Design and Implementation of Security Protocols in Wake-up Radio enabled 5G Networks

Gianmarco Pastore & Arthur Quach

in

IKT454
5G and IoT

Supervised by Frank Y. Li and Harsha Sandaruwan

Department of Information and Communication Technology
Faculty of Engineering and Sciences
University of Agder

Grimstad, December 2022

Abstract

The Internet of Things is becoming increasingly important and the applications more demanding. Wake-up Radio technology appears as a solution to overcome power related limitations in IoT devices. However, Wake-up Radio systems are subject to attacks, notably Denial-of-Service attacks in order to deplete its battery. In this report, we implement a security protocol on Wake-up radio enabled devices in a Wireless Sensor Network scenario related to the world of agriculture without greatly undermining the performances or the power optimization. We describe the protocol to explain how the devices are communicating with each other. Using Python as a programming language and different Python libraries, we implement the multiple devices that will take part in the communication. Through testing in different case scenarios, we first demonstrate that security features added to the protocol do not greatly hinder its efficiency. Then we prove that the more cluster-heads are communicating together, the higher the computation time and transmission time will be. After, we show that un-assisted Device-to-Device communication is ten times faster than assisted Device-to-Device communication. Finally, we study the threat model for our protocol and explain why these attacks could not be carried out.

Contents

List of Figures

List of Tables

Listings

1 Introduction

The Internet of Things (IoT) is becoming increasingly important and the applications more demanding. In various cases such as smart agriculture [10] the battery capacity can be a challenge. Limited sizes and wireless devices require solutions to optimize their power consumptions. These solutions include limiting to the minimum the functioning of the communicating device, notably with Wake-up Radio (WuR) technology. WuR systems are composed of two transceivers at the receiver: A primary transceiver for main data communication called Main Radio (MR) and a secondary receiver called Wake-up Radio receiver (WuX). The WuX, upon receiving a Wake-up Call (WuC), is used to switch the primary transceiver from deep sleep state to operational. This WuX is always listening for a WuC but it has very low-power consumption [9]. In this way saving energy from the IoT device's MR. The number of IoT devices in the world is also growing as it is believed that there will be more than 10 million devices per km^2 by 2030 according to [2]. This implies a possible overhead to the cellular networks. Using Device-to-Device (D2D) communication is one of the solutions in improving resource utilization of the cellular system [4]. D2D communications uses local direct links between devices, therefore offloading the traffic through the base station. However, WuR systems are subject to attacks, notably Denial-of-Service (DoS) attacks by repeatedly sending a WuC to a device in order to deplete its battery. In this report, we study security from the point of view of the communication channel and do not consider the computational capabilities of the devices. By refering to [14], we develop a protocol to have secured communication using WuR systems and D2D. In this solution, we generate unpredictable Wake-up tokens for each WuC to be new within a concrete and precise use case.

1.1 Background

There are multiples implementation of the WuR but we are going to refer to the implementation made in [7]. As explained shortly in the above section, WuR consist in a very low consumption and always listening receiver connected to the device's MR. The device's MR, when not sending or receiving data, is most of the time in deep sleep mode to conserve its power. When a device wants to communicate with another device which is in deep sleep mode, the sleeping device has to wake up to start the communication. To do so, the initiating device's MR sends a signal to the WuR of the sleeping device. This signal is an On-Off Keying modulated signal based on the address of the targeted device. It will be received by the WuR of the sleeping device. This WuR, after demodulating the signal, will send an interrupt signal to the reset pin of his sleeping device's Micro Controller Unit (MCU). The interrupt signal will put the MCU from deep sleep mode to light sleep mode. The light sleep mode will permit the MCU partial functions. Notably to receive and verify address sequence sent by his WuR. While the MCU is in light sleep mode, a timer is set to prevent it from waiting forever. If the address sequence is verified, the device will be put in active mode and communication between both device's MR will start. Once the transmission is over, the devices will go back to deep sleep mode.

After the first communication, a device can exchange information to later communicate through D2D communication. D2D communication is a technique to improve resource utilization of cellular networks by offloading the traffic through base station to local direct links among devices (in close proximity). In this work, we are going to refer to a network assisted D2D communication explained in [6]. In a cellular system, the device exchange signaling with base station (BS) periodically so that they can set up a cellular connection immediately if necessary. However, there is no such signaling exchange between the devices. Therefore, peer discovery is needed before two devices can set up

a D2D connection and start direct communication. Two devices are D2D candidates if they find each other during the peer discovery process. However, these D2D candidates can only exchange information over the direct link if the criterion for mode selection is satisfied. Mode selection is a process of deciding whether the D2D candidates should communicate in D2D mode or should just stick to cellular mode, looking at the channel quality. Moreover, when D2D candidates have decided to work in D2D mode, mode selection can further choose whether the D2D link is allowed to reuse resources with the cellular links. For the general procedure of D2D communication, two phases can be defined, the discovery phase and communication phase:

- **Discovery Phase**: In the discovery phase, the device searches for the potential peer in proximity for D2D communication and determines the identification of the founded peer. The new pair is determined to be D2D candidates. Moreover, this phase includes a number of messages that have to be exchanged between devices and between devices and BS, providing information about their respective link qualities. Once this information is available at the BS, it may serve as the basic input to the mode selection in the communication phase.
- **Communication Phase**: After completing the discovery phase, the new D2D candidates can have actual communication. The communication phase includes channel estimation, mode selection, resource allocation, power control, and the actual transmission of the information.

The general discovery procedure for D2D communication is described as follows. At first, a device sends a discovery signal to detect potential devices in proximity. Then the identity of devices can be exchanged between the new pair which are determined to be D2D candidates. At last, the message about the link quality is exchanged between devices and between devices and BS under the control of the BS. There are two types of peer discovery approaches for D2D communications: direct discovery and network assisted discovery. But we are interested in the network assisted discovery. In network assisted discovery, the devices detect and identify each other with the assistance of the network. The device informs the BS about its intention to communicate with another device and sends the beacon signal. The beacon signal is a reference signal sequence that can include information such as device ID. Then the BS orders some message exchanges between the devices, in order to acquire identity and information about the link between them.

However, security in WuR has not been the main focus when looking at works related to it. According to [1], if an attacker can listen and transmit on the WuR frequency and the MR channel, he can intercept the WuC and replay it to wake up the device until exhaustion. He can also keep the target node awake after a legitimate communication takes place, by continuously sending on the MR data packets destined to it. This is mainly made possible because the WuR address used for the WuC is static so it can be used over and over again. In [18], they emphasize the fact that since no other identification mechanism is included, the MCU needs to wake up from deep sleep every time a WuC arrives to the node. This happens even if the address sequence is not related to the node, in other words, if the node is not the desired destination. Moreover, in [21], the simplicity of the hardware design of the WuR and the modulation makes it questioning its robustness against interferences and collisions. According to [16], multiple attacks on WuR exist. They can go from flooding attacks, which generate a large amount of packet traffic in order to interrupt a system, to connection attacks, which is a sort of Distributed Denial-of-Service (DDoS) and require an excessive number of sessions.

1.2 Problem Statement

In this work, we will study how we can secure our sensor network to prevent any attacks without greatly impacting the power consumption of the devices.

The main attack against WuR we will have to cover is the DoS which consists in intercepting WuCs and replaying them towards WuR devices. The goal of the attackers is to keep the devices awake in order to deplete its battery. In order to define a solution, we must also take account of confidentiality and integrity of the data. Confidentiality aims to prevent unauthorised access to sensitive information. Such access may be intentional, for example an intruder entering the network and reading information. The two main ways of ensuring confidentiality are encryption and access control. Integrity has three objectives that contribute to data security: Preventing modification of information by unauthorised users, prevent unauthorised or inadvertent modification of information by authorised users and maintain internal and external consistency. But in this work, we are more interested in preventing modification by unauthorised users and also detecting these modifications. Various encryption methods can help ensure integrity by confirming that a message has not been altered during transmission. Alteration can render a message inaccurate. If a message is corrupted, the encryption system must include a mechanism to indicate that the message has been corrupted or modified.

1.3 Methodology

Our goal is to implement a security protocol on WuR enabled devices in a Wireless Sensor Network (WSN) to especially prevent DDoS attacks without greatly undermining the performances or the power optimization. The devices must also have the option to communicate with D2D communication. To begin with, through related work review, we have to verify if this is achievable or already has been done. Then we have to imagine a real case scenario of a WSN with WuR enabled devices. This scenario must be precise to justify the use of a secure WuR and D2D communications. After having our network scenario, we have to design the protocol on how they are going to communicate safely. This protocol must counter the attack models we will later describe and allow WuR and D2D. Then, we have to implement the protocol on the selected environment. We have to represent every communicating entities. Additionally, we have to implement variables to track the performances of our implementation and vary some parameters in order to compare the results. Finally, we have to conclude on the results we have obtained and discuss about future work this conclusion leads to.

1.4 Report Outline

The remainder of this report is organized as follows. In Section 2, related work are reviewed. In Section 3, we explain the system design with the scenario and the protocol design. In Section 4 we present the implementation of the protocols, with references to the code. In Section 5 we report the results of the performance evaluation carried out on the protocols. Finally, in Section 6, we discuss the conclusion of this report and future work.

2 Related Work

2.1 5G and IoT

The development of 5G wireless networks [23] unlocks whole new IoT solutions. By introducing new technologies such as mmWave Spectrum and mMIMO, they bring advantages in terms of spectrum efficiency and network capacity. The spectrum increase and the new technologies are necessary to satisfy 5G technical demands. These demands in 5G consist of three technical pillars: Enhanced Mobile Broadband (eMBB), Massive Machine Type Communications (mMTC) and Ultra-Reliable Low Latency Communications (URLLC). In eMBB, the goal is to provide large capacity for mobile broadband services. In mMTC, we allows the connectivity of a massive number of devices. URLLC is used for applications for instance in smart factories and autonomous vehicles.

The most important use case for IoT is mMTC which is projected to grow considerably. According to [17], tens of billions of devices expected to be connected in the next few years. To meet the 5G expectations about mMTC, some objectives has been defined. The devices and networks must not be complex or expensive. We will be able to count a connection density of 1 million devices per square km. The battery life of the devices must be beyond 10 years considering 200 bytes and 50 bytes of uplink and downlink data per day with a better capacity of 5 Wh. And finally a maximum latency of 10 seconds on the uplink to deliver a 20 byte application layer packet. One of the requirement to achieve this is Low-Power Wide-Area (LPWA) networks for connectivity between devices which includes for instance LoRa, Sigfox or 3GPP LTE Narrow Band (NB)-IoT.

2.2 Wake-up Radio

Energy consumption is critical for battery-powered devices in the IoT and WSN, which have to be operational during a long time. Moreover many devices intended for IoT applications will only transmit and receive small amounts of data very rarely, say, a few times a day. In many cases, there is sufficient energy in a coin-cell battery for all useful communication in such a device's lifetime. The problem is that in this scenario an IoT device does not know when to expect to receive data, and thus must perform energy-draining scanning to detect the presence of a packet. Traditionally, duty cycling (DC) has been a major mechanism for energy conservation in IoT systems like WSNs. By allowing nodes to wake up and sleep periodically or aperiodically, a high percentage of energy saving is achieved. However, idle listening and overhearing are problems that cannot be avoided. For this reason, in the recent period, interest is beginning to be shown in WuR systems that bring numerous advantages over MAC protocols based on DC.

In [15] the performance of the WuR platform is compared to some of the most well-known and widely employed duty-cycled MAC protocols for WSN and is demonstrated how the use of WuR presents numerous benefits in several areas, from energy efficiency and latency to packet delivery ratio and applicability.

Also in [3] the authors show that asynchronous Wake-up MAC protocols are in general better than all the other MAC protocols in terms of reducing energy consumption and latency. In paritcular in this survey they provide a taxonomy and deep analysis about the state-of-the-art Wake-up MAC protocols.

Many implementations, system prototyping, and protocols design of WuR systems have been proposed in the last decade. In [7] a WuR-based two-tier system, which offer cellular IoT connectivity via a Bluetooth low energy (BLE)-enabled smartphone. The prototype presented achieves a current consumption of 390 nA and a response time of 95 ms for decoding a WuC.

In WuR implementation, another aspect to take into account, in addition to power consumption, is the sensitivity of the device. The integrated solution reported in [5], for example, has the aim of pushing the sensitivity vs. power trade-off to the limit, achieving a power consumption of 13nW and a sensitivity of -54 dBm. Instead in the implementation proposed in [13] the authors focus mainly on sensitivity, estimated to be around -83 dBm, able to cover an estimated distance of 1200 meters, but at the expense of higher power consumption.

The recent standardization activities to enable WuR in both 5G and wireless fidelity (WiFi) networks has led to a growing interest in applications of WuR to real-life systems. In [19] is provided an overview of Wake-up-based access in 5G systems, considering the associated prospects, benefits, and challenges. Regarding WuR performance in 5G NR networks, in [20] power consumption and latency have been analyzed. In particular delay-constrained optimization problem has been formulated, to maximize the device energy-efficiency under given latency requirements. The proposed solution achieves a reduction of the power consumption by up to 40% compared with an optimized discontinuous reception.

2.3 D2D

D2D communication is expected to be part of IoT because it is a way to optimize and improve the resource utilization. In [12], D2D can contribute to reach high data rate and low latency because of the short range communications and the proximity of the devices. Additionally, the small data from multiple devices can be aggregated and collected by a better device that will have a higher data rate link to send the data. Moreover, D2D guarantees a low energy consumption communication, again because of the short range with the neighbouring devices which will involve a lower transmission power.

D2D communication can also optimize the power consumption of the devices by being assisted by the cellular network. In [22], they study a single cell environment with its BS capable of choosing the best resource sharing scheme for cellular and D2D communications to share the resources. By proper power control, they have managed to coordinate the interferences between cellular and D2D communication and prioritize either the cellular or D2D users. They showed that the interference between the two links can be coordinated to increase the sum rate without overwhelming the cellular service.

2.4 Security Mechanisms for Machine-Type Communications

One of the aspects of WuR technology that has not been much explored in recent literature is that of security. The biggest threat toward WuR are the attacks of the type DoS, where attackers continuously send WuCs to deplete the battery of the nodes. In fact, WuCs[1] can be obtained by an attacker, who listens to the communication channel, and replayed to keep the receiver awake. In [16] is proposed a protocol against DoS attacks, which reduces the average packet delay by 62.84%, and reduces the average power consumption by 93.71% while the flooding attack vulnerabilities.

The most effective and simple approach to avoid DoS attacks consist of generating and updating the WuR address of each node in a pseudo-random fashion. A few different ways to handle the WuR addresses have been presented recently. In [1] the addresses are generated based on key material known only by authorized peers. To manage the exchange of the secret key among legitimate nodes, a strong and secure Key Management Protocol (KMP), based on Elliptic Curve Cryptography, has been used. Another solution has been reported in [14]. The authors have created protocol to

[1] Wake-up Calls

generate hard-to-guess Wake-up tokens at every wake up, leveraging hash functions, that manage to increase the lifetime of an IoT node by more than 40 times under DoS attack, with negligible energy overhead (0.03%).

MTC networks are most of the time power limited as they are often composed of small low-cost devices. Therefore, their computational abilities and storage capabilities must be taken into account when choosing a security algorithm to secure the communication. In [11], they analyze the cost of security in WSN when using MAC algorithms, encryption algorithm and authentication. They show the results of the multiples experiments with multiples algorithms they have used to compare their power consumption and memory usage. For a data length of 16 bytes, results are in the order of the microjoules.

3 System Design

In this chapter we are going to describe the network scenario that we considered in this project, the requirements of our system and the protocols we designed to comply with them.

3.1 Network Scenario

The scenario we considered in this project is related to the world of agriculture. In particular, we considered a set of cultivated fields, in which a network of sensors operates in order to detect anomalies according to various parameters (temperature, pressure, etc.).

In the designed network we have three types of entities:

- **Sensor**: simple IoT device with low computational capacities. It is a WuR-enabled device divided in two parts: WuX and MR. The sensor itself is mechanical, and wakes up the device when it detect some anomalies. The sensors are grouped in clusters.
- **Cluster-head**: Node to which the sensors in a cluster refer to. These nodes have higher computational capabilities and can communicate with each other. They are WuR-enabled devices as well (WuX+MR). In this report we refer to it as CH.
- **Base station**: Cellular station that helps CHs in communication. There is only one base station that can cover all the territory taken in consideration for this scenario. In this report we refer to it as BS.

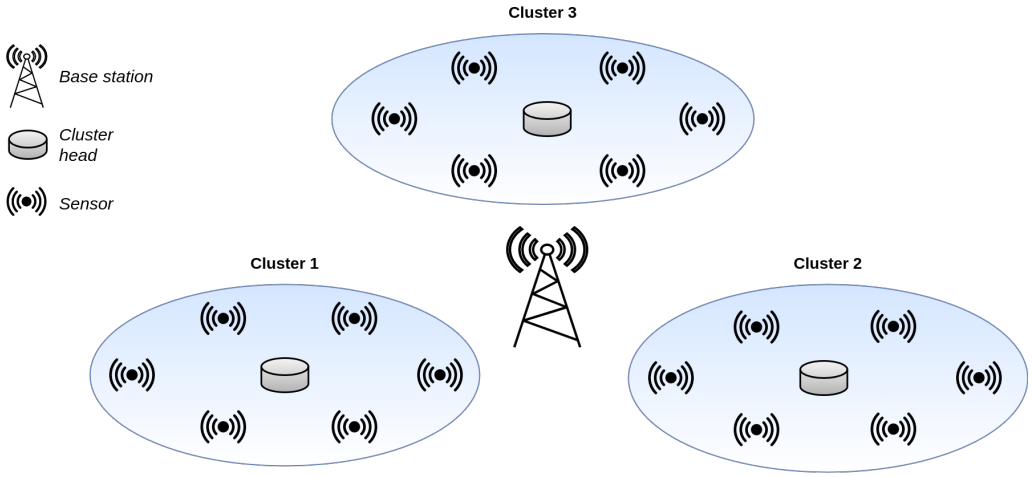

Figure 1: System Topology

Normally the sensors in every cluster sense the surrounding environment periodically until an anomaly is detected. When this happens the sensor wakes up the CH through a WuC to communicate the possible danger. This WuR communication must be secured in order to avoid DoS attacks and therefore keep the power consumption of the CH low. We have implemented a protocol to protect against this kind of attacks.

The CH evaluates sensor reports, and if the temperature (or other parameter) reported t is higher than a predefined threshold T, then a problem is confirmed. This analysis by the CH is

performed to avoid going into alert status in case of false positives in the reports: for example, a sensor might report a single time high temperature caused by a random but non-threatening event. Then the CH tells the sensors to intensify the frequency of detection, in order to receive more information, necessary to evaluate the possible problem.

If the reports are classified as a problem, then the CH will contact the CHs of neighboring clusters to report the presence of the problem. This is done to check if the neighbors are experiencing the same problems, and to let other CHs know that the same problem might occur to them, and thus allow them to intensify their detections. Communication between CHs is done through D2D. Specifically, the first communication takes place with the support of the BS while subsequent communication takes place directly between CHs without help from the BS (Figure 2). This un-assisted communication is established using a resumption mechanism based on key derivation.

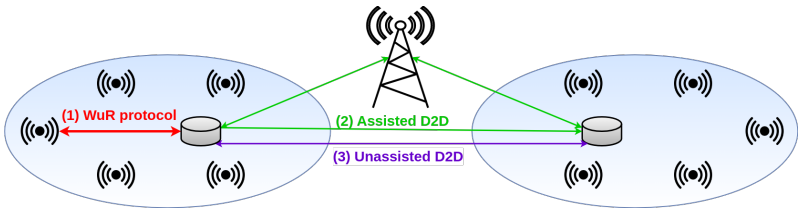

Figure 2: Communications Overview

The protocols adopted for these communications are explained in details in the next chapter. At the end of the communications, the data collected are forwarded to the BS that act as a sink.

3.2 Protocol Design

In this project we designed a security protocol, that can be divided in 4 phases. The first phase concerns communication between sensor and CH. Then in the second phase the CH wakes up all sensors in the cluster to signal the emergency. In the third, the CH initiates D2D communication with neighboring CHs. Finally in the fourth a D2D communication between CHs takes place without help from the BS.

Before describing them in-depth we present the requirements of our system.

3.2.1 System Requirements

We have two types of requirements in our system: performance requirements and security requirements, explored in-depth in Section 5 and Section 6 respectively.

Security requirements:

- WuR must be resistant to DoS attacks based on WuC replaying.
- Each protocol must be resistant to identity spoofing and message tampering.
- Mechanism for connection resumption between CHs must be secure

Performance requirements:

- Computations needed to add security to WuR communications should not add too much overhead
- The un-assisted communication must be much faster than assisted communication, otherwise it is pointless

3.2.2 Phase 1

During phase 1 the sensor that detects an anomaly wakes up the CH and then report the temperature observed. Normal operation of WuR communications involves sending a WuC which usually consist of a predefined code, identical for every wake up, such as the address of the target node. This is particularly prone to replay attacks, since the code used for WuCs is always the same.

Before presenting the actual protocol, it is necessary to explain the assumptions we made regarding the pre-shared secrets used in it. After every sensor is registered to the respective CH and a first handshake is happened between the two, 3 secrets are established:

- *Wu_key*: 16 bytes key used to compute the Wake-up tokens
- *Enc_key*: 16 bytes key used to encrypt
- *Secret_seq*: secret sequence of 8 bit used in the acknowledgments computation

Moreover we must specify that in the actual implementation along with every encrypted message are sent also the 16-byte message authentication code (MAC) and the nonce used for the encryption. Here are omitted for the purpose of simplification. A deeper explanation about this is provided in Subsection 4.2.

The first part of the designed protocol is really simple, because of the limited capabilities of the devices under consideration, but at the same time effective against the previously mentioned attack. The sequence diagram of Figure 3 present the messages exchanged in this phase.

Let us analyze it message by message:

1. **Wake-up Token**: in order to avoid DoS attacks we adopt unique unpredictable tokens to perform the WuC. Every token consist of 2 bytes and is generated in this way:

   - First token:
   $$T_1 = truncate_2(h(wu\_key||node\_address)) \tag{1}$$
   we compute the hash of the wu_key concatenated with the address of the node to wake up, and then we take only the first 2 bytes of the result (through truncation).
   - Next tokens:
   $$T_n = truncate_2(h(T_{n-1}||temperature)) \tag{2}$$
   we compute the hash of the previous token concatenated with the temperature sent by the sensor, and then we take only the first 2 bytes of the result (through truncation).

   In the previous formulas and in all the following ones "h" stands for a secure hash function, in our case we chose SHA-256.

   The CH that receives the token must have a way to figure out what sensor sent the token to establish a proper communication, but in the token is not present any information about the sender. To solve this problem we implemented a table in the CH that contains the updated

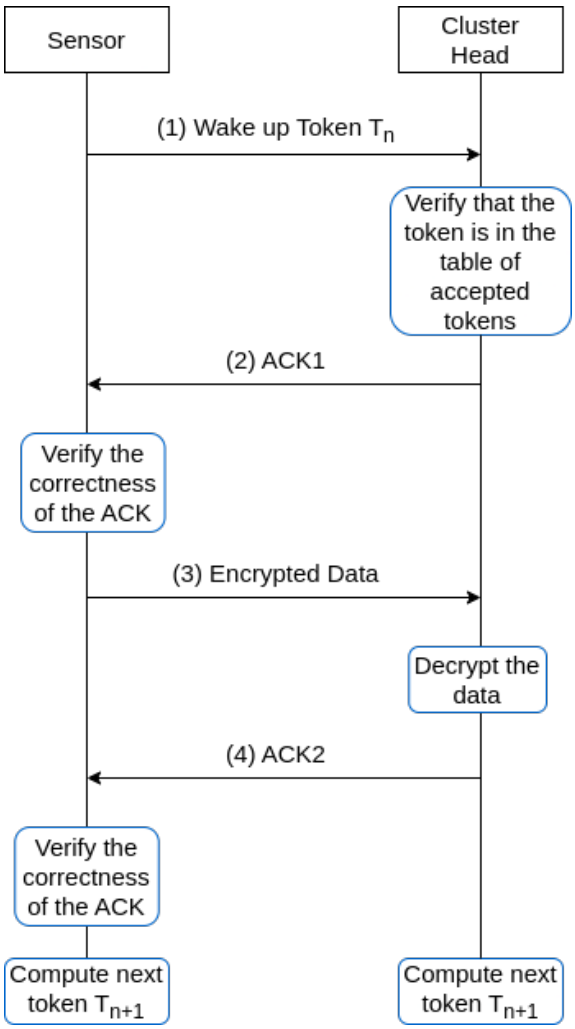

Figure 3: Phase 1 of the protocol

tokens of every sensor, associated with the ID of the sensor. In this way when the wake-up receiver of the CH receives a token it will compare it with the ones in the table. Only if there is a match the MR of the CH is waken up and the protocol can continue.

2. **ACK1**: This message represent the acknowledgment (sent with MR) that confirms that the token has been received. The ACK is computed in this way:

$$ACK = truncate_2(h(T_n||secret_seq)) \tag{3}$$

We compute the hash of the token concatenated to the secret sequence and then we take the first 2 bytes. Computing the ACK with this method, we are sure that only the CH, who knows the secret_seq, can send it correctly. After the ACK has been sent we have 3 possible scenarios:

- The sensor waits for the ACK, but the message has been lost. After a certain amount of time it will send the token again to the CH and will start to wait once more. After the token has been sent for three times, but no response is received, the sensor close the connection.
- The sensor receives the ACK and computes the same quantity (Equation 3) to check its correctness.
  - If the ACK is correct the protocol can continue.
  - If the ACK is incorrect the connection is closed.

3. **Encrypted data**: The data are encrypted using the block cipher AES-CCM [2] with the key enc_key.

$$data = E(temperature||seq\_num, enc\_key) \tag{4}$$

The data sent by the sensor consist of the temperature detected and a sequence number, used by the CH to compute the next ACK.

4. **ACK2**: The second ACK confirm the successful reception of the sensor data. This message is really simple, it is just the sequence number received in the previous message increased by one:

$$ACK2 = seq\_num + 1 \tag{5}$$

This again ensures that the sender is the CH, since the seq_num is encrypted and only he can decrypt it. After the ACK has been sent the possible scenarios are the same of the first ACK.

Without the mechanism of acknowledgments the main obstacle to the protocol would have been the possibility of message loss and the consequent need for a token's resynchronization. In fact, without the ACK2, if the third message was lost, the sensor would proceed with the calculation of the next token, unlike the CH which would wait and then close the connection keeping the token on his side unchanged. This desynchronization is avoided thanks to the acknowledgement.

3.2.3 Phase 2

In phase 2 the CH decides on the basis of reports that a problem is occurring. It then wakes up all the sensors in the cluster so that they send the detected data more frequently. The protocol used is the same as in Subsection 3.2.2, the only difference being that the data sent in the third message is related to the alert type sent by the CH.

3.2.4 Phase 3

In this phase the CH communicates with a neighbor from another cluster to report the problem detected. The communication is carried out through assisted D2D, using as authentication protocol a variation of the Kerberos[3] protocol. Before talking about the protocol itself, again we need to clarify the pre-shared secrets used. Every CH shares a secret key with the BS, so we have N keys with N CHs. These keys are established when the nodes register to BS and the first handshake is

[2] AES is a block cipher, used in this case with mode CCM. Mode CCM ensures an authenticated encryption, so provides both confidentiality and integrity.

[3] Kerberos is a computer-network authentication protocol that works on the basis of tickets to allow nodes communicating over a non-secure network.

concluded. If we consider the Figure 4 that represents the simple case where a CH has only one neighbor, we can consider that before the beginning of the protocol the BS has established two keys K_A and K_B, with CH_A and CH_B respectively.

In Figure 3 is reported the sequence diagram of the phase 3 of the protocol. Let us break it down.

1. **CH ID**: it represents the 2 bytes id of the CH that starts the communication.
2. **Session Key**: the BS generates a session key and encrypt it with the pre-shared key.
$$Session\_key = E(Sk, K_A) \tag{6}$$
3. **Authenticator**: it is used to authenticate the CH through a timestamp encrypted with the session key.
$$Authenticator = E(timestamp, Sk) \tag{7}$$
The timestamp is verified by the BS, which check if its value is within the last 5 seconds. So the check that need to be passed is: $current\_time - timestamp < 5sec$

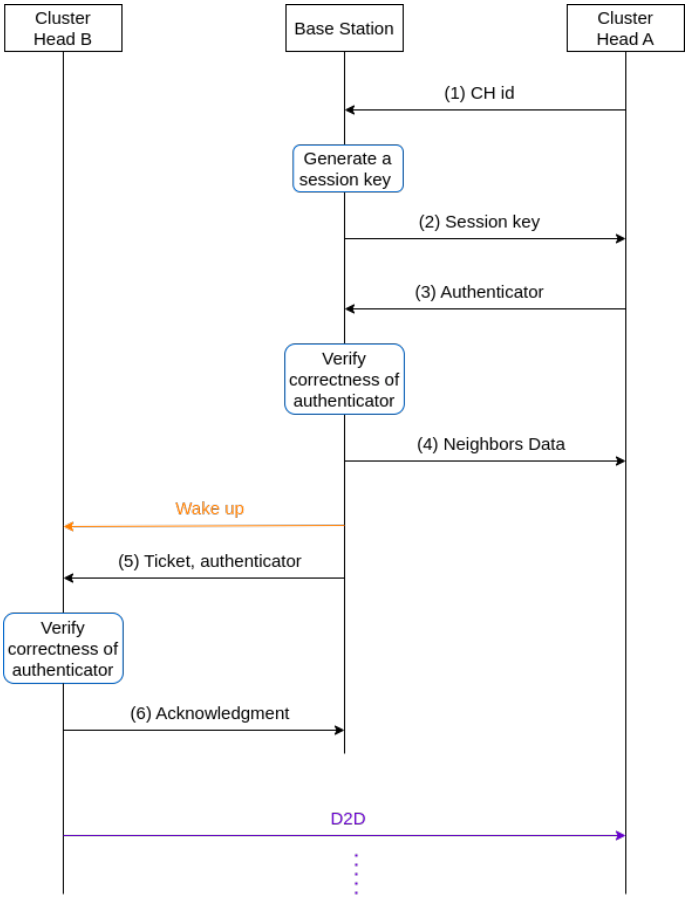

Figure 4: Phase 3 of the protocol

4. **Neighbors Data**: the BS answers with the ids and keys of the CH's neighbors.

$$Data = E(IDs||keys, Sk) \tag{8}$$

5. **Wake-up**: The BS proceeds waking up the neighbors of the CH. This step consist of two messages that are equal to the first two of Subsection 3.2.2.

6. **Ticket and Authenticator**: the ticket contains the identity of the CH which started the D2D (CH A in the Figure 4) and the key to communicate with him. An authenticator created in the same manner as message 4 is sent along with the ticket. It is necessary to authenticate the BS. This information are encrypted with the pre-shared key.

$$\begin{aligned} Ticket &= E("A"||K_{AB}, K_B) \\ Authenticator &= E(timestamp, K_AB) \end{aligned} \tag{9}$$

7. **ACK**: The neighbor CH sends an acknowledgment constructed in this manner to conclude the exchange:

$$ACK = E(timestamp + 1, K_{AB}) \tag{10}$$

Following this stage, the CHs can begin the D2D communication. Furthermore, the BS's aid will not be required in the upcoming communications, as stated in Subsection 3.2.5.

3.2.5 Phase 4

This is the final part of the protocol, in which D2D communication is carried out without assistance from the BS. To do that we use a mechanism similar to TLS resumption, shown in Figure 5. The CH that started the D2D after the last message sends to the other CH, in the encrypted channel, a token ID (similar to a session ticket) and a nonce. The receiver will store this data in a resumption table and use them to create a new encryption key in the event of a new connection. We will only have the option to resume the connection without BS help for a short period because the data in the resumption table is only stored for a limited amount of time.
The key derivation function[4] used to derive the new key is HKDF, which employ HMAC[5] internally.

Let us examine the messages reported in Figure 5:

1. **Token ID, nonce**: the first is simply and identifier used to prove that the two nodes have already communicated, whereas the second is a 16 byte long nonce used to derive the next key. Both the values are encrypted using the key K_{AB}.

$$E(tokenID||nonce, K_{AB}) \tag{11}$$

   The receiver stores these values in a "resumption table". After this message the D2D communication is terminated.

2. **Token ID**: this message is used to resume the communication, without passing through the BS. after determining whether the ID is present in the resumption table, the receiver produces a key using the nonce associated with the ID as the input of HKDF. The other CH generates the same key and then the new D2D can begin.

[4] A cryptographic algorithm known as a "Key Derivation Function," or KDF, generates one or more secret keys from a secret value.

[5] An HMAC is a particular kind of message authentication code (MAC) that uses a cryptographic hash function and a private cryptographic key.

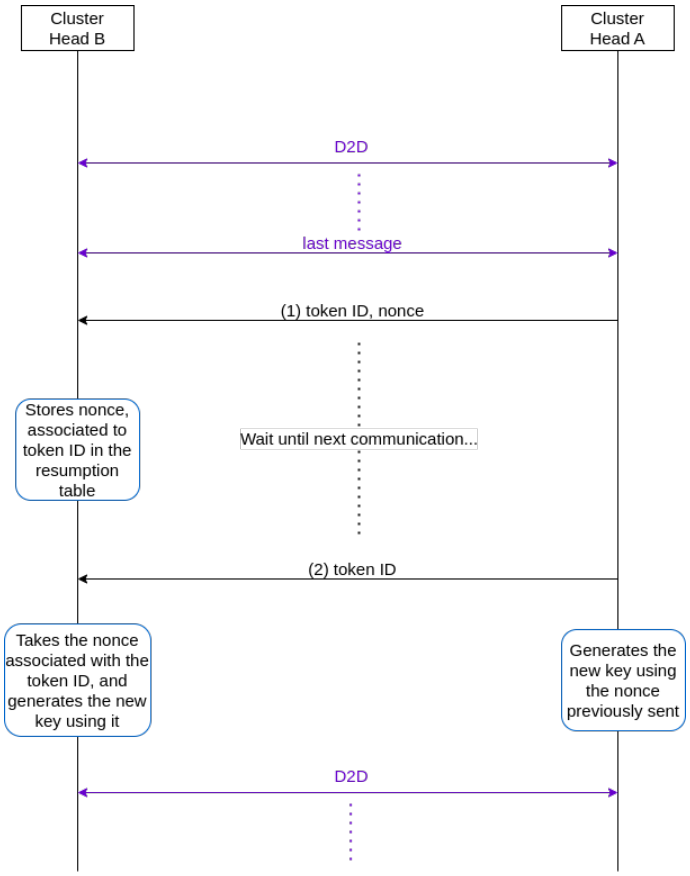

Figure 5: Phase 4 of the protocol

4 Implementation

In this section we are going to explain how we implemented the protocols described in Section 3 and what tools we used. The full implementation of the code can be found on GitHub

4.1 Implementation Tools

For the implementation we decided to use Python 3.8.10, an high-level, general-purpose programming language. We leveraged many different Python libraries:

- **socket**: Provides access to the BSD socket interface. We used sockets to simulate communications without having an actual network. Every message is sent to the loopback address of our PC, to a different port, thanks to sockets.
- **pycrypto**: a collection of different encryption methods (AES, DES, RSA, ElGamal, etc.) as well as secure hash functions (like SHA256 and RIPEMD160). We used this module for every operation that involves encryption.
- **hashlib**: Implements a standard interface for numerous secure hash and message digest techniques. Included are the secure hash algorithms SHA1, SHA224, SHA256, SHA384, and SHA512. We performed all the hashing operations with this library.
- **random**: Implements pseudo-random number generators for various distributions. In particular in our implementation we used the function randint which return a random integer N such that $a <= N <= b$.
- **threading**: Provides functions to implement threading. In the project is used to handle multiple connections to the same device.
- **os**: Provides a portable way of using operating system dependent functionality. In particular in our implementation we used the function urandom which return a bytestring of random bytes.

4.2 Implementation Overview

Before describing the actual implementation we present, in Figure 6, a flow chart that represents the execution flow of the program. The left part of the flow chart refers to the phase 1 and 2 of the protocol whereas the right part refers to the phase 3 and 4.

Our implementation is divided in 4 python files. Three of them represents the entities that we explained in Subsection 3.1: bs.py, ch.py, sensor.py. The last file is my_utils.py and it contains some helper functions, called in the other files.

Now we are going to explain some implementation details that are common to every file.

- **Sockets Communication**: The exchange of messages among entities happens through sockets. In the Listing 1 and 2 we can see how a device can wait for a connection and how it connects to another one. After the connection has been established the devices can send and receive messages to each other through the socket.
- **Encryption**: As reported in Subsection 4.1 to perform encryptions we use the library *pycrypto*. The algorithm adopted is AES CCM that provide authenticated encryption.

  Let us explain it:

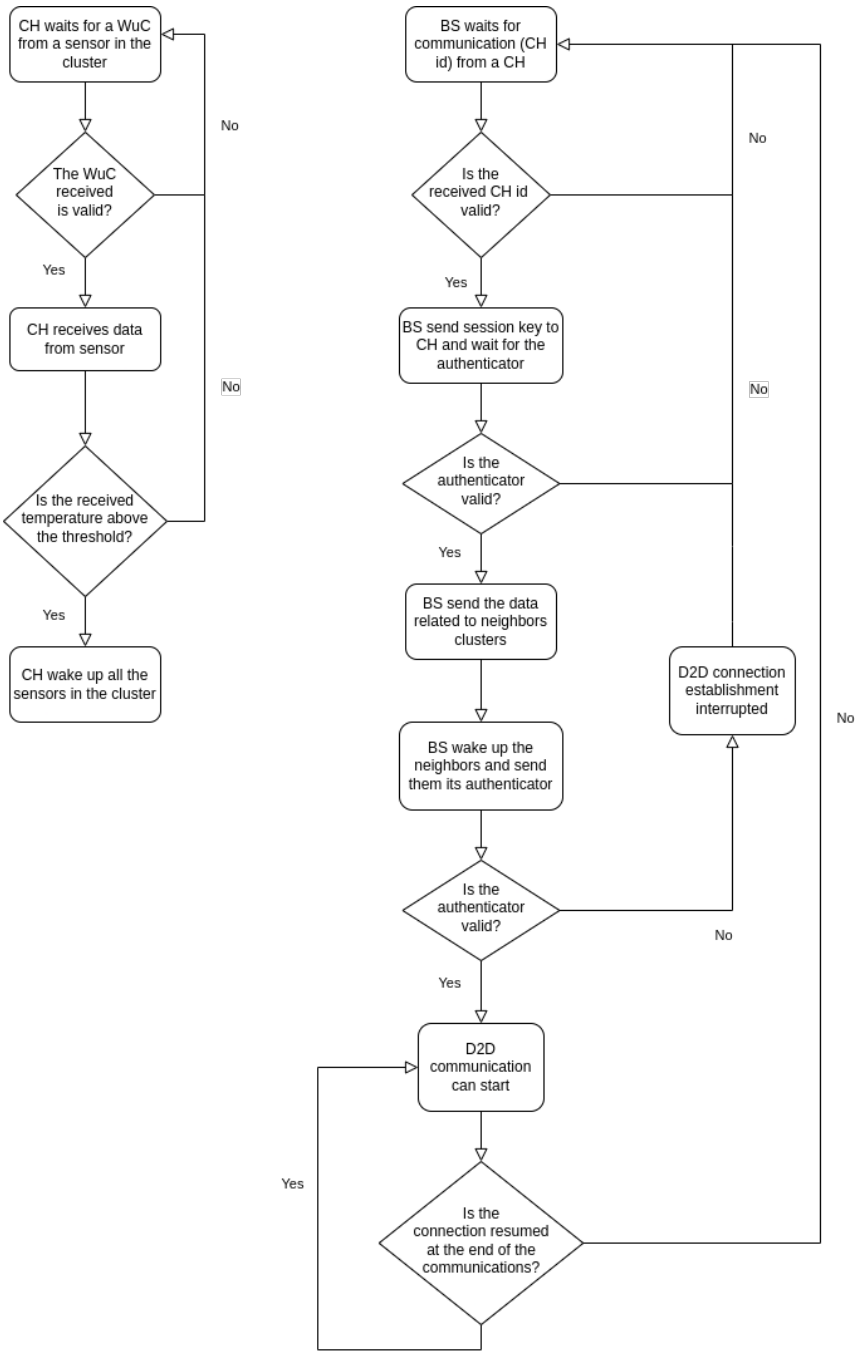

Figure 6: Flow chart of the program. On the left phase 1 and 2, on the right phase 3 and 4

```
1     class Device():
2         def __init__(self) -> None:
3             self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4             self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
5             self.socket .bind((LOCALHOST, OWN_PORT))
6             print("\nCluster head started\nWaiting for a wake up call..")
7
8
9         def start(self):
10            while True:
11                self.socket.listen(1)
12                sensor_sock, sensor_address = self.socket.accept()
```

Listing 1: Socket waiting for a connection

```
1     # connecting to BS
2     ssocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
3     ssocket.connect((LOCALHOST, BTS_PORT))
```

Listing 2: Connection to a socket

- rows 1-2: define simple constants. The *BLOCK_LEN* is the length of the tag, i.e. the message authentication code(MAC)[6].
- rows 4-13: The function *reset_cipher* is used before every encryption and decryption. If we are encrypting we pass to the function only the key as parameter. Then we create a AES cipher in CCM mode, that uses the key and a randomly generated (with *os.urandom()*) nonce[7]. At the end the cipher is returned along with the nonce. We send back the nonce because when we send the encrypted data we need also to send the nonce we used to enable the receiver to decrypt. If we want to decrypt we need to pass a nonce to the function besides the key. In this case the cipher will be generated using the passed nonce and then returned.
- rows 16-18: the actual encryption happen here. We utilize the *encrypt_and_digest* function to encrypt the data and create a message authentication code.
- rows 21-24: for the decryption we use the function *decrypt_and_verify*, which after the decryption verifies the MAC. If the MAC is not correct it throws an exception and the program is stopped.

- **Hashing**: The hashing operations are performed with the hashlib library. The hasher function is firstly created, than its input is updated with the message to hash and at the end the digest is produced (Listing 4).

[6] A MAC provides integrity to the data. The receiver can check it to verify that the received data has not been modified, during the transmission

[7] A nonce is a value, usually random, used by ciphers along with the key

```
    BLOCK_LEN = 16
    NONCE_LEN = 11

    def reset_cipher(key, nonce=None):
    # We are encrypting, we need a new nonce
    if nonce == None:
        new_nonce = os.urandom(NONCE_LEN)
        cipher = AES.new(key, AES.MODE_CCM, new_nonce)
        return cipher, new_nonce
    # We are decrypting, we need to use the received nonce
    else:
        cipher = AES.new(key, AES.MODE_CCM, nonce)
        return cipher

    # Encryption
    s_cipher, nonce = reset_cipher(key)
    msg, mac = s_cipher.encrypt_and_digest(data)
    send_value(server.csocket, msg+mac+nonce)

    # Decryption
    data = recv_value(server.csocket, BLOCK_LEN*2+NONCE_LEN)
    data, nonce = data[:-NONCE_LEN], data[-NONCE_LEN:]
    s_cipher = reset_cipher(s_key, nonce)
    msg = s_cipher.decrypt_and_verify(data[:BLOCK_LEN], data[BLOCK_LEN:])

```

Listing 3: Example of encryption and decryption

```
    hash_fun = hashlib.sha256()
    # first token is the first 16 bits (2 bytes) of sha256(key||cluster__head_address)
    hash_fun.update(token_key+addr)
    # taking the first 16 bits
    token = hash_fun.digest()[:2]
```

Listing 4: Hashing example. In particular this is the creation of a Wake-up token

4.3 Implementation of IoT Devices

The IoT sensors are implemented in the sensor.py file. Every sensor is represented the class Device, which has two inner classes Main_Radio and WakeUp_Radio. This is simply a logic separation designed to represent in the code the actual division in two parts of the real WuR-enabled devices. The main function in this file is reported in Listing 5:

As we can see every message of the protocol is implemented as a single function in order to make the code more readable and to comply with the separation of concerns principle. We used

```
1
2     def notify_ch(self):
3         try:
4             self.sock.connect((SERVER, PORT))
5         except:
6             pass
7         self.wur.wakeup_ch()
8         self.mr.recv_ack_of_wuc(self.wur.token)
9         last_msg, seq_num = self.mr.send_notification()
10        self.mr.recv_ack_of_msg(seq_num)
11        token = self.mr.update_token(last_msg)
12        self.wur.token = token
13        self.mr.nonce = self.mr.reset_cipher()
```

Listing 5: Sensor main function

the same approach in every file.

In Listing 6 is showed the function where the temperature is sent to CH. The temperature and sequence number are 1 and 2 bytes respectively and both are taken random with *randint*. The range of the sequence numbers goes from 0 to $2^{16} - 1$ so it is difficult to have the same sequence number multiple times.

```
1
2     def send_notification(self):
3             temp = int.to_bytes(randint(0, 40), 1, 'big')
4             self.cipher, nonce = reset_cipher(self.enc_key)
5             seq_num = int.to_bytes(randint(0, 2**16-1), 2, 'big')
6             enc, tag = self.cipher.encrypt_and_digest(last_msg)
7             print('sending notification...')
8             send_value(self.socket, enc+tag+nonce)
9             return temp, seq_num
```

Listing 6: Sensor send_notification function

4.4 Implementation of Cluster-Head

In the CH implementation we can find the same logical division for Main_Radio and WakeUp_Radio. This is the biggest file, more than 500 lines of code, so we are going to present only the important features implemented here.

Firstly the CH is capable of accepting multiple connections from different sensors thanks to threading. Every time a new sensor wakes up the CH a new thread is created (Listing 7).

Since the CH can be waken up by different sensors we need a way to store a Wake-up token for each sensor. We implemented this with a python dictionary, because for this data structure the

```
1
2   class Device():
3       def __init__(self) -> None:
4           self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5           self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
6           self.socket .bind((LOCALHOST, OWN_PORT))
7           print("\nCluster head started\nWaiting for a wake up call..")
8
9       def start(self):
10          while True:
11              self.socket.listen(1)
12              sensor_sock, sensor_address = self.socket.accept()
13              self.mr = self.Main_Radio(sensor_sock)
14              self.wur = self.WakeUp_Radio(sensor_sock,   self.mr.token_dict)
15              newthread = SensorThread(self, sensor_address, sensor_sock)
16              newthread.start()
17
18   class SensorThread(threading.Thread):
19      def __init__(self, dev, sensor_address, sensor_socket):
20          threading.Thread.__init__(self)
21          self.socket = sensor_socket
22          self.dev = dev
23          self.sensor_address = sensor_address
24
25      def run(self):
26          # code executed in thread
```

Listing 7: Threading mechanism to handle multiple connections to CH

complexity of the search operation in the worst case (very rare) it is O(n)[8], but the average case time complexity is O(1).

In Listing 8 we present the main function of this file.

The CH wait a wake up token keeping the main radio asleep. When it receives a token it checks if it has been sent by the BS or by a sensor to understand if it has been woken up in phase 1 or phase 3. Then a function for each part of the protocols is executed.

4.5 Implementation of Base Station

The BS implementation is very simple. It can support multiple connections (form CHs) as well. A table with all the data related to the CH's neighbors is kept by the BS and implemented as a python dictionary. Every element in the dictionary has the form showed in Listing 9: every CH id is associated to its key (used to communicate with the BS) and the list of its neighbors.

It is also important to show how the BS carry out the authenticator validity check after the third message of phase 3 of the protocol. It uses the function in Listing 10.

[8]Big O notation describes the complexity of a code using algebraic terms.

```
def run(self):
    print("Connection from : ", self.sensor_address)
    token, caller_id = self.dev.wait_token()
    if caller_id != 'base_station':
        self.dev.wu_protocol(token, caller_id)
        for _ in range(CALLS-1):
            token, caller_id = self.dev.wait_token()
            self.dev.wu_protocol(token, caller_id)
        print('Connection from : ', self.sensor_address, ' closed')
        id, d2d_key = self.dev.kerberos_protocol()
        print('Communication to BS closed, waiting for D2D...')
        sock = self.dev.d2d(id, d2d_key)
        d2d_key = self.dev.unassisted_d2d(sock, d2d_key)
    else:
        print('Communication to BS closed, starting D2D...')
        self.dev.start_d2d(token)
```

Listing 8: Main function in CH

```
NODE_DICT = {b'81': [b'0123456789abcdef', [b'82',b'83',b'84']]
```

Listing 9: Element of the table kept by the BS

```
def is_auth_valid(timestamp):
    TIME_THRESHOLD=5
    current_time = int(time.time())
    print(f'Timestamp check => current time: {current_time}, timestamp: {timestamp}', end='')
    if current_time-timestamp > TIME_THRESHOLD:
        print(' -> INVALID!')
        return False
    else:
        print(' -> VALID!')
        return True
```

Listing 10: Function for authenticator validity check

We take the difference between the current time and the timestamp in the authenticator. If the difference is higher than 5 seconds (reasonable clock skew) the authenticator is considered valid.

The rest on the code can be checked on GitHub, but there are not many interesting feature to report, since the rest of the code consist of encrypting values and then sending them through sockets as described in Section 3.

5 Performance Evaluation

In this chapter we are going to analyze the performance of the protocols from the perspective of time and energy consumption.

5.1 Testing Scenarios

The assessment is divided into three testing scenarios:

- **Testing Scenario 1**: WuR in Phase 1 and 2 with and without security.
- **Testing Scenario 2**: How the execution time varies as the number of entities in the network varies.
- **Testing Scenario 3**: Difference between assisted and un-assisted D2D.

In each Subsection we are going to compute times in this way:

$$t_{total} = t_{computation} + t_{transmission} \tag{12}$$

We use this method because we only have an implementation in code and not actual physical devices so we cannot measures real transmission times. For the same reason we are going to use some parameters taken from [8].

- *WuC duration* = 12.2 ms. We could take this value instead of the data rate because in [8] the size of the WuC is 2 bytes, the same as ours.
- *Main radio data rate* = 250 kbps = 31250 B/s.
- *Power consumption with 3V supply* = 52.2 mW

It is also useful to define now what is the time required to send one byte with the aforementioned *main radio data rate*:

$$T_{1byte} = \frac{1}{mr\_data\_rate} = \frac{1}{31250} = 32\ \mu s \tag{13}$$

So every time we need to compute a transmission time we can compute $T_{1byte} \cdot number\_of\_bytes$

5.2 Testing Results for Scenario 1: WuR with and without security

Given that WuR technology aims to reduce power consumption in the devices in which it is used, it is important that our protocol can provide security without impacting too much on time and power consumption. So, in order to show that our protocol does not impact on the efficiency, we measured the computation and transmission times in our protocol (in Figure 2) to compare it to a simpler and more classical version of WuR protocol that does not involve security (in Figure 7.

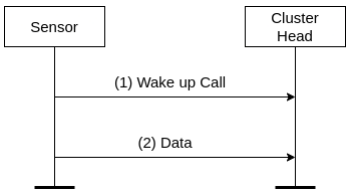

Figure 7: WuR protocol without security

The simpler version mentioned consist of only 2 messages:

- *WuC*: 2 bytes that represent the address of the node to wake up

- *Data*: 3 bytes representing the data to send to the node

The first measurement we made concerns the time required to perform the various cryptographic operations used in our protocol. To get a more precise measurement, we measured the same amount in 100 executions of the protocol to get an average value. The results are shown in the Figure 8

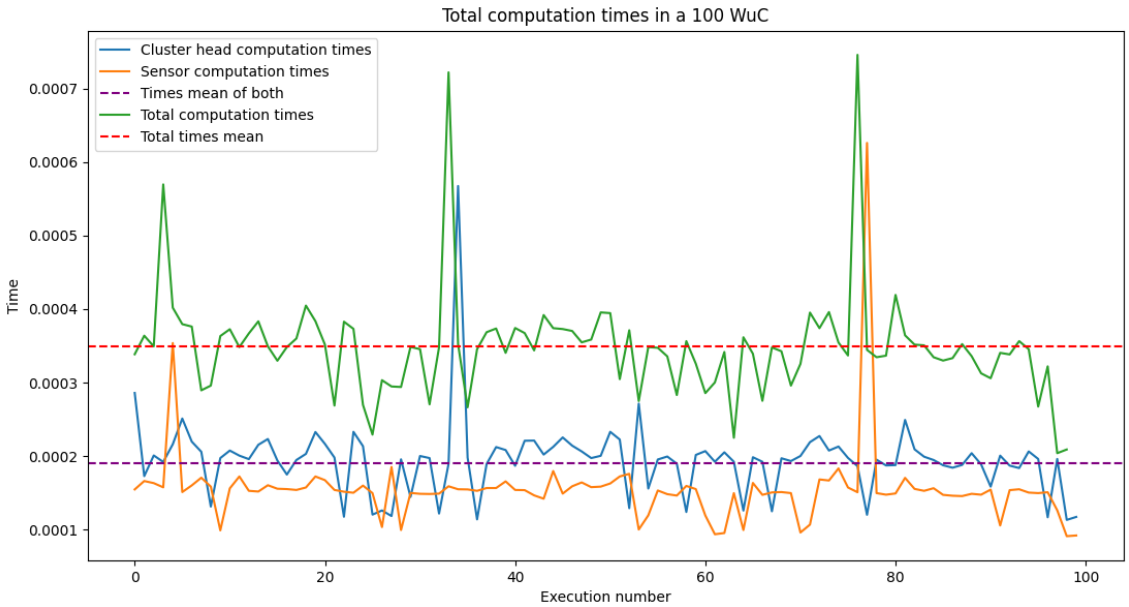

Figure 8: Computation time over 100 executions. In blue and orange we can see the computation times on both sides, CH and sensor respectively. In green we have the total time. The fluctuations and the peaks are given by the internal implementation of the python cryptographic packages that have been used.

The average total computation time is about 0.35 ms. To get the total execution time, we need to add the time required for message transmission, which is given by the following formula:

$$\begin{aligned} t_{transmission} =& wuc\_duration + (ack\_size \cdot 2 + data\_size + nonce\_size + tag\_size) \cdot T_{1byte} \\ &= 12.2 \cdot 10^{-3} + (2 \cdot 2 + 3 + 11 + 16) \cdot (32 \cdot 10^{-6}) \\ &= 0.01329\ s = 13.29\ ms \end{aligned} \tag{14}$$

So the total time turns out to be:

$$t_{total} = t_{transmission} + t_{computation} = 13.29 + 0.35 = 13.64\ ms \tag{15}$$

If we consider the protocol in Figure 7 the transmission time, which corresponds also to the total time (no computations necessary), is the same without the acknowledgments, so:

$$\begin{aligned} t'_{total} = t'_{transmission} =& wuc\_duration + \frac{data\_size}{mr\_data\_rate} \\ &= 12.2 \cdot 10^{-3} + 3 \cdot T_{1byte} \\ &= 0.01230\ s = 12.30\ ms \end{aligned} \tag{16}$$

We can see that the difference between the protocol with and without security is $\sim$ 1.3 ms. This value, compared to the total duration of the protocol in both cases, can be considered a good result, indicating that the security features added to the protocol do not decrease too much its efficiency.

We can also compute the energy consumption multiplying the power consumption and the time:

$$\begin{aligned} E &= P \cdot t_{transmission} = 0.69\ mJ \\ E' &= P \cdot t'_{transmission} = 0.64\ mJ \end{aligned} \tag{17}$$

This results are encouraging and the same observations made for the time are also valid for the energy consumption.

5.3 Testing Results for Scenario 2: Varying the number of elements in the network

One set of evaluations that may be relevant given the project scenario concerns the number of elements in the network and how, as the latter varies, protocol execution times varies.

In Subsection 3.2.4 is presented the phase 3 of the protocol, but for ease of exposition the simplified case where a CH has only one neighbor is depicted. In this section we want to analyze how the number of neighbors affects the execution time.

We computed the total time required to compute and send the neighbors data in case of 1, 4, 8, 16, 32, 64, 128 neighbors. All the results are reported in Table 1 and Figure 9.

The transmission times have been calculated according to the following formula:

$$t_{transmission} = T_{1byte} \cdot (ID\_size + key\_size + nonce\_size + tag\_size) \tag{18}$$

Where $ID\_size = 2,\ key\_size = 16,\ nonce\_size = 11,\ tag\_size = 16$.

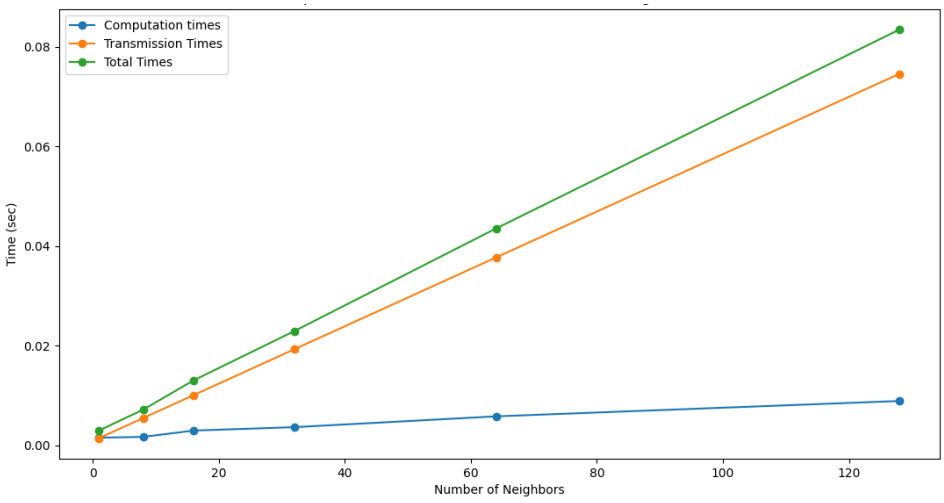

Figure 9: Computation and transmission time related to neighbors data

N neighbors	**Comp. Times** (ms)	**Trans. Times** (ms)	**Tot. Times**(ms)
1	1.54	1.44	2.98
8	1.7	5.47	7.17
16	2.96	10.08	13.04
32	3.65	19.3	22.94
64	5.82	37.73	43.55
128	8.9	74.59	83.49

Table 1: Times related to different number of neighbors

In particular we can notice that the computation times make up a marginal part of the total with respect to the transmission times. The total time increases almost linearly along with the number of neighbors. This makes the system computationally scalable.

The total times with many neighbors (more than 16) are pretty high, however it should be emphasized that in reality we could hardly have many clusters.

5.4 Testing Results for Scenario 3: Difference between assisted and un-assisted D2D

The last measurements we would like to present are about the differences in the time required to complete the D2D connection with and without the help of BS. For the un-assisted connection to make sense, it needs to be much faster than one that takes advantage of BS support. For our calculations we considered part of the assisted connection to be all of phase 3 of the protocol, and for the un-assisted connection to be phase 4, including sending the data necessary for resumption of the connection.

We measured the computation times in both the aforementioned scenarios and we obtained:

- Assisted D2D: 0.0238 s
- Un-assisted D2D: 0.00320 s

We can already notice the there is a big difference between the two. This is due to the fact that during the assisted D2D, multiple cryptographic operations are performed. Regarding transmission times we compute them in the same way as in Subsection 5.2. In this case we consider all the messages sent in phase 3 for assisted D2D and the messages necessary by resumption for un-assisted D2D.

$$\begin{aligned} t_{transmission\_assisted} =& (phase\ 3\ messages\ size) \cdot T_{1byte} \\ &= (2 + 16 + 43 + (N \cdot 18 + 43) + (2 + 2) + 75 + 43) \cdot (32 \cdot 10^{-6}) \\ &= (226 + 18N) \cdot (32 \cdot 10^{-6}) \end{aligned} \tag{19}$$

As mentioned in the previous Subsection the transmission time depends on the number of neighbors (N in the formula above). In fact the fourth message of phase 3 ("neighbors data") consist of the addresses and keys of neighbors CHs of the CH involved in the communication. This dependence represents another disadvantage of assisted communication with respect to the un-assisted one, which is not variable.

If we fix $N = 4$ (reasonable amount of cluster neighbors) we have:

$$\begin{aligned} t_{transmission\_assisted} =& (226 + 72) \cdot (32 \cdot 10^{-6}) \\ &= 0.00954\ s = 9.54\ ms \end{aligned} \tag{20}$$

The transmission time of the un-assisted D2D is:

$$\begin{aligned} t_{transmission\_assisted} =& (token\_id\_size + nonce\_size + token\_id\_size) \cdot T_{1byte} \\ &= 1 + 16 + 1 \cdot (32 \cdot 10^{-6}) \\ &= 0.000576\ s = 0.576\ ms \end{aligned} \tag{21}$$

So in the end the total times are:

- Assisted D2D: 0.03334 s
- Un-assisted D2D: 0.00377 s

The results are satisfying because, as expected, the un-assisted D2D is much faster (~ 10 times) than the assisted D2D.

6 Security Analysis

Security is one of the key objectives of this project, and this section we are going to talk about that. In the first part we present what adversary types we considered during the design phase to provide security for our protocols. In the second part we explain why the described attackers cannot carry out any type of attack, given certain assumptions and the security measures taken.

6.1 Threat Model

We have considered 4 adversary models. Each adversary model describes the capacities of the attacker and the type of attack it could perform. Every attacker we considered has no control on the entities of the network (sensors, CHs, BS) but can eavesdrop all the data sent in the network and can send messages to everyone.

- **Type-I Adversary**: The attacker is capable of replaying WuC and of waking up. Its aim is to perform a DoS attack against the CH by waking him up many times. The attack consists of eavesdropping the WuC and replaying it to waste the CH battery.
- **Type-II Adversary**: The attacker is capable of tampering messages in the communication between a CH and a sensor. Its aim is to modify the message sent by the sensor that contains the temperature, in order to make the CH start the phase 2 of the protocol.
- **Type-III Adversary**: The attacker is capable of performing spoofing attacks, pretending to be a CH during the communication with the BS. The attacker is also able to replay messages sent in old conversations. Its main aim is to recover the keys necessary to communicate with the neighbors of the legitimate CH.
- **Type-IV Adversary**: The attacker is capable of replaying the messages sent for the resumption in the un-assisted D2D. Its aim is to derive the key used for the new connection between two CHs.

6.2 Security Assessment

In our analysis, we rely on the following security assumptions:

- **AES-128 with CCM mode**: we assume that the symmetric key encryption performed with this cipher is secure, so breaking it or bruteforcing the key is computationally unfeasible for an attacker. Moreover we assume that CCM mode provides also integrity to the data.
- **SHA-256 hash function**: we assume that this is a secure hashing algorithm: it is one-way and has strong collision resistance[9].
- **HKDF**: we assume that this key derivation function generates cryptographically strong output key material, impossible to recover without knowing both the two inputs: the old key and a nonce.

In the next part we show that our designed protocol is resistant to the aforementioned adversary types.

[9]It is computationally infeasible to find any two distinct inputs which hash to the same output

- **Resistance to Type-I Adversary**: The adversary can replay token that have already been sent. Therefore it can start a WuR communication with the CH with the objective of carrying out a DoS attack. Let us suppose that the adversary replay the token T_n sent at the n^{th} communication by a sensor to its CH. At the end of the n^{th} communication the CH has computed T_{n+1} as:

$$T_{n+1} = truncate_2(h(T_n||temperature)) \tag{22}$$

So when the CH's WuR receiver receives T_n from the adversary it compares it with the current valid token T_{n+1}, resulting in a negative match. In this case the the MR of the CH is not waked up and the DoS cannot take place.
The adversary is not even able to forge T_{n+1}, because it does not know the temperature used to compute it and it is not able to recover because it is protected by encryption, which we assumed being secure.

$$data = E(temperature||seq\_num, enc\_key) \tag{23}$$

Moreover we also assumed that the hash function (SHA-256) has strong collision resistance, so it is computationally infeasible for the attacker to find a value that has the same hash of $temperature||seq\_num$.

- **Resistance to Type-II Adversary**: The adversary can tamper the messages exchanged between CH and sensors. It tries to modify the third message of phase 1:

$$data = E(temperature||seq\_num, enc\_key) \tag{24}$$

After the modification the CH will decrypt the data obtaining random values for temperature and sequence number[10]. The sequence number is supposed to be random, so would not be suspicious. The decrypted temperature would likely be very high, therefore triggering the CH to start phase 2. Although the attack seems effective, it is prevented by AES-CCM, which provides integrity via a MAC. When receiver checks the MAC of the modified message it obtain a negative match and then stops the protocol.

$$data' = E(M')||MAC(M) \tag{25}$$

The adversary is not able to forge a valid MAC without the enc_key because we assumed that AES CCM is secure.

- **Resistance to Type-III Adversary**: The Type-III adversary can spoof his identity, pretending to be a CH. It can start the protocol with the BS because, as showed in Figure 3, because the communication begins by sending the CH id in plaintext. The attacker can take a valid CH id (which are public) and send it to the BS. Let us say that the adversary pretend to be CH_A. The BS answers with a session key encrypted with the key shared with the legitimate CH:

$$Session\_key = E(Sk, K_A) \tag{26}$$

After this the adversary is supposed to send an authenticator:

$$Authenticator = E(timestamp, Sk) \tag{27}$$

[10] If encrypted data is modified, the result of the decryption consist of random bytes

But It is not able to create this message because it need the session key in order to encrypt and the session key is protected by the encryption with K_a which is known only by CH_A. Furthermore the adversary cannot replay a previously sent authenticator because the session key would be different.

- **Resistance to Type-IV Adversary**: The Type-IV adversary can replay messages sent in the D2D communication. It can try to replay the token ID previously sent by one of the CHs in order to try to resume a connection without knowing the key.
  It cannot succeed in continuing the communication with a CH because it does not know either the nonce or the key necessary to generate the new D2D key.

$$new\_d2d\_key = HKDF(old\_d2d\_key, nonce) \tag{28}$$

7 Conclusions and Future Work

In this last chapter, we are going to conclude by making a summary of all the work that has been done in this report. We are going to discuss what our work has contributed to and then give some future work topics related to our work that could need some more in-depth study.

7.1 Conclusions

In this report, we have presented a WSN scenario related to the world of agriculture. With the WSN composed of WuR enabled devices, we have described a protocol to explain how the devices are communicating with each other. Using Python as a programming language and different Python libraries, we implemented the multiple devices that will take part in the communication. Through testing in different case scenarios, we first demonstrated that security features added to the protocol do not greatly hinder its efficiency. Then we proved that the more CHs are communicating together, the higher the computation time and transmission time will be. After, we showed that un-assisted D2D is ten times faster than assisted D2D. As security being one important aspect of this project, we studied the threat model for our protocol and explained why these attacks could not be carried out.

7.2 Contribution

With this project we provided the following contributions.

- We designed a novel security protocol for WuR that protects against DoS attacks based on wake-up call replaying.
- We designed a secure authentication and key establishment protocol for assisted and un-assisted D2D communications in a WSN scenario.
- We implemented both protocols in Python.
- We evaluated the performances of both protocols obtaining good results from the point of view of time and power consumption.

7.3 Future Work

As our future works, it would be interesting to implement the protocols we devised using physical devices (sensors and CHs) to compare the results obtained in the code with real data. In addition, there are also improvements that could be made to the protocols such as a resynchronization mechanism in case of message loss during communication between CH and sensors or a method to handle multicast collision on the CH-BS and sensors-CH communications. In conclusion, regarding security, a real formal verification of the protocols could be carried out to indisputably prove their validity.

References

[1] Angelo T Capossele, Valerio Cervo, Chiara Petrioli, and Dora Spenza. "Counteracting denial-of-sleep attacks in wake-up-radio-based sensing systems". In: *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE. 2016, pp. 1–9.

[2] Xiaoming Chen, Derrick Wing Kwan Ng, Wei Yu, Erik G Larsson, Naofal Al-Dhahir, and Robert Schober. "Massive access for 5G and beyond". In: *IEEE Journal on Selected Areas in Communications* 39.3 (2020), pp. 615–637.

[3] Fatima Zahra Djiroun and Djamel Djenouri. "MAC protocols with wake-up radio for wireless sensor networks: a review". In: *IEEE Communications Surveys and Tutorials,* vol. 19.1 (2017), pp. 587–618.

[4] Klaus Doppler, Jawad Manssour, Afif Osseiran, and Ming Xiao. "Innovative concepts in peer-to-peer and network coding". In: *Celtic Telecommunication Solutions* 16 (2008), p. 09.

[5] Alessia Elgani, Michele Magno, Francesco Renzini, Luca Perilli, E Franchi Scarselli, Antonio Gnudi, Roberto Canegallo, Giulio Ricotti, and Luca Benini. "Nanowatt wake-up radios: discrete-components and integrated architectures". In: *25th IEEE international conference on electronics, circuits and systems (ICECS)*. IEEE. 2018, pp. 793–796.

[6] Gábor Fodor, Erik Dahlman, Gunnar Mildh, Stefan Parkvall, Norbert Reider, György Miklós, and Zoltán Turányi. "Design aspects of network assisted device-to-device communications". In: *IEEE Communications Magazine* 50.3 (2012), pp. 170–177.

[7] Anders Froytlog, Thomas Foss, Ole Bakker, Geir Jevne, M Arild Haglund, Frank Y Li, Joaquim Oller, and Geoffrey Ye Li. "Ultra-low power wake-up radio for 5G IoT". In: *IEEE Communications Magazine* 57.3 (2019), pp. 111–117.

[8] Debasish Ghose, Luis Tello-Oquendo, Vicent Pla, and Frank Y Li. "On the behavior of synchronous sata sransmission in wake-up radio enabled IoT networks: protocol and absorbing Markov chain based modeling". In: *IEEE Transactions on Wireless Communications* (2022).

[9] Philippe Le-Huy and Sébastien Roy. "Low-power wake-up radio for wireless sensor networks". In: *Mobile Networks and Applications* 15.2 (2010), pp. 226–236.

[10] Mohamed Rawidean Mohd Kassim. "Iot applications in smart agriculture: Issues and challenges". In: *2020 IEEE conference on open systems (ICOS)*. IEEE. 2020, pp. 19–24.

[11] Jongdeog Lee, Krasimira Kapitanova, and Sang H Son. "The price of security in wireless sensor networks". In: *Computer Networks* 54.17 (2010), pp. 2967–2978.

[12] Leonardo Militano, Giuseppe Araniti, Massimo Condoluci, Ivan Farris, and Antonio Iera. "Device-to-device communications for 5G Internet of Things". In: *EAI Endorsed Transactions on Internet of Things* 1.1 (2015), e4–e4.

[13] Heinrich Milosiu, Frank Oehler, Markus Eppel, Dieter Frühsorger, Stephan Lensing, Gralf Popken, and Thomas Thönes. "A 3-μW 868-MHz wake-up receiver with 83 dBm sensitivity and scalable data rate". In: *Proceedings of the ESSCIRC (ESSCIRC)*. IEEE. 2013, pp. 387–390.

[14] Maxime Montoya, Simone Bacles-Min, Anca Molnos, and Jacques JA Fournier. "Sward: a secure wake-up radio against denial-of-service on IoT devices". In: *Proceedings of the 11th ACM conference on security & privacy in wireless and mobile networks*. 2018, pp. 190–195.

[15] Joaquim Oller, Ilker Demirkol, Jordi Casademont, Josep Paradells, Gerd Ulrich Gamm, and Leonhard Reindl. "Has time come to switch from duty-cycled MAC protocols to wake-up radio for wireless sensor networks?" In: *IEEE/ACM Transactions on Networking* 24.2 (2015), pp. 674–687.

[16] Hyunhee Park. "Anti-malicious attack algorithm for low-power wake-up radio protocol". In: *IEEE Access* 8 (2020), pp. 127581–127592.

[17] Rapeepat Ratasuk, Nitin Mangalvedhe, David Bhatoolaul, and Amitava Ghosh. "LTE-M evolution towards 5G massive MTC". In: *2017 IEEE Globecom Workshops (GC Wkshps)*. IEEE. 2017, pp. 1–6.

[18] V Rosello, J Portilla, and T Riesgo. "Ultra low power FPGA-based architecture for wake-up radio in wireless sensor networks". In: *IECON 2011-37th Annual Conference of the IEEE Industrial Electronics Society*. IEEE. 2011, pp. 3826–3831.

[19] Soheil Rostami, Petteri Kela, Kari Leppanen, and Mikko Valkama. "Wake-up radio enabled 5G mobile access: methods, benefits, and challenges". In: *IEEE Communications Magazine* 58.7 (2020), pp. 14–20.

[20] Soheil Rostami, Sandra Lagen, Mário Costa, Mikko Valkama, and Paolo Dini. "Wake-up radio based access in 5G under delay constraints: modeling and optimization". In: *IEEE Transactions on Communications* 68.2 (2019), pp. 1044–1057.

[21] Sebastian L Sampayo, Julien Montavont, Fabien Prégaldiny, and Thomas Noël. "Is wake-up radio the ultimate solution to the latency-energy tradeoff in multi-hop wireless sensor networks?" In: *2018 14th international conference on wireless and mobile computing, networking and communications (WiMob)*. IEEE. 2018, pp. 1–8.

[22] C-H Yu, Olav Tirkkonen, Klaus Doppler, and Cássio Ribeiro. "Power optimization of device-to-device communication underlaying cellular communication". In: *2009 IEEE international conference on communications*. IEEE. 2009, pp. 1–5.

[23] Meisu Zhong, Yongsheng Yang, Haiqing Yao, Xiuwen Fu, Octavia A Dobre, and Octavian Postolache. "5G and IoT: Towards a new era of communications and measurements". In: *IEEE Instrumentation & Measurement Magazine* 22.6 (2019), pp. 18–26.