6-18-2024

A Framework for BLE Device Setup

Benjamin Hofmann
University Neu-Ulm, hb@beohof.de

Tobias Engel
University Neu-Ulm, tobias.engel@hnu.de

A Framework for BLE Device Setup

Benjamin Hofmann
University of Applied Sciences Neu-Ulm, Faculty Information Management, Technology Transfer Center (TTZ Leipheim) Smart Production and Logistics
hb@beohof.de

Tobias Engel
University of Applied Sciences Neu-Ulm, Faculty Information Management, Technology Transfer Center (TTZ Leipheim) Smart Production and Logistics
tobias.engel@hnu.de

ABSTRACT

This paper explores and evaluates a dedicated setup process for appliances connected to the Industrial Internet of Things (IIoT). These differ significantly from classic IoT devices because they often need certified and highly specialized labor for installation and follow an inherently different life cycle. The emphasis lies on the configuration of devices using Bluetooth Low Energy (BLE) highlighting the challenges in establishing secure and efficient communication practices between the actors in the configuration procedure. The derived protocol is implemented prototypically using off the shelf hardware components (Raspberry Pi, ESP32) and standardized encryption components used by internet services. A crucial part is determining if desktop grade encryption can be used to secure connection with embedded devices having significantly less computing power than classic computer equipment. An important focus point is the design fitting a broad range of infrastructure (wireless/wired) and providing a template to adapt the protocol for existing applications.

Keywords

IIoT, BLE device setup, setup protocol, encrypted device setup

INTRODUCTION

The Industrial Internet of Things (IIoT) sector has seen substantial growth and will continue to expand in the following years (Al-Sarawi, Al Hawari, Anbar and Abdullah, 2020 p. 2). This trend not only applies to large-scale enterprises but also to SMEs (small and medium-sized enterprises) in different contexts in which appliances are operated. IIoT devices such as machines and environmental sensors have vastly different requirements such as the infrastructure, the exchange of data for different applications and network topologies (wireless, wired), or also the usage of different communication protocols such as MQTT, HTTP, etc. For this reason, generalization of a setup process is a difficult endeavor, creating the need of a certified technician.

Hence, this paper proposes a framework including a protocol to deploy configuration data to edge-devices generically, securely and in a self-contained manner. In addition, the framework includes a process that does not rely on extended features of the operating system, such as device pairing. This prevents application data being stored outside the context of the application which increases the security of the setup protocol. The framework addresses the simplification of IIoT appliance configuration and provides software and hardware engineers with a template for a broad range of use-cases.

THEORETICAL BACKGROUND

The main goal of the framework and the derived protocol is to provide software and electrical engineers with a template enabling rapid deployment without relying on external network components and leaving orphan data on technician devices. Further, the protocol uses Bluetooth Low Energy (BLE) and desktop grade encryption. To validate the pairing process, the authors use firmware deployable on a ESP32 microcontroller exemplary for a low powered appliance and a python script representing the peer side for setup. Moreover, the proof of concept shows how authentication data for local networks and backend system can be transmitted and how desktop security protocols can be utilized on low powered IoT-Devices on the ESP32 platform without introducing a prohibitively high delay in the setup process.

Bluetooth Low Energy

BLE is designed for very low-power operation and therefore a good fit for IoT devices (Bluetooth SIG, 2023). When two BLE devices communicate, they form a server/client relationship. Servers offer state data known as characteristics, which clients can use to retrieve data and to manipulate the state of the application/server (Woolley and Bluetooth SIG, 2023, p. 7). A service offered to clients is defined by its identifier and the associated characteristics. Additionally, a service contains descriptors, providing metadata about the characteristic. This can be a textual description for humans, or the purpose e.g., reading or writing (Woolley and Bluetooth SIG 2023, p. 67). A service specification can be thought of as defining one aspect of a server device's behavior (Woolley and Bluetooth SIG, 2023, p. 8).

Serial over BLE (Nordic-UART)

UART (Universal asynchronous receiver / transmitter) defines a protocol and set of rules for exchanging serial data between two devices, (Rohde & Schwarz GmbH & Co KG, 2024). The Nordic Semiconductors company defines guidelines on how to implement this protocol over a BLE connection (Nordic Semiconductor, 2024). Data transmitted over the Nordic-UART channel is further structured and processed to ensure reliable and secure transmission.

UTF-8

UTF-8 (8-Bit Unicode Transformation Format) is a ubiquitous character code standard. It is used to encode strings in a well-known format, allowing basic text processes like the rendering of character-sequences. A crucial process is treating text as bulk data for operations as compressing, truncating and transmission (The Unicode Consortium, 2023, p. 10). These are integral functions of communication protocols, and UTF-8 is therefore further defined in RFC-3629. The software appended to this paper enforces its usage to ensure the processing of exchanged data is executed on a unified character set.

JavaScript Object Notation

JavaScript Object Notation henceforth JSON is a lightweight, text-based and programming language agnostic format used for data interchange. It defines a small set of formatting rules for the portable representation of structured data in text format (Bray, 2017, p. 1) and is standardized in RFC-8259. In the proof of concept, it is used to ensure proper serialization/deserialization of data.

Base64

The Base 64 encoding defined in RFC-3548 is designed to represent arbitrary sequences of octets (binary data) that need not be human-readable. (Josefsson, 2006, p. 5) This is useful for a multitude of applications, especially for ciphertext, as UTF-8 characters that have been encrypted may not be a valid JSON strings and therefore serializing such data will fail. This is solved by first converting to base64, which results in a string that can be encoded with a character set and then serialized.

Mbed-TLS and Cipher-Suite

The Mbed-TLS Library focuses on embedded applications and provides the building blocks for secure communication, cryptography and key management. Furthermore, it is designed in the portable C language and targets a broad variety of embedded systems (The Mbed TLS Contributors, 2023). The manufacturer of the ESP32 microcontroller espressif provides a version of Mbed-TLS offering support and hardware acceleration for its functions (Espressif Systems Co. Ltd., 2023). The ciphers used in the proof of concept are the SECP256R1 elliptic curve for key agreement, SHA256 for verification and ChaCha20-Poly1305 utilized in symmetric encryption after the initial handshake. For the verification of the target device, the protocol uses a dedicated key pair which is preceded by a newly generated one to ensure ephemeral use of the cryptographic keys.

Trust on first use

The trust-on-first-use authentication approach assumes that an unauthenticated public key provided on first contact will be good enough to secure future communications (Dukhovni, 2014). This opportunistic approach to security is implemented to allow generic client-side devices to initiate the setup procedure from an unknown actor in the field and to adapt devices into new target environments. Additionally, it ensures that the initial programming of the microcontroller can be executed without providing parameters of the target infrastructure.

RESEARCH METHOD

Design Science is fundamentally a problem-solving paradigm, providing a general framework and useful procedures revolving around the creation of innovations and artifacts (Hevner, March, Park and Ram 2004, p. 3). A crucial result of design science are artifacts, which are broadly defined as *constructs* (vocabulary and symbols) and *instantiations* (implemented and prototype systems). These are ultimately concrete prescriptions that enable IT researchers and practitioners to understand and address the problems inherent in developing and successfully implementing information-systems (Hevner, March, et al. 2004, p. 4). We deployed the design science cycle from (Hevner, March, et al. 2004, p. 7), whereby the knowledge base (rigor cycle) and the environment (relevance cycle) influence the creation of the artifacts and we iteratively evaluate the artifacts by simulating the created protocol in a self-contained environment, based on a Raspberry Pi and an ESP32 microcontroller. This setup was chosen to provide an easily deployable package for third parties.

Target Infrastructure: 3-Party Handshake

Figure one depicts how the protocol can be used to pair IoT devices to a backend service. An intermediate device is used to provide the initial configuration, networking credentials, and to relay encrypted authentication data. This ensures that confidential data is isolated from third parties. This form of communication mandates the backend being able to verify payloads with the cryptographic fingerprint of the IoT device — what must be implemented on a per-application basis.

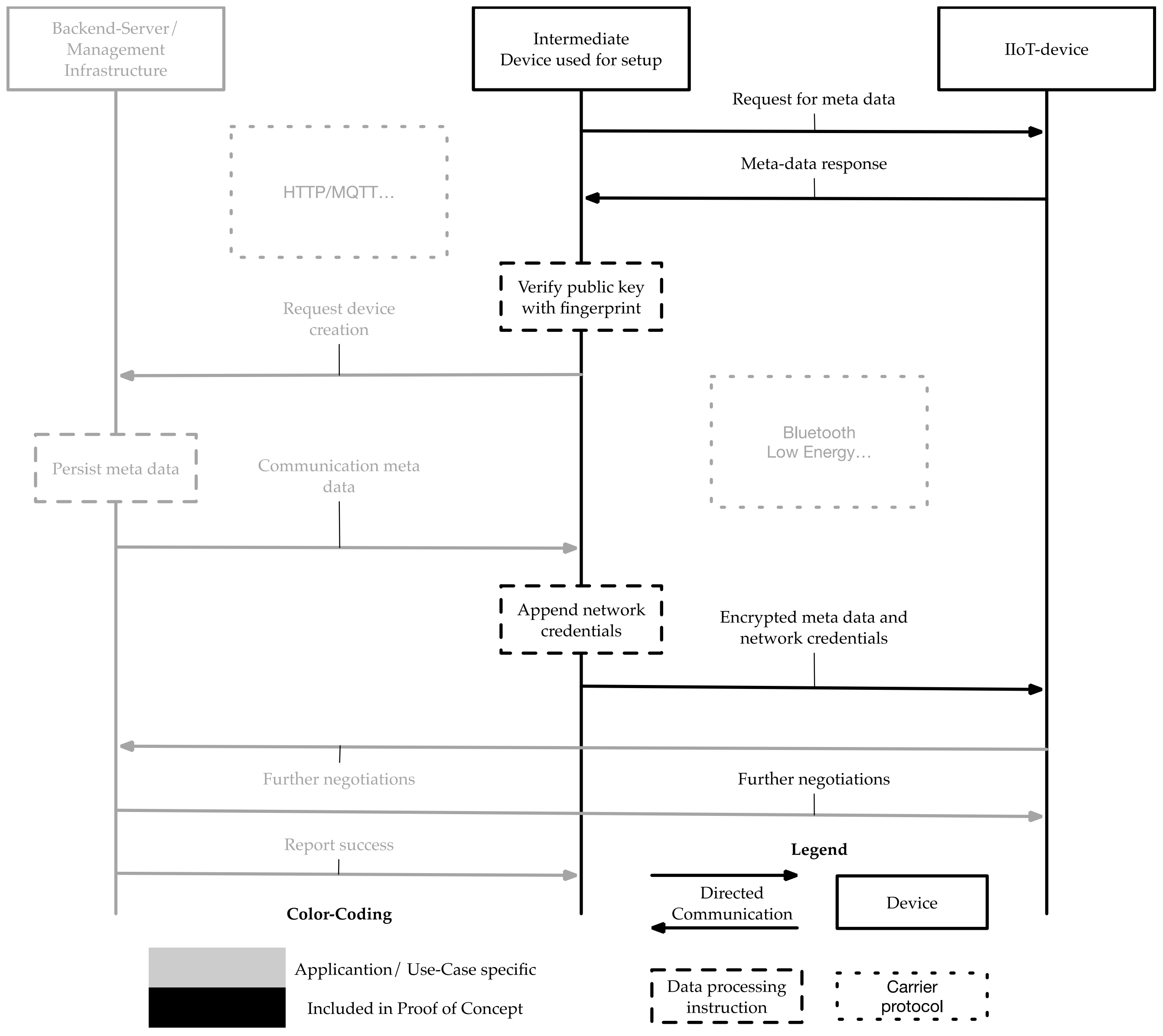

Backend-Server/ Management Infrastructure
Intermediate Device used for setup
IIoT-device
Request for meta data
Meta-data response
HTTP/MQTT…
Verify public key with fingerprint
Request device creation
Bluetooth Low Energy…
Persist meta data
Communication meta data
Append network credentials
Encrypted meta data and network credentials
Further negotiations
Further negotiations
Report success
Legend
Directed Communication
Device
Color-Coding
Applicantion/ Use-Case specific
Included in Proof of Concept
Data processing instruction
Carrier protocol

Figure 1 Protocol Diagram of the 3-party handshake (own source)

Exchange of Data

Data between the intermediate and target device are exchanged via BLE as a bidirectional communications medium. The target device acts as a server and the intermediate device as a client. UTF-8 Data written to the TX-Characteristic is processed by the IoT-device, and the response or error message can be received on the RX-Characteristic. The transmission of a single message is finished once the UTF-8 character for a line break is sent. To ensure the correct transmission of data and the order of chunks, the indicate attribute is used. This introduces the confirmation of messages (Bluetooth SIG, 2021, p. 1452) instead of the notify attribute that defines messages being sent on a best-effort basis.

The data format for transmission is JSON, from which data is easily extracted programmatically. The messages must include a processing command/directive to allow the distinction of IoT-device commands and capabilities. Table one shows an example message and describes its general structure.

Example-Structure	***Description***
{	
“command”: “example”,	Instruction/Function to be executed on the payload.
“payload”: {	Payload is an object wrapper for arbitrary data
„additional_data“: …	Data needed for the execution of the command
}	
}	

Table 1 Message Structure

Proof of Concept

The first step of the device configuration is establishing a secure communication channel. The code section shows the intermediate device initiating the configuration process by sending the “init_config” command and its key-exchange, which is a base64 encoded representation of the public part of a randomly generated key-pair.

```
{
  "command" : "init_config",
  "payload" : {
    "key_exchange": "AwAXQQSpx53lUch+r8HP1scHvPB6NN9GSwrDPD8…"
  }
}
```

The next code section depicts the response of the IoT-device, it contains a key-exchange that has been generated after the initial message and a signature field. The additional field contains a signed SHA256 hash of the key-exchange. If the signature does not match the key-pair, the process will stop with an error message.

```
{
  "key_exchange": "QQSWMf6Xjazj9…",
  "signature": "MEUCIEcZ0xf9oVdj…"
}
```

After the above steps are finished, both devices can derive a symmetric encryption key used for further transmissions and the actual configuration can commence. Below is the massage in plain text.

```
{
  "device_id": "auniqueid"
  "psk": "asupersecurepresharedkey",
  "ssid": "ssidofasecurewifi"
  "backend_server": "iot.example.invalid"
  "backend_server_certificate": "-----BEGIN CERTIFICATE…"
  "backend_auth_token": "AwuSuXe3+…"
}
```

After encrypting the configuration and sending it to the peer device, it will respond with a “configuration finished” message and the connection is closed. The code block bellow shows the encrypted configuration message and additional data. The field nonce is required by the ChaCha20-Poly1305 cipher and must not be repeated for succeeding messages (Nir and Langley, 2015). The tag field allows the peer device to verify the command string.

```
{
  "command": "set_config",
  "payload" : {
    "ciphertext": "iWF1FH5pWUatopKKWg9InALKJtRDkrs9whCzpT6…"
    "nonce" : "u+xnTf0iAImyAOyq",
    "tag" : "jad0Yds25f0dE0Cxn4TYLQ=="
  }
}
```

Setting up the Simulation/Proof of Concept

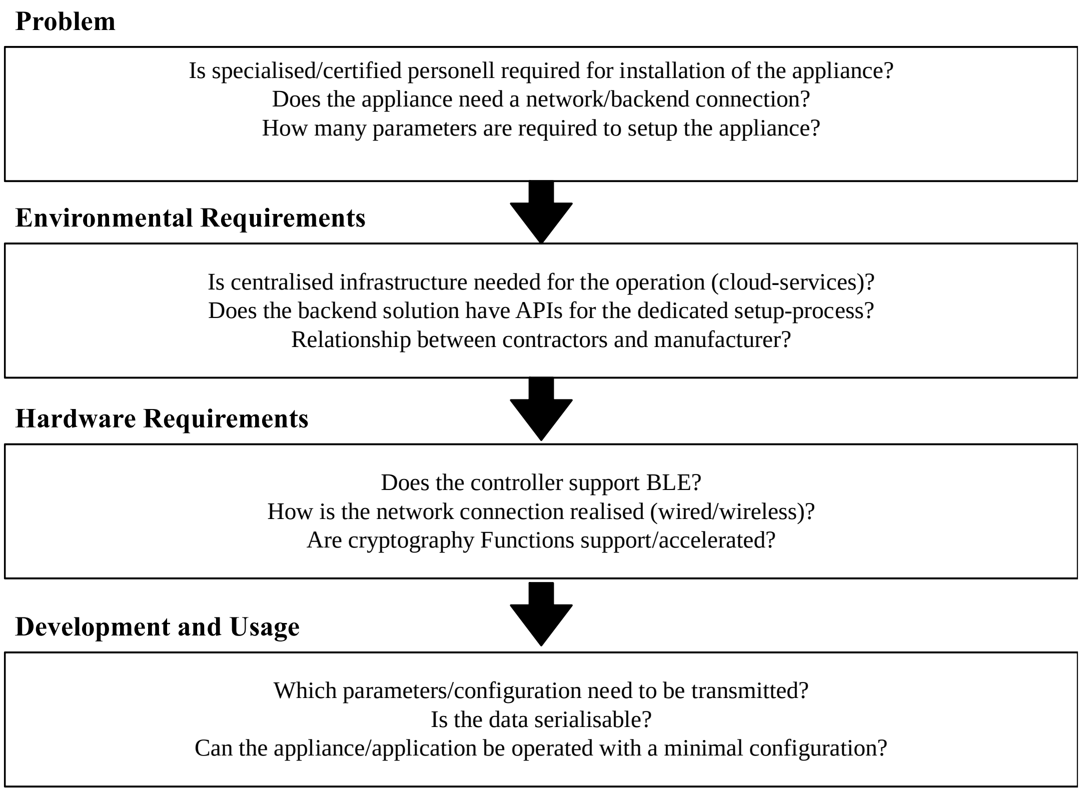
To run the proof of concept, a Raspberry Pi Model 3 or 4 and an ESP32 are required. Table two contains links to the technical specifications of both devices. For a simple setup process, an OS image including the required software and setup process description can be downloaded from: https://url.beohof.de/CXL7W. This can be flashed with the "Raspberry-Pi Imager". After flashing, connect an ESP32 and run "sudo esptool.py --chip ESP32 write_flash 0x00000 /var/firmware/esp32.bin". Lastly execute the proof of concept with "ble_conf_poc.py".

Technical Specifications	
Raspberry Pi Model 4	https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications (Raspberry Pi Ltd, 2024)
ESP32	https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (Espressif Systems Co. Ltd., 2023)

Table 2 Hardware specifications

DISCUSSION, LIMITATIONS, AND FUTURE RESEARCH

As a result of the simulation and the prototypical implementation, figure two illustrates the most important points for assessing if a dedicated setup protocol is recommended for an IoT-Suite.



Problem

Is specialised/certified personell required for installation of the appliance?
Does the appliance need a network/backend connection?
How many parameters are required to setup the appliance?

Environmental Requirements

Is centralised infrastructure needed for the operation (cloud-services)?
Does the backend solution have APIs for the dedicated setup-process?
Relationship between contractors and manufacturer?

Hardware Requirements

Does the controller support BLE?
How is the network connection realised (wired/wireless)?
Are cryptography Functions support/accelerated?

Development and Usage

Which parameters/configuration need to be transmitted?
Is the data serialisable?
Can the appliance/application be operated with a minimal configuration?

Figure 2 Assessing the need for a custom configuration protocol

The concept of independence from external networks addresses a substantial challenge in the IoT – the reliance on third-party actors and network availability. This constraint ensures consistent functionality in environments where network access is limited or temporarily unavailable. The isolation of the setup process from the application/usage specific life cycle can help technicians to eliminate time spent on searching for a reliable network connection in unknown environments.

Another benefit of a dedicated protocol for device instantiation is the reduction of orphan data. This results from applying the trust on first use principle and ephemeral keys. It enables frequent users to exercise a more rigorous security hygiene and reduces the risk of confusing the devices of the same make.

One of the major challenges of implementing a dedicated setup protocol is the integration with existing systems. It necessitates identifying environment-specific parameters, which can vary greatly based on the industrial setting and the device/protocols used.

CONCLUSION

The proof of concept is limited to communication between intermediate and target devices. As backend services are a fundamental part of a complete IOT-infrastructure the payloads contain data regarding a fictional backend. This part of the system can be investigated in following papers. Although the proof of concept uses existing functions for cryptography, the concept should be further evaluated to mitigate security risks and ensure their implementation is formally correct. Furthermore, a role-based scheme can be implemented to ensure authentication against the backend and to provide authorization for the invoked functions.

In conclusion, this paper presents the exploration and evaluation of a protocol to set up appliances with Bluetooth Low Energy allowing for the secure transmission of device and network meta data. The prototype successfully demonstrates that embedded system like the ESP32 have enough computing power to use desktop grade encryption and standard protocols used by internet services. The template in this paper provides software and hardware engineers with a way to decouple the setup process from their IIoT application and streamlines the usage of the data for existing and new use-cases.

REFERENCES

1. Al-Sarawi, S., Al Hawari, A. B., Anbar, M., and Abdullah, R. (2020). *Internet of Things Market Analysis Forecasts, 2020 - 2030*.
2. Bluetooth SIG. (2021). *Bluetooth Core Specification*.
3. Bluetooth SIG. (2023). *Bluetooth Technology Overview*. (https://www.bluetooth.com/learn-about-bluetooth/tech-overview/, accessed November 14, 2023).
4. Bray, T. (2017). *The JavaScript Object Notation (JSON) Data Interchange Format*. (https://doi.org/10.17487/RFC8259).
5. Dukhovni, V. (2014). *Opportunistic Security: Some Protection Most of the Time*. (https://doi.org/10.17487/RFC7435).
6. Espressif Systems Co. Ltd. (2023). *Mbed TLS - ESP32*. (https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/mbedtls.html, accessed January 3, 2024).
7. Espressif Systems Co. Ltd. (2023). *ESP32 Datasheet*. (https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf).
8. Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). *Design Science in Information Systems Research*.
9. Josefsson, S. (2006). *The Base16, Base32, and Base64 Data Encodings*. (https://doi.org/10.17487/RFC4648).
10. Nir, Y., and Langley, A. (2015). *ChaCha20 and Poly1305 for IETF Protocols*. (https://doi.org/10.17487/RFC7539).
11. Nordic Semiconductor. (2024). *Nordic UART Service (NUS)*. (https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/libraries/bluetooth_services/services/nus.html, accessed January 15, 2024).
12. Raspberry Pi Ltd. (2024). *Raspberry Pi 4 Model B Specifications*. (https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/, accessed January 16, 2024).
13. Rohde & Schwarz GmbH & Co KG. (2024). *Understanding UART*. (https://www.rohde-schwarz.com/us/products/test-and-measurement/essentials-test-equipment/digital-oscilloscopes/understanding-uart_254524.html, accessed January 15, 2024).

14. The Mbed TLS Contributors. (2023). *Mbed TLS Tutorial.* (https://mbed-tls.readthedocs.io/en/latest/kb/how-to/mbedtls-tutorial/, accessed January 3, 2024).
15. The Unicode Consortium. (2023). *Unicode 15.1.0*. (https://www.unicode.org/versions/Unicode15.1.0/).
16. Woolley, M. and Bluetooth SIG. (2023). *The Bluetooth LE Primer*. (https://www.bluetooth.com/wp-content/uploads/2022/05/The-Bluetooth-LE-Primer-V1.1.0.pdf).