FROM "shellies" WHERE ("device" =~ /^$device$/) AND $timeFilter GROUP BY time($__interval) fill(null) tz('CET')
```

Format as: Table — Alias by: Naming pattern

```
SELECT mean("P1") AS "P1", mean("P2") AS "P2", mean("P3") AS "P3", mean("Ptot") AS "Ptot" FROM "smartpi" WHERE ("device" =~ /^$device$/) AND $timeFilter GROUP BY time($__interval) fill(null)
```

Format as: Time series — Alias by: Naming pattern

Figure 13. Queries in Grafana

Results of the resulting Dashboard will be presented in Chapter 4.

3 Verification of the data

Wide areas of Germany are still using analog meters which are manually read by an official inspection every year.

The reason for this is a lack of secure implementation of smart-meter value transfer between the households and the operator.

So-called smart-meter-gateways are already implemented in other countries like the Netherlands, Belgium and Spain[6].

Here, the meter values are received by a central operator who takes care of the installation and operates the meter value aggregation.

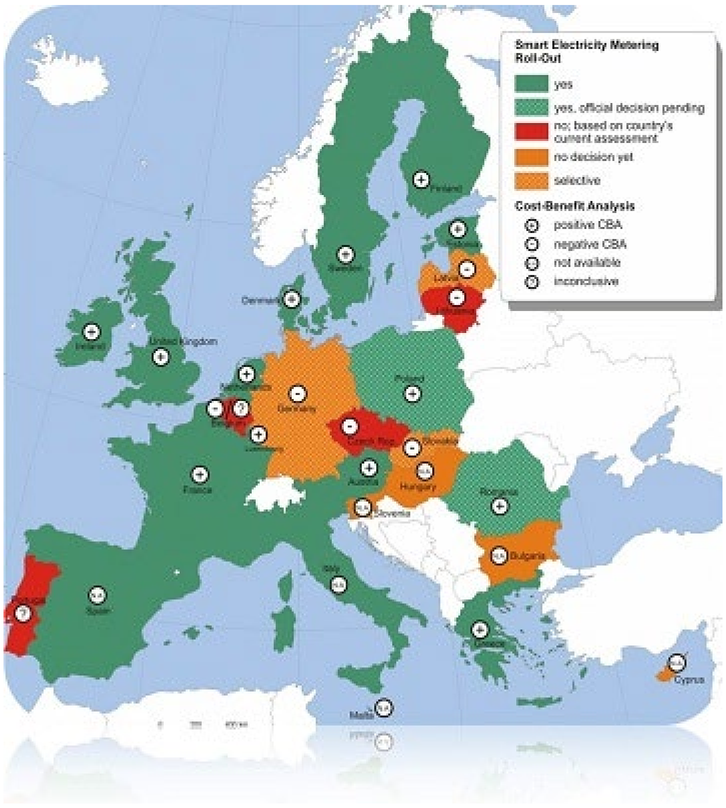


Figure 14. Overview of current smart meter rollout in the European union (Footnote 6)

In the setting of shared measurement data, a few things change.

As one meter can be used by different operators, a standardized solution is needed.

Having stored the data of multiple households creates a high responsibility for the data storage operator if the meter values are used for billing or cross-campus monitoring.

This responsibility could create an incentive to tamper with the devices meter values to influence the billing in a way it is beneficial for a man-in-the-middle or the operator itself. For example, one could reduce the amount of used energy reported to the electricity operator or increase the produced energy sent to the meter value operator.

Using smart meters, the transport needs to be secured in a way that no one can tamper with the measured values created by the smart meter.

The demand for this is very similar to the implementation of the German Eichrecht (§ 3 Nr. 24b MessEG[7]) which handles the correct billing of charging stations for electric cars.

3.1 Asymmetric Encryption/Signatures

Asymmetric encryption, also known as public key encryption, uses two keys: a public key and a private key. Data encrypted with the public key can only be decrypted with the

[6] https://ses.jrc.ec.europa.eu/smart-metering-deployment-european-union

[7] https://www.buzer.de/gesetz/10831/a183861.htm

associated private key. Conversely, data encrypted with the private key can only be decrypted with the public key. This enables secure transmissions because only the recipient who has the private key can read the data.
The recipient's public key is used to encrypt the data, while only the recipient's private key can decrypt the data again. The private key must remain secret, while the public key can be distributed without restrictions.

For example, if Alice wants to send Bob a message, she encrypts the message with Bob's public key. Bob can then decrypt and read the message with his private key. Since only Bob has the private key, the message can be securely protected from third parties.
This type of encryption is often used for secure transmissions on the Internet, such as online banking or the transmission of confidential information.

In the use case of cryptographically secured meter values, the use of signatures is the important aspect, which works just the other way around. Here, the sender's private key is used to create a digital signature for the data. The recipient can then use the sender's public key to verify the signature and ensure that the data has not been altered and originated from the specified sender.

If Alice sends Bob a message and generates a digital signature, Bob can be sure that the message is unaltered and came from Alice. Here, he can use Alice's public key to verify the signature.

When it comes to ensuring the integrity of data sent over the Internet, signatures are the standard way to do it. They guarantee that data remains unchanged and that it comes from the specified sender.

We evaluated a different solution to tackle the need for secure meter values using blockchain.
With blockchain, one would have to additionally serialize the incoming meter values to put them in a sorted queue, which creates a lot of overhead and latency with nearly no additional benefit.
Meter value events do not have the demand for strong consistency of their messages.
Using signatures allows us to verify a given meter value while only requiring its public key, making it obsolete to store the whole blockchain for verification.
An example of a large project where blockchain was used without big success for the sole need of verified PDF files can be seen in the German "Zeugnis-Blockchain", which aimed to improve validation of official education certificates. Yet it had many challenges to handle the complexity which was created by using the blockchain approach, while a more efficient solution using only the chain of trust and asymmetric signatures would be sufficient for this use case[8].

[8] https://www.heise.de/news/Schlechtes-Zeugnis-fuer-Zeugnisse-in-der-Blockchain-6370807.html

3.2 Implementation of the validation process

In order to add digital signatures to the MQTT-messages, and support verification of the data, a couple of changes have been made to the source code of Tasmota, which can be found on GitHub[9].
For this we created a fork of the main repository and added the functionality of verified signatures. This fork can be found on the projects GitHub page[10].

3.2.1 Adding the signature

The algorithm used for the generation of the digital signature is Ed25519, which is a state-of-the-art cryptographic methodology used for encryption and signing. For the implementation the Arduino Cryptography Library of author Rhys Wheaterly was used; source code can be found at arduinolibs[11].
To add this library to Tasmota, it has to be set as the lib_deps expression of platformio.ini, the project configuration file of platformio, which is the tool used to build the firmware. After this, the Ed255519.h file is included when needed. This file contains three functions that are used:

- generatePrivateKey(…), called once every time the device is hard reset,
- derivePublicKey(…), called every time the device is soft reset,
- sign(…), called every time a new message is created.

The addition of the signature is divided into four tasks:

- Loading or generating the private and public keys,
- Creating a message,
- Calculation of the signature,
- Adding the signature to the MQTT message.

At start-up the private and public keys have to be either loaded or generated, depending on whether it is the first time the device is configured or not. This means every device has one private key, which is kept when the device is soft reset (e.g. restart button in webUI or unplugged). But on a hard reset e.g., by power cycling 7 times, it loses all its configuration parameters including its keys, so they will have to be regenerated on the next boot.
To implement this, the setup() function of the tasmota.ino file was changed. This function is called once every time the device is restarted and when completed, the loop() function will be repeated.

[9] https://github.com/arendst/Tasmota
[10] https://github.com/bc4p/Tasmota
[11] https://github.com/rweather/arduinolibs
Documentation at: https://rweather.github.io/arduinolibs/index.html

A flow of the implementation is shown in Figure 15. When restarted, and after the initial unchanged setup of Tasmota, the public and private keys are read from the internal EEPROM (internal memory). They are stored at an offset of 512 bytes to prevent collision with the configuration and setting parameters of the device which are also stored in the EEPROM. After this a check is done to see if a keypair was already generated previously for the device. This is implemented by deriving the public key from the private key that is loaded and checking whether this derived key matches the loaded public key. If this is not the case, a new private key is generated, a new public key is derived and both are stored in the EEPROM. During runtime the keys are stored in global variables, so that they can be accessed everywhere.

Once the keys are loaded into the right variables, the calculations for the signature can start. This is done by making changes to the xdrv_03_energy.ino file and introducing a new custom file createSignature.ino which contains functions to calculate and print the signature. The creation of the message is straight forward: the timestamp is taken from the MQTT-message that is being prepared (TasmotaGlobal.mqtt_data) and the total energy consumption measured by the device is added to it, divided by a comma. For example: “2023-02-08T14:12:20,0.000”.

To calculate the signature, the sign()-function of the Ed25519 library is called. The creation of the message and calculation of the signature are done in the same custom function.

Finally, the signature is added to the MQTT-message by another custom function. Currently it is put between the timestamp and the energy values, as can be seen in the example of 3.2.4, although this can be changed easily by just calling the function at a different line. The printing happens by calling the ResponseAppend_P() function of Tasmota, which expects a formatted string and its arguments. The 32-byte array signature is printed byte by byte.

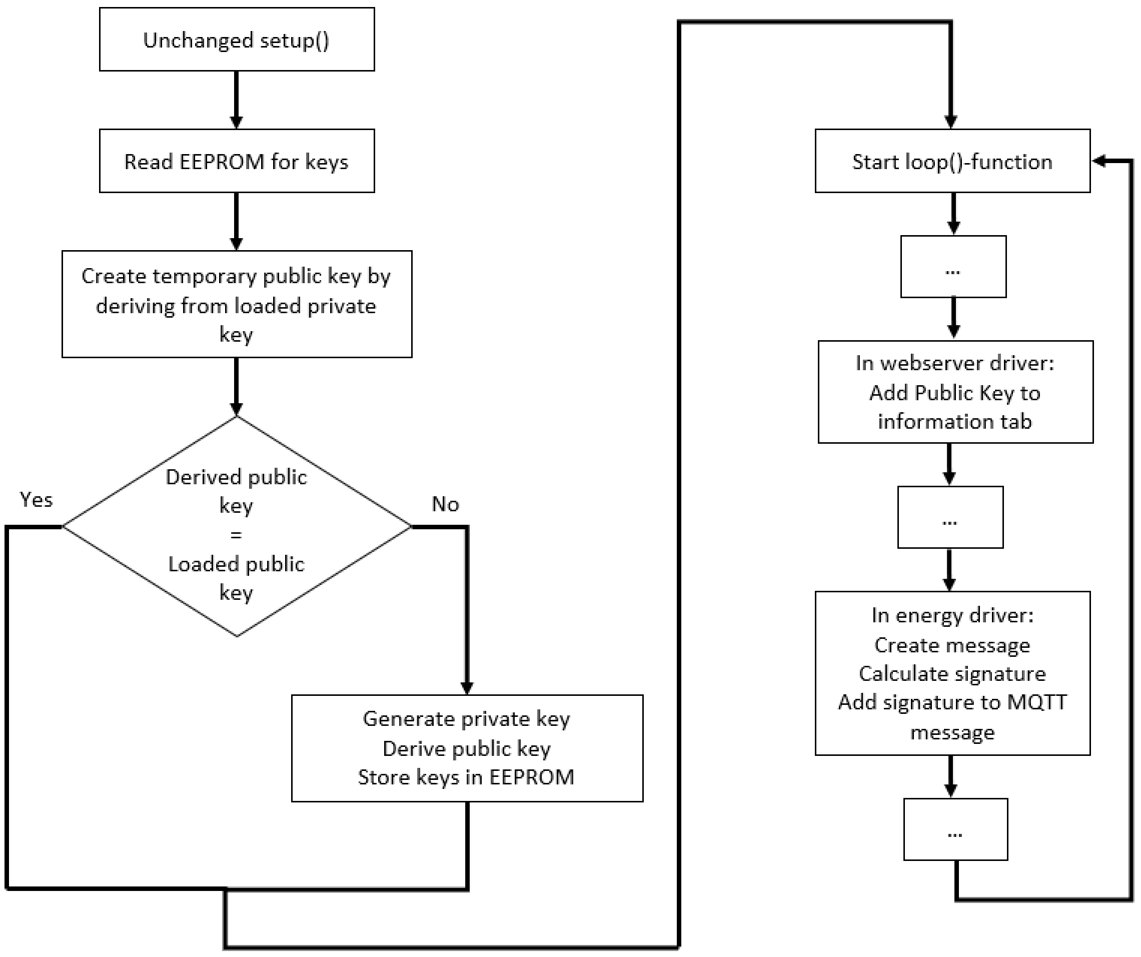
3.2.2 Other changes

Besides adding the signature itself, some other small changes were done. The most important one is implementing a way to communicate the public key of the device. It was opted to show this in the information tab of the webUI, which meant the xdrv_01_9_webserver.ino file had to be changed. The WSContentSend_P() function is called to add the public key in the already existing table; this is done byte by byte.

Another small change is the communication of the firmware name and version, shown in the bottom part of the webUI. This is done in the webserver file by editing the HTTP_END variable, which is a variable stored in the flash memory of the device. The clickable link has also been changed to redirect the user to the bc4p branch of Tasmota. The new name and version have been changed to “BC4P Custom Tasmota 1.0.0”. To change the version (1.0.0), the tasmota_version.h file had to be altered.

Finally, a modification of the firmware was implemented to make flashing of the custom Tasmota firmware easier. When the standard Tasmota is flashed on a device, there is not

enough space available to instantly flash new firmware. Instead, an intermediate step of flashing Tasmota-minimal had to be executed every time. To make the binary file smaller, some unnecessary functionality was excluded by creating a user_config_override.h file and excluding some macro's defined in the my_user_config.h file of Tasmota. In this way, the size was reduced from 628kB to 501kB.
An overview of the flow and the changes made to the Tasmota source code can be seen in Figure 15.



Unchanged setup()
Read EEPROM for keys
Create temporary public key by deriving from loaded private key
Derived public key = Loaded public key
Yes
No
Generate private key
Derive public key
Store keys in EEPROM
Start loop()-function
...
In webserver driver: Add Public Key to information tab
...
In energy driver: Create message Calculate signature Add signature to MQTT message
...

Figure 15. Flow and overview of changes made to Tasmota source code

3.2.3 Digital signature in charging stations

Our implementation of adding a digital signature to energy-monitoring devices can be compared with the software in charging stations for electrical cars, where digital signatures are also used. In Germany this is required by the Eichrecht.

The Eichrecht is a set of legal regulations that governs the use of measuring instruments in commercial transactions. The purpose of these regulations is to ensure that measuring instruments are accurate, reliable and traceable, and that they are used in a manner that is transparent and fair to consumers.
In addition to some regulations on the physical measuring instruments itself, the Eichrecht also requires charging stations to be equipped with digital signatures to verify

the integrity of the data generated by the measuring instruments. This is intended to prevent manipulation or tampering of the data.

Information about the exact implementation, was not found. However, the signature itself is calculated on the following information [4]:

- Measurement: e.g. meter start and stop value or difference
- Unit of measurement
- Time stamp
- Unique ID of charging system (e.g. EVSE-ID or Meter-ID)
- Identification of consumer (e.g. EMAID, Session-ID, UID, RFID)

Figure 16 visualizes where data is signed and encrypted when using charging stations for electrical vehicles and how the customer can verify the data.

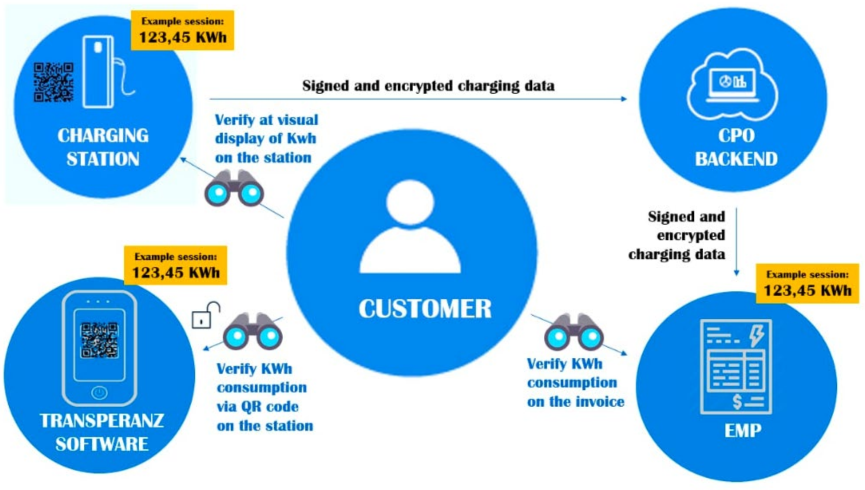

Example session:
123,45 KWh
CHARGING STATION
Signed and encrypted charging data
CPO BACKEND
Verify at visual display of Kwh on the station
CUSTOMER
Signed and encrypted charging data
Example session:
123,45 KWh
TRANSPERANZ SOFTWARE
Verify KWh consumption via QR code on the station
Verify KWh consumption on the invoice
Example session:
123,45 KWh
EMP

Figure 16. Signed data at charging stations

3.2.4 Validation server

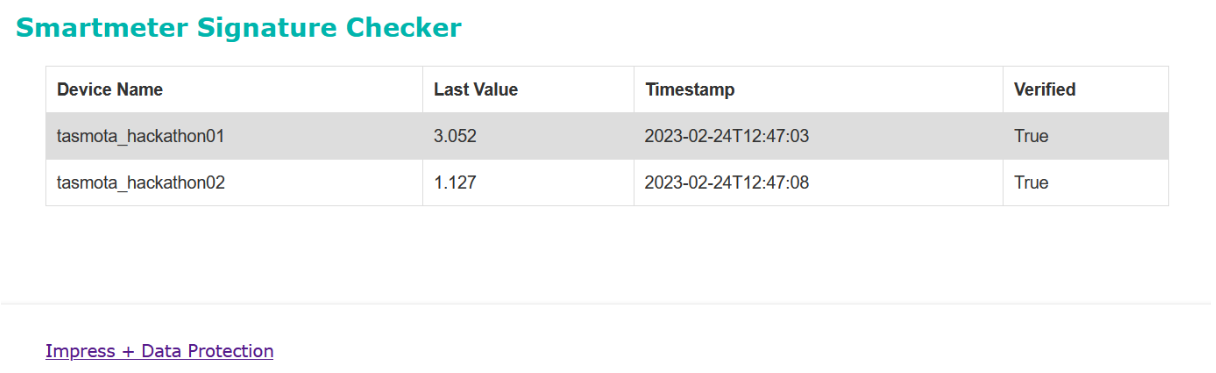
To verify that the messages sent by the Tasmota devices are not changed or manipulated in any way, the server receiving the MQTT messages from the Tasmota device needs to verify the signature.
Just to give a short recap how the message is signed by the Tasmota device: A hashing algorithm is applied to the time of the measurement and the total value which yields the signature as a string using the private key of the Tasmota device. The signature created is then appended to the MQTT message with the Key “Signature”. An example for the full MQTT Message which is received by the MQTT Broker looks like this:

```
tele/tasmota_hackathon01/SENSOR{"Time":"2023-02-08T13:54:26","Signature":
{9ae109b672774e2edc29bce5b626146af35fb482dedd73ba369b9351b508708a277aa568
b56b8a3b85105b19633506460afb82622fc20fb01fb10d35d0db3508},
"ENERGY":{"TotalStartTime":"2023-02-06T08:33:28","Total":1.581,"Yesterday":0.584,
"Today":0.498,"Period":1,"Power":51,"ApparentPower":78,"ReactivePower":59,
"Factor":0.65,"Voltage":234,"Current":0.333}}
```

To validate this message on the Server, the public key of the Tasmota device which sent the message is needed. The public key is then used to decrypt the Signature in the MQTT message. If the decryption yields exactly the encrypted message, the signature is verified. In this example, the decryption must return “2023-02-08T13:54:26,1.581” to be validated.

This process is implemented in the SmartCryptometer-Validator which can be found here[12].
To set up and run this program, the steps described on the GitHub page can be followed. It is important to put the device name of the Tasmota device as well as the public key of the device into the ***pub_keys.json*** file in the form of "DEVICE NAME":"PUBLIC KEY". By doing that, the program can match the name of the device with the topic of the MQTT messages to get the public key of that specific device. After that, the message is decrypted with the public key as described before.

The program continuously listens for MQTT messages and checks if the signature is valid. The result of the validation process for the last sent message of each device is displayed on a website which can be found under[13].
The figure below shows the linked website. All devices specified in the ***pub_keys.json*** file which sent messages to the MQTT Broker are listed here. For each device, the name, last value, timestamp and whether the message is verified or not, can be seen. Last value and timestamp are, again, the values used to verify the signature.



Smartmeter Signature Checker

Device Name	Last Value	Timestamp	Verified
tasmota_hackathon01	3.052	2023-02-24T12:47:03	True
tasmota_hackathon02	1.127	2023-02-24T12:47:08	True

Impress + Data Protection

Figure 17. Website of the Smartmeter Signature Checker

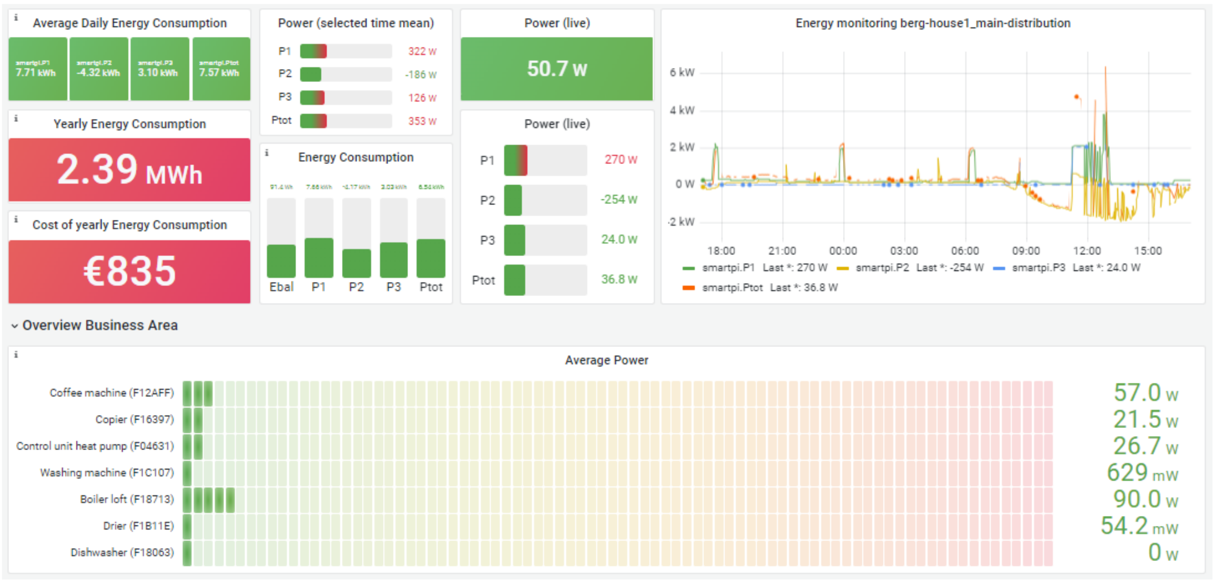
In this example, two devices were flashed with the forked Tasmota Firmware sending messages to the MQTT Broker. This can be extended with more devices compatible with Tasmota. Furthermore, this implementation gives the security needed for the transfer of energy monitoring data so that the data can be trusted and used for billing or trading, to give some examples.
Still, one has to trust that the measurement from the physical measuring sensor is correct and the device is not moved to a different power plug, where different measurements are possible.

[12] https://github.com/bc4p/SmartCryptometer-Validator
[13] https://bc4p.nowum.fh-aachen.de/validator

To conclude, this method can only validate the data against manipulation on the software side but not against a physical manipulation of the device or the circumstances where and how it measures the data.

4 Results and Visualizations

The data that is being sent to the server over MQTT and stored in the InfluxDB is visualized using Grafana. Several dashboards for the different pilot projects have been created, where the energy data can be monitored and analyzed. Data from different metering devices can be accessed from these dashboards. Besides information about the past energy consumption/production and resulting costs, one can also get live energy usage from the monitoring system and a detailed view on electrical devices that are connected to a Smartplug, as these sensors can send updated measures every 10 seconds.



Average Daily Energy Consumption
7.71 kWh
-4.32 kWh
3.10 kWh
7.57 kWh
Yearly Energy Consumption
2.39 MWh
Cost of yearly Energy Consumption
€835
Power (selected time mean)
P1 322 W
P2 -186 W
P3 126 W
Ptot 353 W
Energy Consumption
Ebal P1 P2 P3 Ptot
Power (live)
50.7 W
P1 270 W
P2 -254 W
P3 24.0 W
Ptot 36.8 W
Energy monitoring berg-house1_main-distribution
6 kW
4 kW
2 kW
0 W
-2 kW
18:00 21:00 00:00 03:00 06:00 09:00 12:00 15:00
smartpi.P1 Last *: 270 W
smartpi.P2 Last *: -254 W
smartpi.P3 Last *: 24.0 W
smartpi.Ptot Last *: 36.8 W
Overview Business Area
Average Power
Coffee machine (F12AFF) 57.0 W
Copier (F16397) 21.5 W
Control unit heat pump (F04631) 26.7 W
Washing machine (F1C107) 629 mW
Boiler loft (F18713) 90.0 W
Drier (F1B11E) 54.2 mW
Dishwasher (F18063) 0 W

Figure 18: Data Analysis in a Grafana Dashboard

The live data can be used to optimize energy consumption. Smart devices can be programmed to preferably run when a building with energy generation from e.g. solar panels produces more energy than the building requires.

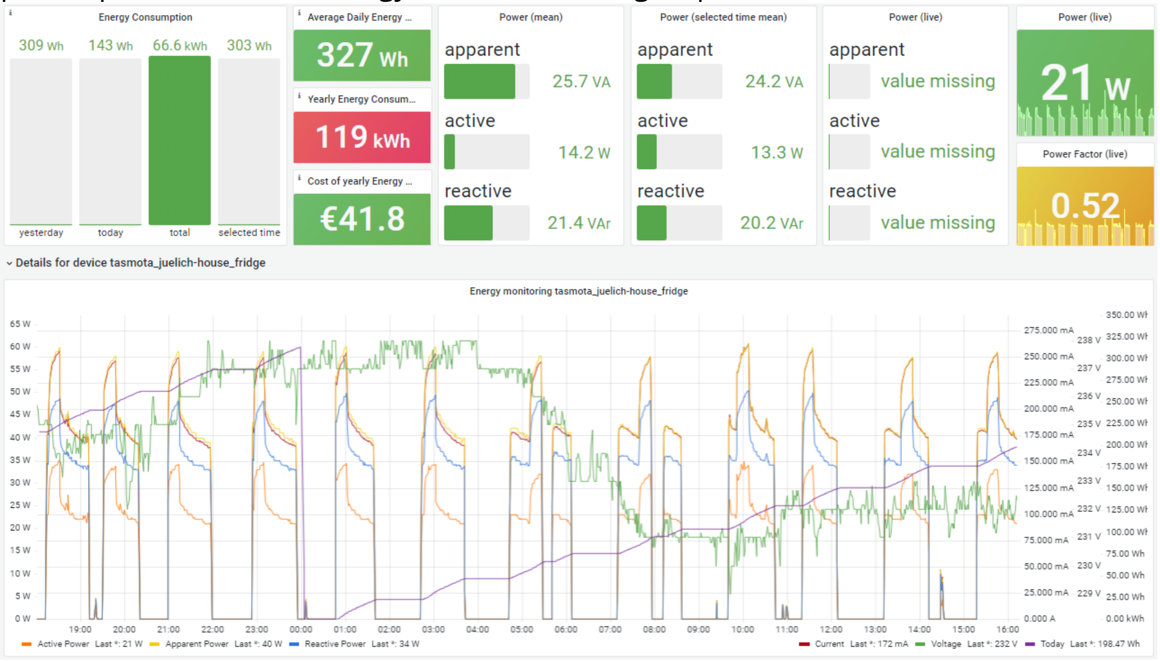


Figure 19 Analysis dashboard for economic parameters

If the building produces more energy than it can use, the excess energy can be sold. Looking at yearly cost and consumption of a device allows the user to compare the actual power usage of a device with its energy label, and predict the yearly cost of operating a tested device.

5 Conclusion & Further Developments

In the work package WPT3.2 we evaluated different metering devices, some which are usable as a plug while others can be used as non-intrusive energy measurement devices. Both categories can be used in conjunction with the open-source firmware Tasmota, which allowed us to showcase a proof of concept for cryptographically signed secure energy measurements.
The verification of the data can be seen on a dashboard which shows the current health of the meters as well as their latest values.
In a first step, the resulting data was visualized on various dashboards, to provide insights on energy usage.
Besides of the visualization of the data, it is possible to use the meter values which are stored in a consistent and efficient manner in a time series database. The database being used is InfluxDB. This database can be accessed publicly and used for further data analysis of the historical energy usage time series.
The data can be used for many more valuable use cases and to gain better insights. As many different devices and buildings are measured, there is data on a device level as well as aggregated data of the main electric power line of buildings measured. This data could be used for the training of machine learning models. The aggregated data can be used as input for a neural network which disaggregates the data into the device-specific energy consumption. This is especially valuable for buildings and companies without the need to measure each appliance individually because energy monitoring and insights could be generated by applying the algorithm to the aggregated time series metering measurements.

As machine learning-based prediction models require a lot of data to be efficiently used, we are also investigating to measure multiple different laboratory rooms to aggregate data on their typical energy usage mid-semester as well as in the typical non-lecture period (base load). The results will be provided in the pilot of Jülich.

Finally, cryptographic secure measurements are required for distributed metering solutions as needed in blockchain-based energy markets. This proof of concept therefore provides a foundation for the implementation and investigation of future distributed energy markets.

6 References

[1] Aretz, Astrid, Mark Bost, and Bernd Hirschl. "Prosumer für die Energiewende." Ökologisches Wirtschaften-Fachzeitschrift 2 (2016): 14-15.

[2] public grafana https://monitor.nowum.fh-aachen.de

[3] Klaus, Joachim, et al. "Blockchain in der Energiewirtschaft." Smart City–Made in Germany: Die Smart-City-Bewegung als Treiber einer gesellschaftlichen Transformation. Wiesbaden: Springer Fachmedien Wiesbaden, 2020. 329-337.

[4] Charging infrastructure
https://www.rvo.nl/sites/default/files/2019/04/German%20charging%20infrastructure%20regulations%20report%20march%202019_0.pdf or
https://www.linkedin.com/pulse/mess-und-eichrecht-germany-sahil-taksali

[5] P. A. Schirmer and I. Mporas, "Non-Intrusive Load Monitoring: A Review," in IEEE Transactions on Smart Grid, vol. 14, no. 1, pp. 769-784, Jan. 2023, doi: 10.1109/TSG.2022.3189598.