Project acronym: **IoTrust**

Project title: Secure trust bootstrapping and peer-to-peer network connections in

the Internet of Things

Third Party: DW/ODINS

Logo of partners

|  |  |
| --- | --- |
| Digital Worx GmbH (DW) - Germany |  |
| Odin Solutions SL (ODINS) - Spain |  |

Deliverable D.1

IoTrust Architecture Design

|  |  |
| --- | --- |
| **Deliverables leader:** | **Digital Worx GmbH, Germany** |
| **Authors:** | Rohit Bohara, Jesus Sanchez-Gomez |
| **Due date:** | 31-01-2021 |
| **Actual submission date:** | 15-01-2021 |
| **Dissemination level:** | Public / confidential |

Abstract: It is a deliverable document designed and developed under the IoTrust project. It gives extensive information about the IoTrust architecture. This document will serve as a reference document for the future deliverables of the IoTrust.

**Disclaimer**

The sole responsibility for the content of this publication lies with the authors. It does not necessarily reflect the opinion of the European Commission. The European Commission is not responsible for any use that may be made of the information contained therein.

**Copyright**

This document may not be copied, reproduced, or modified in whole or in part for any purpose without written permission from the NGI Consortium. In addition, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced.

All rights reserved.

This document may change without notice.

**Table of contents**

[1 Introduction 1](#_Toc26358819)

[2 Activities carried out to complete the deliverable 1](#_Toc26358820)

[3 Technical description 1](#_Toc26358821)

[4 Conclusions and next steps 1](#_Toc26358822)

[Appendix 2](#_Toc26358823)

Nb: The deliverable structure below is only provided for guidance and you may adapt in a free form manner the structure to fit your needs.

# Introduction

The deliverable **D.1 IoTtrust Architecture Design** fulfils the objective **O1** which aims to design a human-centric and open IoTrust solution to increase the use trust and application of secure IoT networks in worldwide sectors like Smart Cities, Industry 4.0 etc. The deliverable D.1 is the output of the task **T.1 IoTrust Architecture Design**. The task T.1 was completed in the duration of month M1 to M6. The DW was the leader of the task. The milestone **MS2 Enhanced final version of IoTrust architecture** was achieved by D.1. The milestone MS2 is the advanced version of the MS1.

# Activities carried out to complete the deliverable

The user-centric requirement analysis was performed in the task T.1 to deliver deliverable D.1. It was an iterative process in which requirements of end users and other stockholders such as internet developers were taken in to consideration in designing the IoTrust architecture.

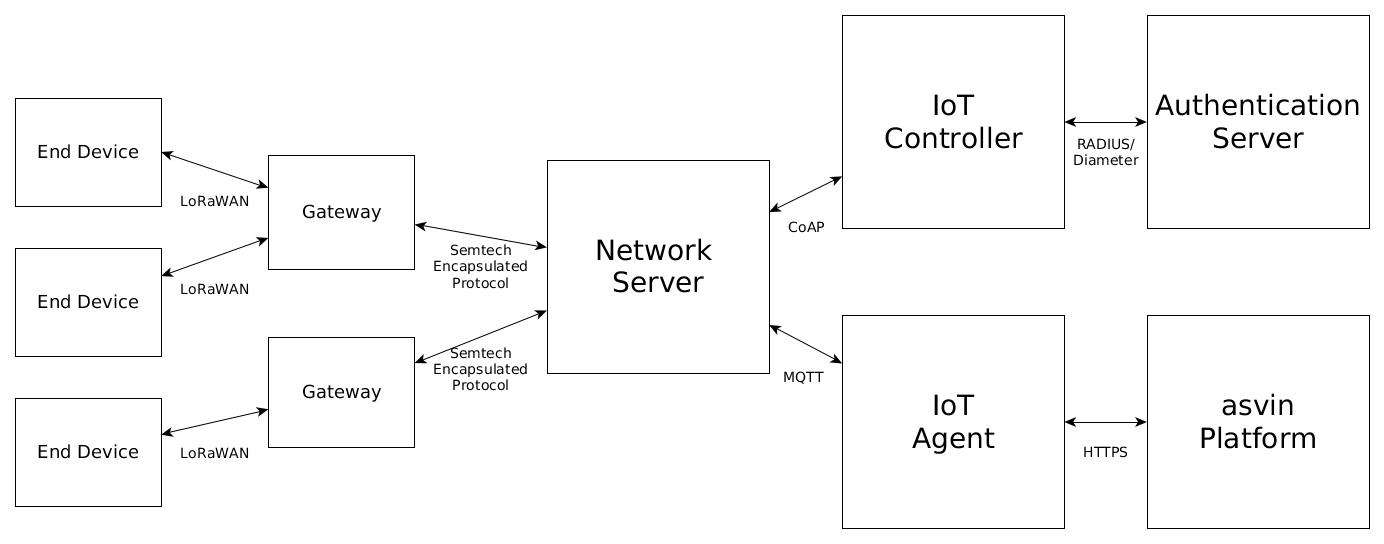
The task T.1 was performed based on Agile SCRUM[[1]](#endnote-1) methodology. Each SCRUM sprint cycle was of 2 weeks. At the start of each sprint cycle requirements were gathered from end users and patterners. These requirements were analysed and an IoTrust architecture draft was designed and developed based on them. At the end of the cycle, this draft was verified and validated against the requirements. This process was performed iteratively throughout the lifecycle of the task T.1.

In a project like this, where a final product is shaped according to the end user and external stakeholders' requirements, some unforeseen issues might arise in the areas such as requirement gathering, changing and unclear requirements, functional requirements verification and validation criterion etc. These problems were identified and solved using the iterative SCRUM cycle before they could occur and hinder the project. The requirement gathering and analysis were continuous process like the designing the architecture.

There were some unforeseen technical issues also addressed and fixed in the task T.1. The project is going to employ LoRaWAN[[2]](#endnote-2) protocol to send data packets between a LoRa[[3]](#endnote-3) node and gateway using radio communication in the 868 MHz ISM band. There are a large number of development boards available with different LoRa modems such as SX1276/77/78/79[[4]](#endnote-4) from [Semtech](https://www.semtech.com/), RFM95/96/97[[5]](#endnote-5) from [HopeRF](https://www.hoperf.com/), RN2483[[6]](#endnote-6) from [Microchip](https://www.microchip.com/) etc. We identified early enough that some libraries used for SX127X chips can send a packet of maximum 51 bytes which is not desirable for this project. It would have been a blocker in the project if we had not identified it at the start of the project.

# Technical description

The core aim of the deliverable D.1 was to prepare an advanced IoTrust architecture design with an iterative process. There are many aspects to the architecture design. We have analysed and prepared it with the details of hardware, software stack, communication protocols, DevOps, user interface, customer experience, API end points etc. This architecture design will serve as a reference for further deliverables. Although the core attributes of the architecture will be the same, there might be some minor changes as we reach to the next milestones. The Figure 1 [fig ref] illustrates the overall IoTrust architecture.



**Figure 1: The IoTrust Architecture**

The architecture components are described as follows.

## End-Device

It is a small form-factor hardware which sits on the edge of an IoT network. It consists of microcontroller, memory, input/output peripherals, communication protocol etc. These end-devices are put in to work for a specialized task. These end-devices are typically installed in hard to reach locations or in adverse condition, and are meant to work autonomously without human supervision during months or even years. For this purpose, these devices are commonly operated using a battery and do not include user interfaces such as keypads or displays. In some cases, they’re permanently installed in hazardous locations, sustaining extreme conditions, and require rugged cases that prevent dust and water penetration. Due to their small form-factor and lower power consumption, these end-devices are relegated to very specific monitoring and actuation tasks, with a simple operation logic that normally relies communications with a centralized cloud infrastructure.

In the IoTrust architecture, an end-device will be used to collect, format and send sensor data to a server. It is paramount to authenticate an end-device before it connects to the server using a critical network. Because if the end-device is compromised than it opens the flood gate to the critical network infrastructure. The authentication, authorization and key management tasks will be performed by a secure bootstrapping protocol, peer to peer and distributed ledger technologies.

The Smart Everything (SME) Lion[[7]](#endnote-7) [[8]](#endnote-8) development board will be employed as an end-device for the IoTrust project. It is designed and developed by Arrow[[9]](#footnote-1). It is packed with Atmel SAMD21[[10]](#endnote-9) microcontroller based on the ARM Cortex M0+ architecture, Microchip RN2483 LoRaWAN module, Telit Jupiter SE868-A GPS module, Microchip RN4871[[11]](#endnote-10) BLE module, Atmel AT24C256C 32Kx8 Bits EEProm and Atmel ATECC508A[[12]](#endnote-11) crypto authentication chip. An end-device will use LoRaWAN protocol for communication. It will send LoRa packets using radio channels.

## Gateway

A LoRaWAN Gateway — Gateway for short — is device that provides last-mile LoRaWAN radio access to the end-devices. It is the edge component at the end of the LoRaWAN network infrastructure. Gateways are base-stations that deliver the end-device messages to a central network server through a non-constrained backhaul network, e.g., LTE or Ethernet. Basically, a gateway is a multi-channel high performance LoRa transceiver module that can receive, process, and send several LoRa packets simultaneously using different spreading factors on various channels. An end-device will send LoRaWAN messages via LoRa PHY layer technology. The LoRa packets will be received by all gateways within the radio coverage area of the end-device, and will deliver their contents to the central network-server, which perform deduplication tasks. In order to provide scalable massive coverage area, gateways can handle communications from thousands of devices in the range up to a few kilometres in densely populated urban areas, and up to tens of kilometres in rural areas. Therefore, large coverage areas can be covered with a reduced number of gateways, which make LoRaWAN a desirable technology for expansive deployments.

The RHF2S008P4G[[13]](#endnote-12) will be utilized as a gateway. It is designed and developed by RisingHF. It connects to the backhaul network through 4G LTE connectivity or via Ethernet. Additionally, it includes a set of sensors and features like GPS, temperature monitor, RTC and power management unit. This particular model is rugged and protected against outdoor conditions with a dust and water certification rating IP66. It also houses fully integrated a SX1301[[14]](#endnote-13) LoRa high performance transceiver and is installed with a high performance 868 MHz fiberglass antenna with peak gain of 3.0dBi, and runs an optimized Linux kernel over ARM Cortex A53 microprocessor. The gateways and end-devices both will operate in the EU868 ISM band.

## Network Server

The Network Server is part of the LoRaWAN back-end infrastructure. It represents the central hub of all communications from and to LoRaWAN end-devices. It aims to hide the Physical (PHY) and Medium Access Control (MAC) layer details of the LoRaWAN protocol to the components that need to communicate with end-devices. The network server is in charge of collaborating with the end-devices to keep the overall network health, i.e., optimise the data-rate and overall energy consumption of the deployment site, as well as orchestrate what radio configuration parameters end-devices should employ in order to avoid packet loss or unnecessary retransmissions.

The IoTrust project will employ the ChirpStack.io open source LoRaWAN Network Server Stack[[15]](#endnote-14). This project is popular a Free Open-Source Software (FOSS) implementation of the LoRaWAN network server that provides several operation and administrative facilities in order to deploy a network of end-devices. All the components are licensed under the MIT license. Therefore, modifications and improvements can be made commercially available. Its architecture employs several operation and administrative end-points common in the IoT application scenario. These include, a web interface dashboard, standardised protocol event-based broker using MQTT[[16]](#endnote-15) and a REST[[17]](#endnote-16) API over secure HTTPS connections. Therefore, its integration with other IoT libraries and networking components is relatively easy.

Overall, the LoRaWAN network server is a unique component. There is only one single instance per deployment and provides high-level abstraction of end-device communications. This is, applications and users are presented with a high-level abstraction end-point to send and receive messages to and from end-devices. These end-points may be a REST API, an MQTT broker or other customizable solutions. The network server will manage all the low-level details in order to guarantee secure and reliable delivery of messages to and from the LoRaWAN infrastructure.

## IoT Controller

The IoT Controller plays the role of authenticator in the Authentication, Authorisation, and Accounting (AAA) architecture[[18]](#endnote-17). End-devices perform a bootstrapping process when they are deployed for the first time. This process includes an authentication and key agreement stage. Once the device successfully authenticates itself, session keys are shared with the device in order to securely perform the regular operation tasks.

Typically, end-devices transmitting over non-constrained networks perform the bootstrapping by directly addressing any authentication server connected to an IP network. This exchange usually employs a standardised protocol such as RADIUS[[19]](#endnote-18) or Diameter[[20]](#endnote-19) to carry Extended Authentication Protocol (EAP)[[21]](#endnote-20) messages over regular IP networks. However, RADIUS and Diameter require an exchange of relatively large messages with a large number of transmissions. This only exacerbates the problem of energy consumption and radio bandwidth usage due to header overhead for constrained radio technologies such as LoRaWAN.

Therefore, a lightweight Low-Overhead EAP over CoAP (LO-CoAP-EAP)[[22]](#endnote-21) protocol is chosen instead. LO-CoAP-EAP employs the novel Constrained Application Protocol (CoAP)[[23]](#endnote-22) and a set of efficient primitives to significatively reduce the header overhead of transmitting authentication EAP messages over a constrained network. The IoT Controller includes the LO-CoAP-EAP protocol logic that parses the upstream messages transmitted by the end-devices, and forwards its contents to an authentication server that employs typical AAA protocols such as RADIUS or Diameter to carry EAP payloads. Likewise, when the authentication server answers with the new downlink EAP messages, the IoT Controller generates a new LO-CoAP-EAP packet and forwards it to the end-device.

## Authentication Server

The AAA architecture has been proposed by standardisation organisation, such as IETF, to provide a scalable solution to security management tasks in heterogeneous IoT ecosystems, especially those employing long-range wide-area networks[[24]](#endnote-23). At the centre of the AAA architecture, lays the Authentication Server. It provides an administrative end-point that abstracts the technology specific details of deployed end-devices. Thus, the administrator simply manages identity and key materials, and relays on the technology to employ the security mechanisms that fit each specific case. In order to do so, the authentication server employs EAP, a flexible solution that supports several methods, with various degrees of performance requirements for each end-device. On the one hand, more constrained devices may employ lightweight cryptographic primitives, such as AES with the EAP-PSK method. On the other hand, other non-constrained end-devices may rely on more computationally demanding methods such as those based on public key infrastructure, or certificates.

First, each end-device credentials and key information need to be previously configured in the authentication server. Next, the end-device will be installed in its deployment site and will perform the bootstrapping procedure. During the bootstrapping, the device will authenticate itself against the network and will obtain a set of session specific keys. Finally, the end-device finishes the bootstrapping procedure and commences its operation phase, securing the following communications with the obtained keys.

The integration of different kinds of end-devices and networks leverages on the communication technologies of each deployment and the homogeneous security administration interface provided by the authentication server. Additionally, this architecture framework facilitates mobility scenarios with roaming and network-infrastructure assisted hand-over, as well as multi-radio access technologies (Multi-RAT) embedded within devices. Multi-RAT features improve overall network and power efficiency by offloading bandwidth demanding tasks, such as configuration updates or firmware-over-the-air (FOTA), on higher data-rate technologies, such as WiFi or 4G LTE, while relegating the device to a low-power technology, such as LoRaWAN, for small sporadic transmissions.

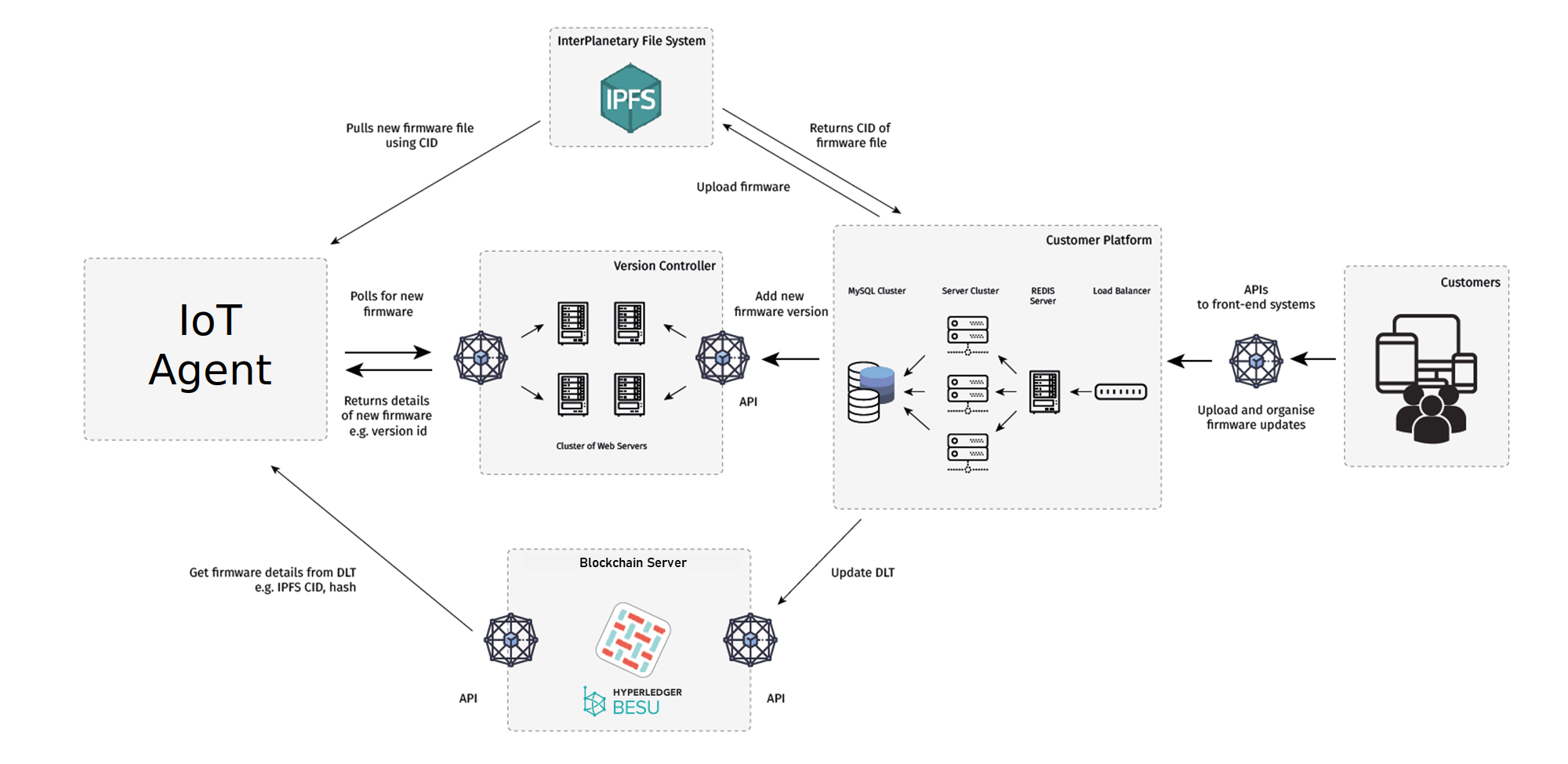
## IoT Agent

The IoT Agent is a MQTT client which subscribes to the topics exposed by the MQTT broker running in the Network Server. At the heart of MQTT are the MQTT broker and clients. The data sent by the end-devices is received by the Network Server over LoRaWAN, which is in turn dispatched using MQTT messages. Each message is posted in a device-specific application reception topic. The IoT agent will subscribe to the topics to receive these messages. Additionally, it will publish messages in the device-specific transmission topics, exposed for this purpose. The topics will post events with device registration, device data, config data etc.

IoT Agent forwards the device metadata and sensor data to the asvin platform. It does it over HTTPs using REST API end-points. The IoT Agent acts as an agent between the network server and the asvin platform.

## asvin Platform

It is a Platform as a Service (PaaS) to facilitate over the air security patches for IoT devices using novel decentralized and distributed technologies. The asvin Platform[[25]](#endnote-24) provides a complete solution for device, security patches and rollout management. It is comprised of 4 components as depicted the figure (figure ref).



Each component of the asvin platform is tailored to perform specific set of tasks efficiently.

### IPFS

Interplanetary File System (IPFS)[[26]](#endnote-12077) protocol is utilized to store firmware and patches. The IPFS is a content-addressable peer to peer method for storing and sharing hypermedia in a distributed file system. It solves problem of duplicate files across the network as it exists in the HTTPs and remove redundancy. When a firmware file is stored on the network a hash is generated based on content of the firmware and stored on blockchain network. This unique hash is called Content Identifier (CID). Subsequently, the CID is utilized to pull the firmware from IPFS server.

### Blockchain

Asvin employs distributed ledger technology to provide an extra layer of security and resiliency to the platform. All events of the platform are recorded in a shared ledger. The ledger also contains critical meta data information of devices and firmware. The blockchain infrastructure is based on Hyperledger Fabric and Besu. Both are private permissioned blockchain network technologies designed and developed under the Linux Foundataion.

### Customer Platform

The Customer platform provides one stop solution to register, control, manage devices, firmware and rollouts. It facilitates an intuitive and wholesome dashboard web interface to customers. It hides the complexity of distributed and decentralized technologies of the asvin platform.

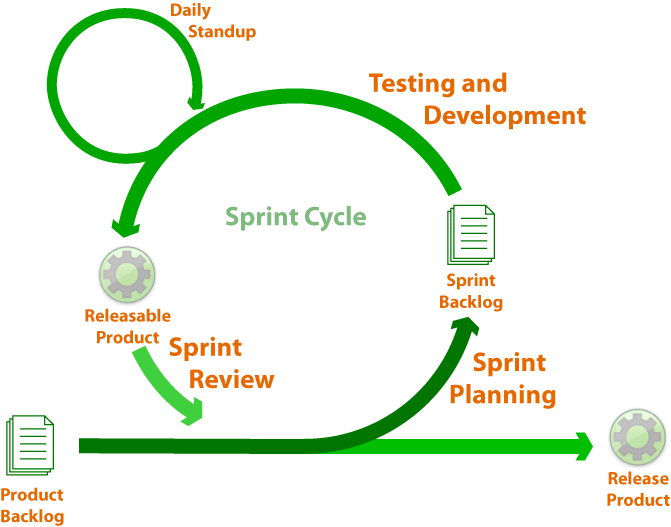
### Version Controller

The Version Controller exposes backend REST APIs for device and rollout management. The Customer Platform and Version Controller work hand in hand. It handles the device registration and rollout success requests from the IoT agent.

The asvin platform exposes its services using the REST API end-points. The other IoT services and platforms can interact with the asvin platform using these APIs. The IoT agent forward its data to the asvin platform using the respective API end-point. The asvin platform can send data to the network server directly as it also has REST API end-points or it can send data through the IoT Agent.

# Conclusions and next steps

The IoTtrust architecture is designed and submitted in the deliverable D.1. This deliverable gives comprehensive information about each component of the architecture. Every component of the IoTtrust architecture is designed after multiple rounds of discussions. It will be employed as a reference for the future deliverables.

[[27]](#footnote-2)

Going ahead in the future, we will develop the components of the architecture using the SCRUM framework. We will plan one week sprint. At the start of the week, we will have a sprint planning meeting. It will include the tasks to be completed, acceptance criterion etc. We will complete the week with the sprint review meeting. All tasks are discussed in the sprint review meeting. The sprint review meeting includes the problems faced, solved and their status. The development backlogs are also examined. This process will be executed iteratively for the IoTrust development.

Appendix

Acronyms

AAA Authentication Authorization and Accounting

API Application Programming Interface

BLE Bluetooth Low Energy

CoAP Constrained Application Protocol

EAP Extended Authentication Protocol

FOSS Free Open-Source Software

FOTA Firmware Over the Air

GPS Global Positioning System

HTTPS Hypertext Transfer Protocol Secure

IoT Internet of Things

IPFS InterPlanetary File System

LoRa Long Range

LoRaWAN Long Range Wireless Area Network

LTE Long-Term Evolution

MAC Medium Access Control

MQTT Message Queuing Telemetry Transport

RADIUS Remote Authentication Dial In User Service

REST Representational State Transfer

RTC Real Time Clock

PaaS Platform as a Service

1. https://www.scrumguides.org/scrum-guide.html [↑](#endnote-ref-1)
2. https://lora-alliance.org/sites/default/files/2018-05/2015\_-\_lorawan\_specification\_1r0\_611\_1.pdf [↑](#endnote-ref-2)
3. http://wiki.lahoud.fr/lib/exe/fetch.php?media=an1200.22.pdf [↑](#endnote-ref-3)
4. https://www.mouser.com/datasheet/2/761/sx1276-1278113.pdf [↑](#endnote-ref-4)
5. https://www.hoperf.com/modules/lora/RFM95.html [↑](#endnote-ref-5)
6. http://ww1.microchip.com/downloads/en/devicedoc/50002346c.pdf [↑](#endnote-ref-6)
7. https://static6.arrow.com/aropdfconversion/5ff647cd30f423703234cbf85de7f2e794f2b199/smarteverythingasmelionuserguide.pdf [↑](#endnote-ref-7)
8. https://lorawan-hackathon.readthedocs.io/en/latest/lion.html [↑](#endnote-ref-8)
9. https://www.arrow.com/ [↑](#footnote-ref-1)
10. https://cdn.sparkfun.com/datasheets/Dev/Arduino/Boards/Atmel-42181-SAM-D21\_Datasheet.pdf [↑](#endnote-ref-9)
11. https://ww1.microchip.com/downloads/en/DeviceDoc/RN4870-71-Bluetooth-Low-Energy-Module-Data-Sheet-DS50002489D.pdf [↑](#endnote-ref-10)
12. https://ww1.microchip.com/downloads/en/DeviceDoc/20005928A.pdf [↑](#endnote-ref-11)
13. http://www.risinghf.com/product/detail/21 [↑](#endnote-ref-12)
14. https://www.mouser.com/datasheet/2/761/sx1301-1523429.pdf [↑](#endnote-ref-13)
15. <https://www.chirpstack.io/> ChirpStack, open-source LoRaWAN® Network Server stack [↑](#endnote-ref-14)
16. OASIS. (2014). MQTT Version 3.1.1. OASIS Standard, October, 81. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html [↑](#endnote-ref-15)
17. https://roy.gbiv.com/pubs/dissertation/top.htm [↑](#endnote-ref-16)
18. Spence, D., Gross, G., de Laat, C., Farrell, S., Gommans, L. H. M., Calhoun, P. R., Holdrege, M., de Bruijn, B. W., & Vollbrecht, J. (2000). AAA Authorization Framework (Issue 2904). RFC Editor. https://doi.org/10.17487/RFC2904 [↑](#endnote-ref-17)
19. Rubens, A., Rigney, C., Willens, S., & Simpson, W. A. (2000). Remote Authentication Dial In User Service (RADIUS) (Issue 2865). RFC Editor. https://doi.org/10.17487/RFC2865 [↑](#endnote-ref-18)
20. Zorn, G. (2014). Diameter Network Access Server Application (Issue 7155). RFC Editor. https://doi.org/10.17487/RFC7155 [↑](#endnote-ref-19)
21. Aboba, B., Blunk, L., Vollbrecht, J., & Carlson, J. (2004). Extensible Authentication Protocol (EAP) (H. Levkowetz (ed.)). https://doi.org/10.17487/rfc3748 [↑](#endnote-ref-20)
22. Garcia-Carrillo, D., Marin-Lopez, R., Kandasamy, A., & Pelov, A. (2017). A CoAP-Based Network Access Authentication Service for Low-Power Wide Area Networks: LO-CoAP-EAP. Sensors, 17(11), 2646. https://doi.org/10.3390/s17112646 [↑](#endnote-ref-21)
23. Shelby, Z., Hartke, K., & Bormann, C. (2014). The Constrained Application Protocol (CoAP). In Journal of Chemical Information and Modeling. https://doi.org/10.17487/rfc7252 [↑](#endnote-ref-22)
24. Farrell, S. (2018). Low-Power Wide Area Network (LPWAN) Overview (Issue 8376). RFC Editor. https://doi.org/10.17487/RFC8376 [↑](#endnote-ref-23)
25. https://asvin.readthedocs.io/en/latest/ [↑](#endnote-ref-24)
26. ttps://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf [↑](#endnote-ref-12077)
27. https://carlleedrowsycat.wordpress.com/2014/09/28/what-is-an-agile-sprint-retrospective/ [↑](#footnote-ref-2)