Hellenic Mediterranean University

SCHOOL OF ENGINEERING
DEPARTMENT OF INFORMATICS ENGINEERING

Evaluation and Certification of Software

for Examining the Integrity of Software on IoT devices

Student :MariolasAdamantios

Supervisor :Kornaros George, Associate Professor

ΕΛΛΗΝΙΚΟ ΜΕΣΟΓΕΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μέτρηση και Πιστοποίηση Λογισμικού για την εξέταση της ακεραιότητας του λογισμικού σε συσκευές IoT

Σπουδαστής : Αδαμάντιος Μαργιόλας
Επιβλέπων : Κορνάρος Γεώργιος, Αναπληρωτής Καθηγητής

Τριμελής Εξεταστική Επιτροπή

Γεώργιος Κορνάρος
Αναπληρωτής Καθηγητής του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών (ΗΜΜΥ) του Ελληνικού Μεσογειακού Πανεπιστημίου (ΕΛ.ΜΕ.ΠΑ)

Σπυρίδων Παναγιωτάκης
Αναπληρωτής Καθηγητής του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών (ΗΜΜΥ) του Ελληνικού Μεσογειακού Πανεπιστημίου (ΕΛ.ΜΕ.ΠΑ)

Παπαδάκης Νικόλαος
Αναπληρωτής Καθηγητής του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών (ΗΜΜΥ) του Ελληνικού Μεσογειακού Πανεπιστημίου (ΕΛ.ΜΕ.ΠΑ)

Declaration of Academic Integrity

I hereby confirm that the present thesis on

_____

is solely my own work and that if any text passages or diagrams from books, papers, the Web or other sources have been copied or in any other way used, all references – including those found in electronic media – have been acknowledged and fully cited.

Acknowledgements

I would like to say a special thank you to my supervisor, Kornaros George for his support, guidance and overall insights in this field. I would also like to thank my family for supporting me during the compilation of this dissertation.

Table of Contents

Σύνοψη στα ελληνικά

Στις μέρες μας, παρατηρούνται μεγάλης κλίμακας κυβερνητικών επιθέσεων στον παγκόσμιο ιστό, όπως επίσης καταστροφές σημαντικής υποδομής πληροφοριών, λόγω επιθέσεων λογισμικού όπως το worm, το botnet και το DDoS σε συσκευές IoT. Η μέτρηση και η πιστοποίηση λογισμικού αποτελούν γενικές μεθόδους για τον εντοπισμό της ακεραιότητας του λογισμικού και των καταστάσεων του κατά την διάρκεια που εκτελείται σε ένα στοιχείο IoT. Στην παρούσα πτυχιακή εργασία θα αναπτυχθούν πρακτικές μέθοδοι για την αξιόπιστη εκτέλεση λογισμικού που βασίζεται σε τεχνικές "ελαφριά" εκτέλεση και εμπιστοσύνη. Επιπλέον, θα εφαρμοστούν μέθοδοι που συνδυάζουν τη δυναμική μέτρηση και την ακεραιότητα ροής ελέγχου με το κλειδί σύνδεσης συσκευής και κρυπτογράφησης της διεύθυνσης κώδικα ή δεδομένων, έτσι ώστε να μπορεί να προστατευτεί η ακεραιότητα του λογισμικού κατά το χρόνο εκτέλεσης στη συσκευή IoT.

Λέξεις κλειδιά: κυβερνητικές επιθέσεις, μέτρηση και πιστοποίηση λογισμικού, κρυπτογράφηση, IoT, "ελαφριά" εκτέλεση και εμπιστοσύνη.

Abstract in English

Nowadays, there are large-scale cyber-attacks on the World Wide Web, as well as destruction of important information infrastructure, due to software attacks such as worm, botnet and DDoS on IoT devices. Software measurement and certification are general methods for identifying the integrity of software and its states while running on an IoT component. In this thesis, practical methods will be developed for the reliable execution of software based on "lightweight" execution and trust techniques. In addition, methods that combine dynamic measurement and control flow integrity with device connection key and code or data address encryption will be implemented so that the integrity of the software at runtime on the IoT device can be protected.

Keywords: cyber-attacks, "lightweight" execution and trust, software measurement and certification, IoT, control flow integrity.

Table of Figures

Tables

Abbreviations

Term	Description
IoT	Internet of Things
NOx (NO, NO_2, N_2O)	Nitrogen deposition that can cause respiratory and cardiovascular sickness
CO	Combustion transport and power generation that can cause headache, functioning of heart and prolong inhalation lead to comma
CO_2	Fossil fuel, cement construction and vehicles that can affect O_2 movement in blood
SO_2	Combustion, power generation that can cause problem in breathing of children, visibility impairment and respiratory disorder

Chapters

Chapter 1. Introduction

1.1 Problem

IoT devices are gradually increasing the interest of the wider consumer public. Several studies predict an almost exponential increase in the number of IoT devices, which reached around 20 - 30 billion by 2020. On the other hand, a major challenge affecting consumer and industry decisions are the security and privacy issues. These problems are the main factor that slows down interoperability and development the domain of Internet of Things.

Reports have been published from time to time documenting vulnerabilities in either smart home automation or in wider IoT networks. It is important to identifythe vulnerabilities with automated tools and to be investigated remotely, especially via Internet.

1.2 The objective of the Thesis

The aim of the thesis is to present a secure framework of an IoT network; analyzing the aspects of the ecosystem, the main vulnerabilities and the real risks. More specifically, this thesis proposes an application that may contribute to the security issues that may derive in IoT devices and more general in microcontrollers. Therefore, it is a means of preventing attacks either from malicious software or from malicious users with the aim of falsifying data.

Its purpose is to ensure the integrity of the software. More specifically, it enables two or more threads to be able to communicate with each other, and to be able to recognize and determine the authenticity of the message from the sender (in our case regarding temperatures retrieved from sensor), as well as reply back to it or execute certain commands.

Let us think about what could happen if the sensor we have placed in a factory, in a house, or in general in a device, sent incorrect data (e.g. false temperatures). The results could be fatal and in some cases even dangerous.

For the application (proof of concept) of such methodology, we will follow a practical approach using some open source tools to detect network-explorable vulnerabilities TCP/IP. The proposed methodology allows manufacturers of IoT devices to detect vulnerabilities in cyber-attacks with low cost, as well as, companies or even individuals can be benefited from this application as they can ensure the correct/proper operation of both the applications and the machines themselves where the sensors are used.

The above tools are installed on Linux OS and any conclusions/measurements help us to evaluate and verify the proposed methodology.

1.3 Contribution

In this thesis, we carry out an extensive bibliographic research on the security issues of IoT. We have collected and listed in detail related material of IoT terminology, as well as security issues (IoT attacks) and solutions (chapter 2); taking into account existing work and solutions. We reviewed a number of literature sources that were sourced from conference proceedings, publications, surveys, scientific articles, books and Internet sources

1.4 Structure

Abstract in Greek and in English are presented in the beginning of this Thesis.

Chapter 1 is the introduction of the thesis. More specifically, we are referring to the problem that this thesis deals with, its purpose, its contribution as well as its structure.

In Chapter 2, entitled as “Development environment & board”, we are talking about the installation, development environment and the equipment that have been used in the context of the work of this thesis.

In Chapter 3, entitled as “IoT & Security Issues”, a literature review of IoT and the security issues detected in IoT domain is conducted. Particularly, we first present the terminology needed; regarding IoT (terminology, applications, etc.). We also list in details security issues that are important for IoT concept (challenges, security standards used, solutions, etc.).

Moving on Chapter 4, entitled as “Zephyr OS and SHA-2”, we present what is zephyr and how zephyr contributes to this work, as well as SHA-2 concepts.

In Chapter 5, entitled as “Implementation & code in C”, we exhibit a detailed overview and description of the programming code that accompanies this thesis, while in Chapter 6, entitled as “Measurements & results”, we present some of the experimentation that has been conducted as well as results that we get by executing this code; using the board and the sensor as described in Chapter 2.

The last chapter, Chapter 7, entitled as “Conclusion & Future Work”, concludes the work made in this Thesis; by presenting the conclusions of the work and some future work.

Chapter 2.Development environment & board

2.1 Development environment

For the implementation of the application, the following IDE has been used:
Microsoft visual studio [1] with the following extensions: C++ syntax, C/C++, C/C++ Extension Pack, codelldb. VS code is mainly chosen in order the writing of the code to be easier, by using the graphical element that it provides.

Moreover, an installation of zephyr os is a pre-requisite on the system and was done in our case in Linux peppermint (Debian based), in order our solution to be compiled. Specifically, the following steps are followed via terminal:

- Open the *zephyrproject/zephyr folder*.
- Run the command

west build -b stm32f469i-disco samples/adamantios_demos/final/ --pristine.

2.2 Hardware equipment

2.2.1 ST Microelectronics Company

ST Microelectronics [2] is a company that is established in 1994 and deals with the design of products that aim to provide a friendly ecosystem for humans. In this Thesis, we make use of the STMicro Discovery Board of this company named as stm32f469i-disco.

2.2.2 Board features

Figure 1: stm32f469i-disco board

The stm32f469i-disco board provides users with a simple application development environment and features [3] as follows:

- STM32F469NIH6 microcontroller featuring 2 Mbytes of Flash memory and 324 Kbytes of RAM in BGA216 package
- On-board ST-LINK/V2-1 SWD debugger, supporting USB remuneration capability:
    - Mbed-enabled (mbed.org)
    - USB functions: USB virtual COM port, mass storage, debug port
- 4 inches 800x480 pixel TFT colour LCD with MIPI DSI interface and capacitive touch screen
- SAI Audio DAC, with a stereo headphone output jack
- 3 MEMS microphones
- MicroSD card connector
- I2C extension connector
- 4Mx32bit SDRAM
- 128-Mbit Quad-SPI NOR Flash
- Reset and wake-up buttons
- 4 color user LEDs
- USB OTG FS with Micro-AB connector
- Three power supply options:
- Expansion connectors and Arduino™ UNO V3 connectors

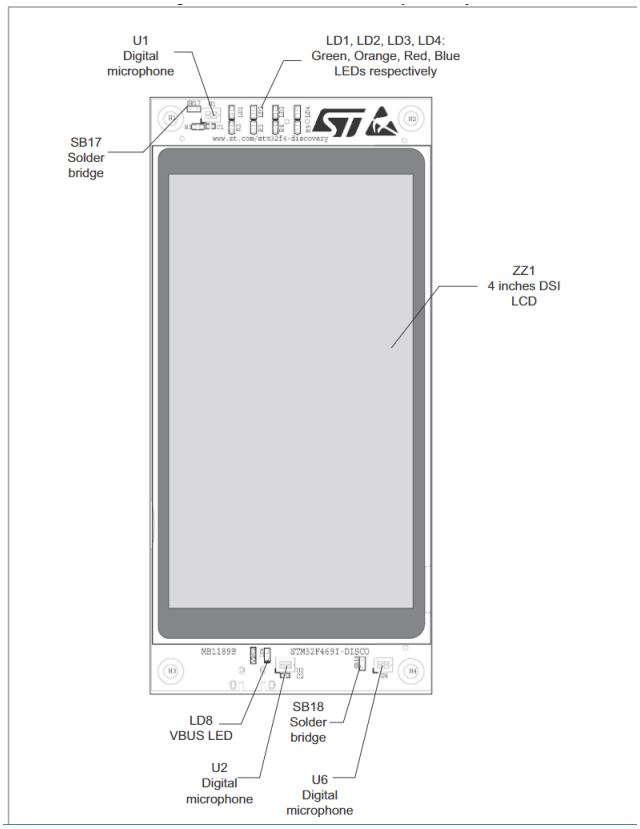

Figure 2: Top side layout

As we can see, there is also a touch screen that can be used for the development and testing of applications.

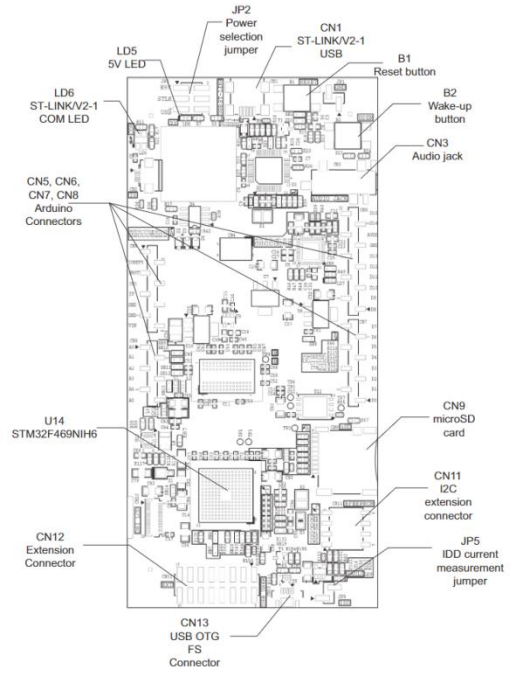

Figure 3: Bottom side layout

These applications can provide a GUI for the user to interact with, as well as there is a possibility to include many types of sensors and a wide variety of connectivity features with other devices (e.g. Arduino connectors for external sensor connectivity).

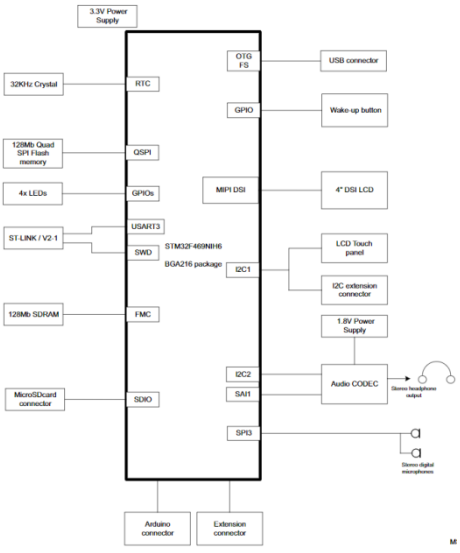

Figure 4: Hardware block diagram

2.2.3 BME680 gas sensor

The BME680 gas sensor is used in the context of this Thesis as well. It can integrate high-linearity and high-accuracy since it measures gas on the air, atmospheric pressure, humidity and temperature; using the latest sensors on the market.

It is especially developed for mobile applications (size 3.0 x 3.0 x 0.93 mm^3)that have low power consumption. BME680 can guarantee optimization on consumption, long-term stability and high EMC robustness.

The reason that BME680 is used in this Thesis is because the gas sensor within the BME680 can detect a broad range of gases such as volatile organic compounds (VOC); allow the users to measure air quality for personal wellbeing.

More specifically, in this thesis as we have already mentioned, the equipment used is the board STM32F469i with an ARM® Cortex®-M4 processor. On the board, we have placed this external Bosch sensor - BME680, to get the measurements of the temperatures (10 Celsius values per measurement).

Chapter 3.IoT, Security Issuesand Related Work

3.1 What is IoT?

The Internet of Things (IoT), also called Internet of Everything (IoE), is a new technological achievement, envisioned as the global network of computing machines and electronic devices that can interact with each other. IoT is recognized as one of the most important areas of modern and future technology and attracts great attention from a wide range of industries.

IoT is a collection of many interconnected objects, services, people and devices, which can communicate, share data and information, in order to achieve a common goal in various fields and applications. IoT devices support "Identity Management" identified in homogeneous and heterogeneous devices. Similarly, an IoT domain can be defined by an address IP. However, each device within this range can have its own address IP.

The term IoT was first invented by Kevin Ashton in 1999 in the context of supply chain management [4]. However, during the latter decade, the definition has covered a wider range of applications such as healthcare, utilities, energy distribution, etc. [5]. Although the definition of "Things" has changed as technology has evolved, the main goal of providing information about the machines without human assistance intervention remains the same.

A radical evolution of the Internet in one Network of interconnected objects that not only collects information from the environment (detection) and interacts with the physical world (activation / command / control), but also makes use of existing Internet standards to provide services for the transfer of information, applications and communications. Some examples include enabled devices with open wireless technology such as Bluetooth, radio frequency identification (RFID), Wi-Fi and telephone data services, as well as embedded sensor nodes and activator.

The Internet revolution has led to the interconnection between objects to create an intelligent environment. From 2011 the number of interconnected devices on the planet exceeds the number of population. In 2019, there were 9 billion interconnected devices and this number reached 24 billion devices by
2020. According to the GSMA, regarding the healthcare, automotive, and consumer electronics sectors, the aforementioned amounts translated to $1.3 trillion revenue opportunities.

3.2 IoT Terminology

Some of the definitions given according to different sources are:

- **IEEE** – 2015 [5]: “A global network of interconnected objects withunique addressing based on basic communication protocols".

- **Wikipedia** [6]: “The Internet of Things or Internet of Things(English: Internet of things) is the mass communication network appliances, household appliances, cars as well as any object that integrates electronics, software, sensors and connectivity into network to allow connection and data exchange. simpler, the philosophy of IoT is to connect all electronic devices between them (local area network) or connected to the internet (world wide web)"
- **Tutorials Point** - Simply Easy Learning [7]: “IoT (Internet of Things) is an advanced automation and analysis system that takes advantage of networking, sensing, big data and artificial intelligence to provide integrated systems for a product or a service. These systems allow greater transparency, control and performance when applied to any industry or system.”

- **IEEE 2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies** [8]: "Group of structures which interconnect the connected objects and allow their management, the data mining and access to the data they generate.

3.3 Applications in IoT

Internet technologies are expected to drive innovation in a number of key industrial sectors, such as health, factory automation / smart manufacturing, food production and distribution, environmental monitoring, buildings, living environments, energy, smart cities, etc. [11].

In more detail, the application value of IoT could be analyzed in some areas, as shown below [12]:

- **In the living environment**

IoT domain can bring benefits to people's daily living environments. Citizens work and spend many hours of the day in these environments, e.g. home, work, sports, entertainment, social activities, etc. People exercise and be engaged in a wide variety of activities that can be designed for activities such as health and fitness, work from home experiences and experiences from energy consumption.

- **In Agriculture and Nutrition**

IoT technology enables monitoring and control of plants and animal products throughout the farm-to-plate cycle. The challenge in the future is to design architectures and implement algorithms that will support behavior optimization, according to its role in smart agriculture and smart food chain, reducing the ecological footprint and economic costs and increasing the food and production and consumption safety.

- **In the Energy sector**

IoT enables the connection and monitoring of energy resources and goods, from almost anywhere, using connected devices and utilities so that energy consumers / promoters can have access to monitor and improve their energy efficiency. Using IoT technology, the utility programs are equipped to provide more power to improve operations efficiency, reduce emissions and management costs and restore their power faster, while energy operators (private and public) are able to immediately recognize outages; allowing them to improve their efficiency

- **Smart City**

There are some basic elements needed to set up a smart city, such as smart society, smart buildings, smart energy, intelligent lighting, intelligent mobility, intelligent management of water etc. The basic infrastructure of the above is based on interconnection of sensors, actuators, and electronic systems, which deal with software, data, support Internet and PC connection. The IoT is applied to improve all those systems that create a smart city that are autonomous and interoperable, secure and reliable. The interaction of systems depends on the degree of interconnectivity and systems communication.

3.4 Information Security

Information Security (Information Security) or Information System Security (Information System Security) is the Information Science that aims to protect the information resources of an Information System (IS), from possible damage or malicious actions, which may cause directly or indirectly, reducing their value. Moreover, Software Security aims to provide reliable information, which is available to authorized users when they need it [9].

The safeguarding of information resources and data is based on the principles of the three fundamental properties of Information Security.

These properties are known internationally as "CIA", and are as follows [10]:

• Confidentiality: concerns the protection of information from its unauthorized disclosure (reading).

• Integrity (Integrity): concerns the protection of information from its unauthorized change (modification or deletion).

• Availability: concerns the safeguarding of authorized access (whether for disclosure or change) to information, without obstacles or delay.

3.5 Key Definitions in Software Security

An **asset** is any object (computing or network resource or data), which has value for its owner and for this reason must be protected from its occasional or permanent loss. **Risk** represents the cause to limit the value of the asset. **Harm** is the limitation of the value of the asset. Any situation that can cause damage to a computer system is a threat to it.

Threats can be categorized into:

• **Natural threats** (environmental threats, e.g. fire, flood, earthquakes, etc.), Intentional threats arising intentionally from malicious users, and Unintentional threats arising unintentionally and incorrectly by computer users systems.
Damages to a system can be caused after **attacks**. Each attack exploits one or more OS vulnerabilities. **Vulnerability** refers to a weakness in system settings or management or a weak point in a security subsystem.

The ranking of vulnerabilities is summarized as follows:

• **Human Vulnerabilities**: are the most critical category for the security of a PF. They are the worst as they can cause the worst effects, since they come from users who know the OS well.

• **Hardware and Software Vulnerabilities**: concern problematic construction, as well as incorrect settings and malfunctions of hardware and software.

• **Media Vulnerabilities**: involve problematic management processes that can lead to the theft or destruction of magnetic, optical or paper data storage media

• **Communications Vulnerabilities**: involve manufacturing weaknesses, incorrect settings, as well as network connection malfunctions.

• **Physical Vulnerabilities**: concern the physical space where they develop and systems (e.g. data centers) operate.

• **Natural Vulnerabilities**: concern natural phenomena (e.g. natural disasters), environmental dependencies etc.

3.6 Related work

In this Section, we are going to present some related work that either include sensors or can tackle security issues on sensors, or even use innovated technologies and methodologies to tackle security issues may arrive in IoT era.

In [13], G. Kornaros, O. Tomoutzoglou et al. are presenting in their work, an investigation into machine learning (ML) and deep learning (DL) methodologies for IoT device security; while examining benefits, drawbacks, and potential. Various solutions hardware-based methods for ML-based IoT authentication, access control, secure offloading, and malware detection schemes are studied and reviewed in the context of integration of accelerators and customizing embedded device architectures for effective use of ML-based methods.

In [14], G. Kornaros et al. present a real implementation on an electric motorcycle. This work refers to a layered systematic approach to harden vehicle's electronic architecture against potential attacks; and more specifically, to ensure that vehicle systems are able to take actions to decreasethe success of cyber-attacks and mitigate the ramifications of potential unauthorized access. This infrastructure

offers secure interconnection mechanisms where trustworthy communications for electronic control nodes can be ensured; hardware firewall as well, to prevent any unauthorized access from untrusted applications/firmware. They introduce a secure technology that encourage prevention from cyber-attacks in automotive Controller Area Network (CAN) protocol which is used in in-vehicle networks; named secure CAN (sCAN) that respects standard CAN-bus and is made for security mechanisms implemented in software or hardware; while adding less than 1 ms latency on the communication.

In [15], G.Trouli&G.Kornaros, as a continuing work of [14] and inspired by the literature review of [13], they introduce virtual sensors by extending vehicle's gateway functionality with supervised machine learning to monitor temporal behavior of CAN bus messages. They designed a vehicle gateway enabled with k-NN classification to achieve real-time analysis of traffic from three fully loaded CAN buses for secure vehicle networking.

In [16], O. Vermesan, M. Coppola et al. present the new technological developments of Internet of Things (IoT) and Industrial Internet of Things (IIoT) and the how Artificial Intelligence (AI) is benefited from. Some of the improvements,, according to the writers include:"edge computing processing, new sensing capabilities, more security protection and autonomous functions accelerating progress towards the ability for IoT systems to self-develop, self-maintain and self-optimize". In this book Chapter, the writers are trying to provide a complete review of the most recent advances in the next wave of the IoT; as well as platforms and smart data aspects that will offer intelligence, sustainability, dependability, autonomy, and also can support human-centric solutions.

In [17], connection of industrial automation devices and equipment with cloud-based systems to process and analyze information faster and toprovide innovated customer experience and services is described. More specifically, LoRaWAN environment along with concepts required in order complete IIoT security to be achieved and implemented are discussed. The threat model includes attacks such as malicious network, firmware attacks, etc. by modification of the IoT node firmware or injection of malicious code into the firmware/operating system/kernel driver.

In [18], research challenges of ensuring security in Cyber-physical systems (CPS) are presented. More specifically, due to CPS applications are able to access and modify safety critical device internals; cyber-physical attacks can easily affect the integrity, availability and confidentiality ofthese systems. Some examples could include: false-data-injection, sensor and actuator attacks, replay attacks, and also denial-of-service attacks. This chapter presents an architectural approach as well as methods for open CPS applications.

In [19], intelligent and dynamic security policy enforcement methodologies and networking technologies such as SDN-NFV have been proposed, in order to minimize cyber-security threats at the edge of the network in Internet-of Things (IoT) domains. The aforementioned methodologies can ensure network communications for IoT services where traditional security is embedded and privacy riskssuch as service hijacking, DDoS attack, denial service, IP spoofing, man-in-the-middle, may arise. In this work, they extend these frameworks and present a software-defined

protection-oriented hardware technique that supports physical isolation of memory compartments and of hardware devices such as DMAs and accelerators inside modern Systems-on-Chip (SoCs), not only at the edge but also at the IoT high-end accelerator-rich devices. These mechanisms can enhanceIoT ecosystem security by design.

In the article [20], S. Leivadaros, G. Kornaros and M. Coppola propose an implementation of IOTA Tangle architecture for data transactions; which is extended with HMAC signing through using STM32 (F7 CPU) IoT devices. They also present evaluation results that show the following: with 32 light nodes to exceed 28 transactions per second by using 4 full nodes. Thus, making IOTA-based distributed ledger, an effective solution for IoT-based manufacturing environments with zero-value (data) transactions can be achieved.

In [21], D. Bakoyiannis, O. Tomoutzoglou, G. Kornaros and M. Coppolaintroduce hardware mechanisms to ensure security in terms of secure key and signature storage through RFID/NFC secure modules and an IoT infrastructure communicating over LoRaWAN in conjunction with Hyperledger Fabric for traceability and immutability. They also present a practical implementation along with the evaluation where the results show the following: an average throughput of more than 70 transactions/sec for 16 peers.

In [22], G. Kornaros, D. Bakoyiannis, O. Tomoutzoglou,et al. propose a lightweight technique in order to enable virtual trusted channel and normal untrusted channel over the same physical CAN-bus network (TrustNet) and their intention is to secure CAN-bus sensitive communications by providing protection against replay attacks with minimum overhead and full legacy support.

In [23], D. Bakoyiannis, O. Tomoutzoglou, and G. Kornarosdepict a secure over-the-air firmware updating that offer homogenized updating process across OEMs, suppliers and sub-tiers decreasing at the same time the costs for security precautions and cryptographic countermeasures for each sub-system. This work aims to overcome any attacks to the servers, to the networks as well as to the diverse electronic control units (ECUs) in modern vehicles. They also demonstrate a real vehicle case (STM32F7xx-based prototype).

In [24], G. Kornaros and S. Leivadarosdemonstrate propose a secure framework for runtime updating of firmware in Internet of Things devices where they execute critical applications. They proposed and developed a methodology where dynamic updating of real-time applications is achieved; when executing on a Xilinx ZYNQ-based platform. They present a bio-signal monitoring use case where there are accelerometer data andthe results are referring to if a person has fallen; while a distant medical management system can dynamically perform firmware updates.

Chapter 4.Zephyr OSand SHA–2

4.1. What is Zephyr?

Zephyr os[25,26] is a real time operating system. Its use includes embedded systems and microprocessors. It can be supported by multiple architectures under the apache licenses 2.0.

It includes the following:

- a kernel
- components and libraries
- device drivers
- protocol stacks
- file systems
- firmware updates

All the aforementioned described elements that Zephyr os provides are needed in order full application software to be implemented

It is available on windows / mac / linux and very simple to use and install.

Zephyr also offers numerous features such as:

- Extensive suite of Kernel services
- Multiple Scheduling Algorithms
- Highly configurable / Modular for flexibility
- Cross Architecture
- Memory Protection
- Compile-time resource definition
- Optimized Device Driver Model
- Devicetree Support
- Native Networking Stack supporting multiple protocols
- Bluetooth Low Energy 5.0 support
- Native Linux, macOS, and Windows Development
- Virtual File System Interface with LittleFS and FATFS Support
- Powerful multi-backend logging Framework
- User friendly and full-featured Shell interface
- Settings on non-volatile storage
- Non-volatile storage (NVS)
- Native POSIX port

4.2. Zephyr OS Features

Zephyr has RO/NX memory protection, stack depth overflow prevention, and stack buffer overflow detection, like the case of Linux. There is no kernel and no user

address space layout randomization(ASLR), which “will likely move to a build time randomization and a small boot time relocation,” according to Smalley and James [27].The aforementioned writers explained that in Zephyr, the process isolation is missing, but nevertheless there is a userspace thread model.

In Zephyr, the security is dependent on particular SoCs and kernel configurations and in general the user is working in a single application compared to Linux where there are a number of core OS neutral and independent security features. The first release of Zephyr had a single executable with a single address space with all threads in supervisor mode and no memory protection or virtual memory. However, Zephyr added OS protections, and minimized changes to kernel APIs in order to be backward compatible. Zephyr philosophy is to do almost everything at build time, and then as much as possible at last view time, to minimize runtime overheads and to ensure bounded latency for real-time.

Because some of the MCUs Zephyr targets include memory protection units (MPUs), while others do not, this makes the security of Zephyr, complicated. Zephyr started offering memory safeguards in releases 1.8 and 1.9, with support for both kinds of MCUs. The Kernel Self Protection Project (KSPP) for Linux's lkdtm tests [30] served as the inspiration for the NSA team's creation of a set of kernel memory protection tests. According to Smalley and James [27] [28], "the tests were useful in finding flaws in Zephyr MPU drivers and are now used for regression testing."

Versions 1.10 and 1.11 of Zephyr added userspace support for user mode threads with isolated memory. The userspace tests created by Smalley's team "to confirm the security properties for user mode threads were being applied." There is no virtual memory in Zephyr's userspace memory model, which is still constrained to a single executable and address space. Smalley and James said, "It can support user mode threads but not complete processes”.

The object permissions paradigm used by Zephyr's security features requires that user threads be granted access to an object before they can utilize it. According to Smalley and James [27],[28], access privileges can be granted from a kernel mode thread to a user mode thread, and they can be passed down via an inheritance mechanism. All user threads have access to every global variable in the program because it is an all or nothing model.

This all-or-nothing strategy imposes a considerable cost on the application developer, who must manually design the application global variable memory layout to match MPU limits. In order to make up for this, the NSA team[28] created a feature that supports a slightly more developer friendly approach of organizing application global based on desired protections and is scheduled for version 1.13.
Some Zephyr security future work that is in progress includes adding MPU virtualization, which would allow users to support a larger number of regions instead of just eight that can be swapped in and out of the MPU on demand, according to Smalley and James [27], [28].

Kernel code is completely trusted in Zephyr [27],[28]. In order to minimize runtime overheads, Linux-type mitigations for kernel vulns leveraging KSPP kernel self-

protection features can be applied. Utilizing armv8-m for Cortex-M MCUs to provide TrustZone security is a plus as well.

4.3. What is SHA-2and is SHA-2 secure?

SHA-2 (**Secure Hash Algorithm 2**)[31, 32] is a set of cryptographic hash functions. These functions are designed by the United States National Security Agency (NSA) in 2001 and are built using the Merkle–Damgård construction.

The SHA-2 family consists of six hash functions, which are the following: **SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256**.

SHA-256 and SHA-512 are novel hash functions computed with eight 32-bit and 64-bit words, respectively. They use different shift amounts and additive constants. However, their structures are virtually identical; differing only in the number of rounds.

SHA-224 and SHA-384 are truncated versions of SHA-256 and SHA-512 respectively, computed with different initial values. SHA-512/224 and SHA-512/256 are also truncated versions of SHA-512.

Currently, the best public attacks break preimage resistance for 52 out of 64 rounds of SHA-256 or 57 out of 80 rounds of SHA-512, and collision resistance for 46 out of 64 rounds of SHA-256.

SHA-2 family of algorithms is characterized as secure; since there has been significant research into the security of the SHA-2 family over the years, and no significant issue is detected. These are some of the reasons that SHA-2 family of algorithms is used in this Thesis, where a secure hash algorithm is needed.

Each of the six algorithms are secure among various scenarios.For example, if length extension attacks are a threat, the best solution would be SHA-512/224 or SHA-512/256.

Chapter 5. Implementation & code in C

5.1. Implementation in C – Why?

The implementation that accompanies this thesis is basically in C.

Some of the reasons why C language is selected include:

- **Flexibility**: C language is extensively used and provides flexibility [34] in terms of memory management and allocation [35] (namely, a user has control on memory allocation/reallocation – e.g. calloc (), malloc ()).

- **Portability**: C language is portable [36] with a huge number of libraries to be used that practically are compatible to any processor architecture. It is known that compilers, libraries, and interpreters for other programming languages are written in C.

- **Simplicity**: C is language is simple to use [33, 37] since it combines characteristics of high-level and low-level languages, as well as it provides simple break down of code into smaller parts, since it is a structured language (namely, functions written in C can be used to break down a whole program into smaller and allow the reusability of code.

- **Speed**: C provides a considerable speed in compilation and execution (namely, lesser inbuilt functions, the lesser overhead).

- **Minimal C library part of Zephyr** [38]: First of all, the minimal C library is part of Zephyr and second this library contains a set of C functions that are needed by Zephyr [REF]. The functions that are implemented in this library, included with Zephyr are the following:

    - abs()
    - atoi()
    - bsearch()
    - calloc()
    - free()
    - gmtime()
    - gmtime_r()
    - isalnum()
    - isalpha()
    - isdigit()
    - isgraph()
    - isprint()
    - isspace()
    - isupper()
    - isxdigit()
    - localtime()
    - malloc()
    - memchr()
    - memcmp()

- memcpy()
- memmove()
- memset()
- mktime()
- rand()
- realloc()
- snprintf()
- sprintf()
- strcat()
- strchr()
- strcmp()
- strcpy()
- strlen()
- trncat()
- strncmp()
- strncpy()
- strrchr()
- strstr()
- strtol()
- trtoul()
- time()
- tolower()
- toupper()
- vsnprintf()
- vsprintf()

5.2. Structure of the code

The package that accompanies this thesis includes:

- a source file (main.c)
- two header files (hash.h / crypto.h)

5.2.1 Source file– Description of deliverable package

5.2.1.1 main.c

The main.c source file contains the following:

The first part includes the libraries that are needed in order the code to be compiled and executed (see Figure6). As far as we can see the first four lines, include the libraries of zephyr.

```
#include <zephyr.h>
#include <zephyr/kernel.h>
#include <zephyr/device.h>
#include <zephyr/drivers/sensor.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <crypto/crypto.h>
#include <crypto/hash.h>
```

Figure 5 : include header files

- **#include<zephyr .h>**: This header is used to enable the zephyr os.
- **#include<zephyr/kernel .h>:** This header includes all the resources from zephyr Linux kernel.
- **#include <zephyr/device .h>:**This header is responsible for detecting the divide that zephyr is booting on and the peripheral devices , so it will set the system to support each hardware part , at the best possible clocks. In that way the hardware will give the best results to the user (performance issues).
- **#include<zephyr/drivers/sensor .h>:** This header is responsible for detecting the sensors that are connected on the main board, so that the best data exchange between the board and the sensors can be achieved.
- **#include<stdio .h>:** This header defines three variable types, several macros, and various functions for performing input and output.
- **#include<stdlib .h>:** This header defines four variable types, several macros, and various functions for performing general functions.
- **#include<string .h>:** This header defines one variable type, one macro, and various functions for manipulating arrays of characters.
- **#include<crypto/crypto .h>:** This header contains the crypto abstraction layer APIs. It contains function to start and stop the hash process.
- **#include<crypto/hash .h>:** This header is used to select one or more hashing algorithms for the project.

The second part includes definitions of variables that are used in the body of the code(see Figure7).

```
#define STACK_SIZE (2048) // <-- the size of the stack for each thread
#define MAIL_LEN 64       // <-- the max length that a mail can have
#define CRYPTO_DRV_NAME CONFIG_CRYPTO_MBEDTLS_SHIM_DRV_NAME // <-- driver name

#define HASH_SIZE 32 // <-- the size of the hash
#define LOG_HASH 0   // <-- This log allows to user to see the hash that is been generated.

#define CMD_REQ_MALLOC 1 // <-- Malloc request from thread_a to thread_b
#define CMD_MALLOC_PTR 2 // <-- Malloc pointer from thread_b to thread_a
#define CMD_DATA_READY 3 // <-- Data alert from thread_b to thread_a
```

Figure 6 : Definition of Variables

- **stack_size(2048):** The stack_size sets the thread_stack_size of each thread.
- **Mail_len:** The mail_len is the max available length of the mailbox buffer. It is needed to have more than 32 bytes of size. The reason is because the hash is 32 bytes but we do not know the size of the message each time.
- **Crypto_drv_name:** This is the driver for the sha-256
- **Hash_size:** The hash has the same size each time .32 bytes.
- **Log_hash:**This define a part of the code to be ignored. If it is set to 1 , then the preprocessor will compile the code between the #if ….. #endif.

- **CMD_REQ_MALLOC – CDM_MALLOC_PTR – CMD_DATA_READY**: These three variables define, set the option, so that the threads will know for what the message is for.

The third part includes the declarations of the structs and functions used in the main function that constitutes the core of the implementation that accompanies this thesis (see Figure8).

```
static K_THREAD_STACK_DEFINE(ta_stack, STACK_SIZE); // <-- thread_a stack
static K_THREAD_STACK_DEFINE(tb_stack, STACK_SIZE); // <-- thread_b stack

static struct k_mbox exchange_mbox; // <-- here the mailbox is set
struct k_thread ta, tb;             // <-- here the thread names are set

void thread_a(void *, void *, void *); // <-- function of thread_a
void thread_b(void *, void *, void *); // <-- function of thread_b
```

Figure 7: Signatures of functions and structs

- **K_thread_stack_define: T**he first variable is the actual name of the threads' stack. The second variable is the stack size of the thread.
- **K_mbox:** it is the mailbox structure, where the threads will use to communicate with each other.
- **K_thread: T**he k_thread is the name of the thread.

```
// The uint_7 is created so there will be 1 kb of integers (128 integers)
// In that way every time that the buffer reaches the maximum, the values will be written
// at the begging of the buffer.
struct uint7 head = {0};
struct uint7 tail = {0};
```

Figure 8: UINT7

- **Uint7:** The reason the unsigned integer 7 is created, is that the threads will use 1 kilobyte of memory space. In that way there will be 128 integers. So when the thread will reach the 128^{th} integer of the buffer it resets and rewrites at the same memory space.

The fourth part includes the implementation of the functions. This part consists of 5 functions which are the following:

- The hash function is responsible for calculating the hash from the given parameters (see Figure10).

```c
void hash(uint8_t * hash_in, uint8_t * hash_out, uint16_t size)
{
    struct device *dev;  // <-- Runtime device structure
    struct hash_ctx ctx; // <-- Pointer to the context structure
    struct hash_pkt pkt; // <-- Structure encoding IO parameters of a hash operation

    ctx.flags = CAP_SYNC_OPS | CAP_SEPARATE_IO_BUFS;
    dev = device_get_binding(CRYPTO_DRV_NAME);

    hash_begin_session(dev, &ctx, CRYPTO_HASH_ALGO_SHA256); //<-- the hash session starts here

    pkt.in_buf = hash_in;   // <-- buffer as input to create the hash
    pkt.out_buf = hash_out; // <--  buffer as output the hash result
    pkt.in_len = size;      // <-- the length of the hash_in buffer

    hash_compute(&ctx, &pkt); // <-- Perform a cryptographic hash function

    hash_free_session(dev, &ctx); // <-- the hash session stops here

    #if LOG_HASH == 1
        printk("Hash: ");
        for (int i = 0; i < 32; i++) //<-- this loop will print the hash
        {
            printk("%02X ", hash_out[i]);
        }
        printk("\r\n");
    #endif
}
```

Figure 9: Hash function

When the hash function is called, there are 3 parameters. The hash_in, the hash_out, and the size. The first and the second are pointer for the input and output.

For hash computation, there is needed to start and stop the process. At the begging the driver needs to read the device info, the ctx pointer(context),and the algorithm to compute the hash. In this Thesis, SHA-256 is used. The hash is created and then the memory for the whole process needs to be freed.

- The send function that is called from the threads when a mailbox needs to be send(see Figure11). Note that in this function there is a call of the hash function in order to generate the hash before sending the mail.

```c
void send(struct k_mbox *mbox, uint8_t *id, uint8_t command, uint8_t *data, uint16_t size)
{
    struct k_mbox_msg mmsg; //<-- the message that will be send will be stored here

    uint8_t txdata[MAIL_LEN];          // <-- the data of the message
    uint8_t hash_in[MAIL_LEN] = { 0 }; // <-- Hash input
    uint8_t hash_out[32] = { 0 };      // <-- Hash output

    memcpy(hash_in, id, 4);            // <-- store the id into hash_in
    memcpy((hash_in + 4), data, size); // <-- store the data after the id

    hash(hash_in, hash_out, (size + sizeof(id))); // <-- call of local hash function for hash compute

    memcpy(txdata, hash_out, HASH_SIZE);      // <-- store the hash_out into txdata
    memcpy((txdata + HASH_SIZE), data, size); // <-- store the data after the hash_out into the txdata

    mmsg.info = command;            // <-- message request type
    mmsg.size = (HASH_SIZE + size); // <-- the size of the mail
    mmsg.tx_data = txdata;          // <-- data to be send
    mmsg.tx_target_thread = K_ANY;  // <-- the destination thread

    k_mbox_put(mbox, &mmsg, K_MSEC(1000)); // <-- this function puts the mail in a queue
}
```

Figure 10: Send function

When the send function is called, there are 5 parameters. The mailbox, the thread id, the command, the data, and the size , of the data to be hashed. After copping the thread id into the hash_in, it is extended by the size of the message. At the end, after the hash function is called (by zephyr), the output will be a hash, created with the thread id combined with the user data. Then a message is formed to be sent by the mailbox to the receiver thread.

- The receive function is called from the threads when a mailbox needs to be read(see Figure12).Note that the 2 threads know the id of each other, soin this function there is a call of the hash function in order tothe hash to be generated, as well as to check if there are any changes on the received hash.

```
void receive(struct k_mbox *mbox, uint8_t *sender_id, uint8_t *data, uint16_t size, uint8_t *command)
{
    struct k_mbox_msg mmsg = { 0 }; // <-- here the message will be stored

    uint8_t rxdata[MAIL_LEN];            // <-- data to be recieved
    uint8_t hash_in[MAIL_LEN] = { 0 };   // <-- Hash input
    uint8_t hash_out[HASH_SIZE] = { 0 }; // <-- Hash output

    mmsg.size = HASH_SIZE + size;  // <-- sizeof(rxdata)
    mmsg.rx_source_thread = K_ANY; // <-- the source thread

    k_mbox_get(mbox, &mmsg, rxdata,  K_MSEC(1000)); // <-- this function gets the mail from the queue

    *command = mmsg.info;  // <-- here is the command option that says if there is a new malloc or data to be read.

    if(mmsg.info != 0)
    {
        memcpy(hash_in, sender_id, 4);                      // <-- store the sender_id into hash_in
        memcpy((hash_in + 4), (rxdata + HASH_SIZE), size);  // <-- store the mailbox data at after the id

        hash(hash_in, hash_out, (size + sizeof(sender_id)));// <-- call of local hash function for hash compute

        if(memcmp(rxdata, hash_out, HASH_SIZE) == 0)  // if the input hash is the same as the local hash then the sender is valid.
        {
            memcpy(data, (rxdata + HASH_SIZE), size);
        }
        else
        {
            printk("Data invalid, aborting...\r\n");
        }
    }

}
```

Figure 11: Receive function

When the send function is called, there are 5 parameters which are the following: the mailbox, the sender thread id, data, size, and command. The 2 threads know the id of each other. For security reasons, it is needed to recreate the incoming hash, by ignoring the first 32 bits of the incoming mailbox. Then, using the sender id and the rest of the mailbox data, the local hash is created. If the local hash is the same as the incoming hash, then the message is valid. The received data are store to the data pointer, so in that way the thread with data in need will be informed.

- The thread_a is responsible to request the size of the memory (malloc) and to display the values that thread_btakes(see Figure 13).

```
void thread_a(void *p1, void *p2, void *p3)
{
    static struct k_mbox_msg mmsg;      // <-- mailbox message
    static uint16_t malloc_size = 1024; // <-- 1KB of memmory
    static uint8_t txdata[MAIL_LEN];    // <-- the data for transmition
    static uint8_t rxdata[MAIL_LEN];    // <-- the data for recieption

    struct sensor_value *mem;           // <-- pointer for malloc
    struct k_mbox *mbox = (struct k_mbox *)p1; // <-- this a local mailbox

    int i = 0;

    struct uint7 head = { 0 };         // <-- the head of the 10 pack on each receive
    struct uint7 read_counter = { 0 }; // <-- the counter that indicates the memory cell , from 0 - 127

    uint32_t malloc_pointer = 0; // <-- setting the malloc pointer

    uint8_t command = 0; // <-- setting the command

    // setting the mail for transmission
    txdata[0] = malloc_size >> 8;
    txdata[1] = malloc_size;

    uint8_t id[4] = {0xBA, 0xAD, 0xBE, 0xEF};        // <-- the id of thread_a
    uint8_t sender_id[4] = {0xCA, 0xFE, 0x12, 0x34}; // <-- the id of thread_b

    send(mbox, id, CMD_REQ_MALLOC, txdata, sizeof(malloc_size)); //<-- sending the malloc size to thread_b

    receive(mbox, sender_id, rxdata, 4, &command); // <-- the pointer of malloc from thread_b

    //Here the malloc pointer is shifted because it is send as an unsighed_integer_32 and not as a pointer.
    malloc_pointer = rxdata[0];
    malloc_pointer = malloc_pointer | (rxdata[1] << 8);
    malloc_pointer = malloc_pointer | (rxdata[2] << 16);
    malloc_pointer = malloc_pointer | (rxdata[3] << 24);

    mem = (struct sensor_value *) malloc_pointer; // <-- after the shifting it is needed to cast the interger as a pointer to memory

    printk("Received pointer 0x%p\r\n",mem);

    k_sleep(K_MSEC(1500));
    //In this while loop , it is cheacked if a new data pack has arrived.
    //If there is then it will be displayed on terminal
    //After the display the command becomes zero again
    while(1)
    {
        receive(mbox, sender_id, rxdata, 1, &command);

        if(command == CMD_DATA_READY)
        {
            head.u7 = rxdata[0];

            for(i = 0; i < 10; i++)
            {
                read_counter.u7 = i + head.u7;
                printf("Index: %d, Temp: %d.%06d \r\n",read_counter.u7, mem[read_counter.u7].val1, mem[read_counter.u7].val2);
            }

            command = 0;
        }
    }
}
```

Figure 12: Tread a

Thread_a is the master of the 2 user created threads. When the thread_a starts , it sets the mailbox, the memory size for the operation, and the pointers for the data that will be used from the slave. First, the malloc size is sent to thread_b(slave), after that , if the malloc was successfully allocate the memory, it gets the malloc pointer. Then, every 10 new sensor reads, it will be informed by the thread_b to display the values on the console.

- The thread_b is responsible to reserve the memory (malloc) that thread_a requests' and to inform it accordingly by providing each time 10 new entries of temperatures.

```c
void thread_b(void *p1, void *p2, void *p3)
{
    const struct device *dev1 = device_get_binding(DT_LABEL(DT_INST(0, bosch_bme680))); // <-- getting the devide label (bosch -- bme680)
    struct sensor_value temp;         // <-- Tempreture from BME680
    struct device *dev;               // <-- Runtime device structure
    struct hash_ctx ctx;              // <-- Pointer to the context structure
    struct hash_pkt pkt;              // <-- Structure encoding IO parameters of a hash operation
    static struct k_mbox_msg mmsg;    // <-- mailbox message
    struct sensor_value *mem;         // <-- pointer for malloc
    static uint8_t rxdata[MAIL_LEN]; // <-- data to receive
    static uint8_t txdata[MAIL_LEN]; // <-- data to transmit
    static uint16_t malloc_size = 0; // <-- set as zero on reciever

    int i = 0;

    uint32_t malloc_pointer = 0; // <-- setting the malloc pointer

    uint8_t command = 0;

    uint8_t id[4] = {0xCA, 0xFE, 0x12, 0x34};        // <-- the id of thread_b
    uint8_t sender_id[4] = {0xBA, 0xAD, 0xBE, 0xEF}; // <-- the id of thread_a

    struct k_mbox *mbox = (struct k_mbox *)p1; // <-- this a local mailbox

    // The uint_7 is created so there will be 1 kb of integers (128 integers)
    // In that way every time that the buffer reaches the maximum, the values will be written
    // at the begging of the buffer.
    struct uint7 head = {0};
    struct uint7 tail = {0};

    receive(mbox, sender_id, rxdata, 2, &command);  // <-- this mailbox contains the malloc size

    printk("Received data: %02X %02X\r\n", rxdata[0], rxdata[1]);

    malloc_size = (rxdata[1] << 8) | rxdata[0]; //<-- get the size for malloc

    if(malloc_size > 0)
    {
        mem = (struct sensor_value *)k_malloc(malloc_size); //<-- malloc 1KB for this example
        printk("malloc pointer --> %p\r\n", mem); //<-- print malloc pointer

        malloc_pointer = mem; //<-- here the pointer is stored as an unsighed_32bit_integer

        //Here the malloc pointer is shifted because it is send as an unsighed_integer_32 and not as a pointer.
        txdata[0] = malloc_pointer;
        txdata[1] = (malloc_pointer >> 8);
        txdata[2] = (malloc_pointer >> 16);
        txdata[3] = (malloc_pointer >> 24);

        send(mbox, id, CMD_MALLOC_PTR, txdata, sizeof(malloc_pointer));//<-- the mailbox contains the pointer of the malloc
    }

    // In this while the tempe values are read and stored in packes of 10.
    // There is a delay because the read is to fast. In that was there will be diferent values.
    while(1)
    {
        k_sleep(K_MSEC(400));

        for(i = 0; i < 10; i++)
        {
            sensor_sample_fetch(dev1);
            sensor_channel_get(dev1, SENSOR_CHAN_AMBIENT_TEMP, &temp);
            mem[tail.u7].val1 = temp.val1 - 6; //<-- the int part of the temp (-6 deg for calibration)
            mem[tail.u7].val2 = temp.val2;//<-- the float part of the temp

            tail.u7++;

            k_sleep(K_MSEC(253)); // <-- the sample time of the sensor is 253ms.
        }

        txdata[0] = head.u7;
        send(mbox, id, CMD_DATA_READY, txdata, 1); //<-- A pack of 10 integers is been send to thread_a

        head.u7 = tail.u7;
    }
}
```

Figure 13: Tread b

Thread_b is the slave of the two user threads. When the thread_b starts, it sets the sensor driver, the mailbox. First, it waits for the malloc size from thread_b. If the malloc function is successful, it returns the pointer to thread_a. Then it sets up the sensor and reads 10 temps values, one every 200 ms, and informs the thread_a for the new reads.

Finally, the main function defines the parameters “k_thread_create” in order for the thread to be created as well as the tb_tid : receiver thread and tba_tid: sender thread are both created(see Figure 15). It has void type and is responsible for creating the threads and prevents the program from quitting; using a while(1) loop.

After setting the ids and the mailbox the threads are created. The first thread is the master whilethread_b is the slave thread. The reason why the thread_b is created first, it is because the send function is blocking.
Each thread needs some parameters such as: the thread name, the thread stack size name, the stack size, the thread name, pointer 1, pointer 2, pointer 3, the permissions , and finally the delay of the creation.

```c
void main(void)
{
    static k_tid_t ta_tid, tb_tid; // <-- thread_id
    k_mbox_init(&exchange_mbox);   // <-- initialize of mailbox

    // Here are the parameters the k_thread_create need for the thread creation !!
    // k_tid_t k_thread_create(struct k_thread * new_thread, k_thread_stack_t * stack, size_t stack_size,
    // k_thread_entry_t entry, void * p1, void * p2, void * p3, int prio, uint32_t options, k_timeout_t delay)

    // here the receiver thread is created !!
    tb_tid = k_thread_create(&tb, tb_stack, STACK_SIZE, thread_b, &exchange_mbox, NULL,
                NULL, -1, K_INHERIT_PERMS, K_NO_WAIT);

    // here the sender thread is created !!
    ta_tid = k_thread_create(&ta, ta_stack, STACK_SIZE, thread_a, &exchange_mbox, NULL,
                NULL, -1, K_INHERIT_PERMS, K_NO_WAIT);

    while (1)
    {
        //....//
    }
}
```

Figure 14: Main function

5.2.1.2 crypto.h

This file contains the Crypto Abstraction layer APIs under the license: SPDX-License-Identifier: Apache-2.0 which provides some important functions as follows:

```c
/**
 * @brief Setup a crypto session
 *
 * Initializes one time parameters, like the session key, algorithm and cipher
 * mode which may remain constant for all operations in the session. The state
 * may be cached in hardware and/or driver data state variables.
 *
 * @param  dev      Pointer to the device structure for the driver instance.
 * @param  ctx      Pointer to the context structure. Various one time
 *          parameters like key, keylength, etc. are supplied via
 *          this structure. The structure documentation specifies
 *          which fields are to be populated by the app before
 *          making this call.
 * @param  algo     The crypto algorithm to be used in this session. e.g AES
 * @param  mode     The cipher mode to be used in this session. e.g CBC, CTR
 * @param  optype   Whether we should encrypt or decrypt in this session
 *
 * @return 0 on success, negative errno code on fail.
 */
static inline int cipher_begin_session(const struct device *dev,
                       struct cipher_ctx *ctx,
                       enum cipher_algo algo,
                       enum cipher_mode  mode,
                       enum cipher_op optype)
{
```

```c
/**
 * @brief Cleanup a crypto session
 *
 * Clears the hardware and/or driver state of a previous session.
 *
 * @param  dev      Pointer to the device structure for the driver inst
 * @param  ctx      Pointer to the crypto context structure of the ses
 *          to be freed.
 *
 * @return 0 on success, negative errno code on fail.
 */
static inline int cipher_free_session(const struct device *dev,
                      struct cipher_ctx *ctx)
{
```

```
    /**
     * @brief Registers an async crypto op completion callback with the driver
     *
     * The application can register an async crypto op completion callback handler
     * to be invoked by the driver, on completion of a prior request submitted via
     * cipher_do_op(). Based on crypto device hardware semantics, this is likely to
     * be invoked from an ISR context.
     *
     * @param  dev   Pointer to the device structure for the driver instance.
     * @param  cb    Pointer to application callback to be called by the driver.
     *
     * @return 0 on success, -ENOTSUP if the driver does not support async op,
     *            negative errno code on other error.
     */
    static inline int cipher_callback_set(const struct device *dev,
                                          cipher_completion_cb cb)
    {
/**
 * @brief Perform single-block crypto operation (ECB cipher mode). This
 * should not be overloaded to operate on multiple blocks for security reasons.
 *
 * @param  ctx       Pointer to the crypto context of this op.
 * @param  pkt   Structure holding the input/output buffer pointers.
 *
 * @return 0 on success, negative errno code on fail.
 */
static inline int cipher_block_op(struct cipher_ctx *ctx,
                                  struct cipher_pkt *pkt)
{
    /**
     * @brief Perform Cipher Block Chaining (CBC) crypto operation.
     *
     * @param  ctx       Pointer to the crypto context of this op.
     * @param  pkt   Structure holding the input/output buffer pointers.
     * @param  iv        Initialization Vector (IV) for the operation. Same
     *           IV value should not be reused across multiple
     *           operations (within a session context) for security.
     *
     * @return 0 on success, negative errno code on fail.
     */
    static inline int cipher_cbc_op(struct cipher_ctx *ctx,
                                    struct cipher_pkt *pkt, uint8_t *iv)
    {
    /**
     * @brief Perform Counter (CTR) mode crypto operation.
     *
     * @param  ctx       Pointer to the crypto context of this op.
     * @param  pkt   Structure holding the input/output buffer pointers.
     * @param  iv        Initialization Vector (IV) for the operation. We use a
     *           split counter formed by appending IV and ctr.
     *           Consequently  ivlen = keylen - ctrlen. 'ctrlen' is
     *           specified during session setup through the
     *           'ctx.mode_params.ctr_params.ctr_len' parameter. IV
     *           should not be reused across multiple operations
     *           (within a session context) for security. The non-IV
     *           part of the split counter is transparent to the caller
     *           and is fully managed by the crypto provider.
     *
     * @return 0 on success, negative errno code on fail.
     */
    static inline int cipher_ctr_op(struct cipher_ctx *ctx,
                                    struct cipher_pkt *pkt, uint8_t *iv)
    {
```

```
/**
 * @brief Perform Counter with CBC-MAC (CCM) mode crypto operation
 *
 * @param  ctx       Pointer to the crypto context of this op.
 * @param  pkt   Structure holding the input/output, Associated
 *           Data (AD) and auth tag buffer pointers.
 * @param  nonce     Nonce for the operation. Same nonce value should not
 *           be reused across multiple operations (within a
 *           session context) for security.
 *
 * @return 0 on success, negative errno code on fail.
 */
static inline int cipher_ccm_op(struct cipher_ctx *ctx,
                struct cipher_aead_pkt *pkt, uint8_t *nonce)
{

/**
 * @brief Perform Galois/Counter Mode (GCM) crypto operation
 *
 * @param  ctx       Pointer to the crypto context of this op.
 * @param  pkt   Structure holding the input/output, Associated
 *           Data (AD) and auth tag buffer pointers.
 * @param  nonce     Nonce for the operation. Same nonce value should not
 *           be reused across multiple operations (within a
 *           session context) for security.
 *
 * @return 0 on success, negative errno code on fail.
 */
static inline int cipher_gcm_op(struct cipher_ctx *ctx,
                struct cipher_aead_pkt *pkt, uint8_t *nonce)
{

/**
 * @brief Setup a hash session
 *
 * Initializes one time parameters, like the algorithm which may
 * remain constant for all operations in the session. The state may be
 * cached in hardware and/or driver data state variables.
 *
 * @param  dev      Pointer to the device structure for the driver instance.
 * @param  ctx      Pointer to the context structure. Various one time
 *          parameters like session capabilities and algorithm are
 *          supplied via this structure. The structure documentation
 *          specifies which fields are to be populated by the app
 *          before making this call.
 * @param  algo     The hash algorithm to be used in this session. e.g sha256
 *
 * @return 0 on success, negative errno code on fail.
 */
static inline int hash_begin_session(const struct device *dev,
                struct hash_ctx *ctx,
                enum hash_algo algo)
{

 /**
  * @brief Cleanup a hash session
  *
  * Clears the hardware and/or driver state of a session. @see hash_begin_session
  *
  * @param  dev      Pointer to the device structure for the driver instance.
  * @param  ctx      Pointer to the crypto hash  context structure of the session
  *          to be freed.
  *
  * @return 0 on success, negative errno code on fail.
  */
 static inline int hash_free_session(const struct device *dev,
                struct hash_ctx *ctx)
 {
```

```c
/**
 * @brief Registers an async hash completion callback with the driver
 *
 * The application can register an async hash completion callback handler
 * to be invoked by the driver, on completion of a prior request submitted via
 * hash_compute(). Based on crypto device hardware semantics, this is likely to
 * be invoked from an ISR context.
 *
 * @param  dev   Pointer to the device structure for the driver instance.
 * @param  cb    Pointer to application callback to be called by the driver.
 *
 * @return 0 on success, -ENOTSUP if the driver does not support async op,
 *           negative errno code on other error.
 */
static inline int hash_callback_set(const struct device *dev,
                                    hash_completion_cb cb)
{
```

```c
/**
 * @brief Perform  a cryptographic hash function.
 *
 * @param  ctx       Pointer to the hash context of this op.
 * @param  pkt       Structure holding the input/output.

 * @return 0 on success, negative errno code on fail.
 */
static inline int hash_compute(struct hash_ctx *ctx, struct hash_pkt *pkt)
{
```

```c
/**
 * @brief Perform  a cryptographic multipart hash operation.
 *
 * This function can be called zero or more times, passing a slice of the
 * the data. The hash is calculated using all the given pieces.
 * To calculate the hash call @c hash_compute().
 *
 * @param  ctx       Pointer to the hash context of this op.
 * @param  pkt       Structure holding the input.

 * @return 0 on success, negative errno code on fail.
 */
static inline int hash_update(struct hash_ctx *ctx, struct hash_pkt *pkt)
{
```

5.2.1.3 hash.h

This file contains the Crypto Abstraction layer APIs under the license: SPDX-License-Identifier: Apache-2.0 and is displayed below:

```
typedef int (*hash_op_t)(struct hash_ctx *ctx, struct hash_pkt *pkt,
			 bool finish);

/**
 * Structure encoding session parameters.
 *
 * Refer to comments for individual fields to know the contract
 * in terms of who fills what and when w.r.t begin_session() call.
 */
struct hash_ctx {
	/** The device driver instance this crypto context relates to. Will be
	 * populated by the begin_session() API.
	 */
	const struct device *device;

	/** If the driver supports multiple simultaneously crypto sessions, this
	 * will identify the specific driver state this crypto session relates
	 * to. Since dynamic memory allocation is not possible, it is
	 * suggested that at build time drivers allocate space for the
	 * max simultaneous sessions they intend to support. To be populated
	 * by the driver on return from begin_session().
	 */
	void *drv_sessn_state;

	/**
	 * Hash handler set up when the session begins.
	 */
	hash_op_t hash_hndlr;

	/**
	 * If it has started a multipart hash operation.
	 */
	bool started;

	/** How certain fields are to be interpreted for this session.
	 * (A bitmask of CAP_* below.)
	 * To be populated by the app before calling hash_begin_session().
	 * An app can obtain the capability flags supported by a hw/driver
	 * by calling crypto_query_hwcaps().
	 */
	uint16_t flags;
};
```

```
/**
 * Structure encoding IO parameters of a hash
 * operation.
 *
 * The fields which has not been explicitly called out has to
 * be filled up by the app before calling hash_compute().
 */
struct hash_pkt {

	/** Start address of input buffer */
	uint8_t *in_buf;

	/** Bytes to be operated upon */
	size_t  in_len;

	/**
	 * Start of the output buffer, to be allocated by
	 * the application. Can be NULL for in-place ops. To be populated
	 * with contents by the driver on return from op / async callback.
	 */
	uint8_t *out_buf;

	/**
	 * Context this packet relates to. This can be useful to get the
	 * session details, especially for async ops.
	 */
	struct hash_ctx *ctx;
};

/* Prototype for the application function to be invoked by the crypto driver
 * on completion of an async request. The app may get the session context
 * via the pkt->ctx field.
 */
typedef void (*hash_completion_cb)(struct hash_pkt *completed, int status);
```

Chapter 6. Use Case Scenarios

As we presented in Chapter 2, the equipment used in this Thesis is the board STM32F469i with an ARM® Cortex®-M4 processor. On the board, we have placed an external Bosch sensor - BME680, to get the measurements of the temperatures (10 Celsius values per measurement).

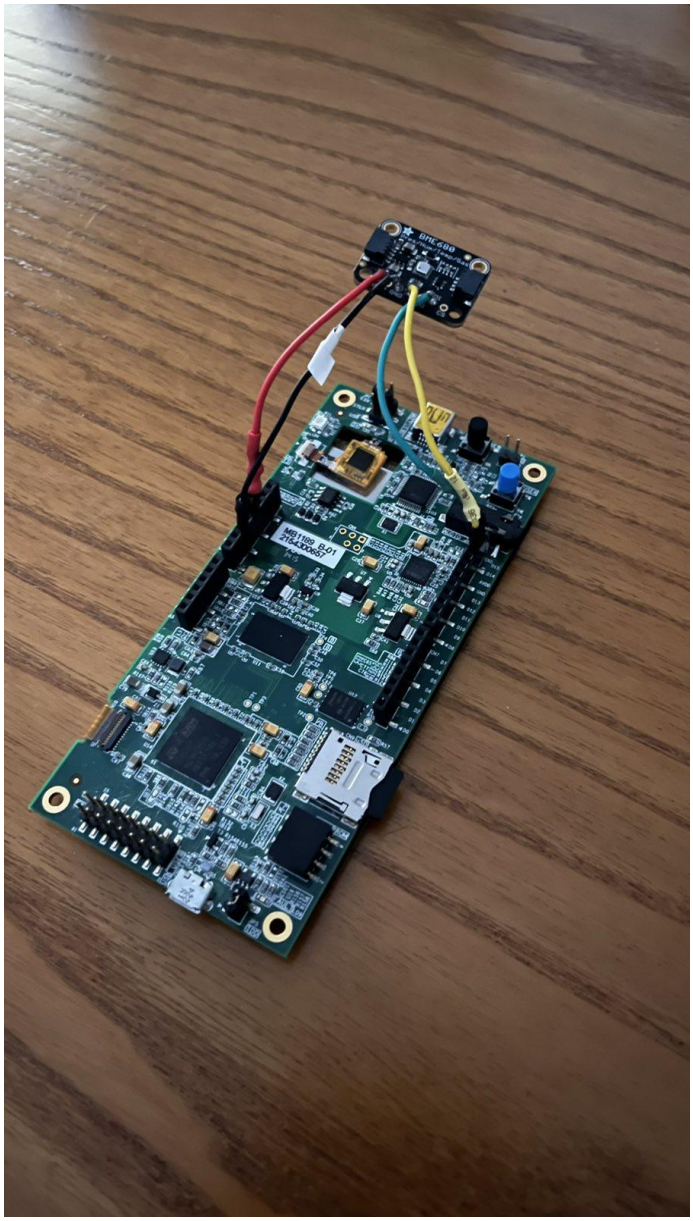

Figure 15: STM32F469I-DISCO along with BME680

As we connect the board to the computer, it immediately powers up and starts running the code that was last loaded onto it (see Figure 17). For instance, if the code that is already uploaded on the board is not the desired one, it is needed to load a new one.

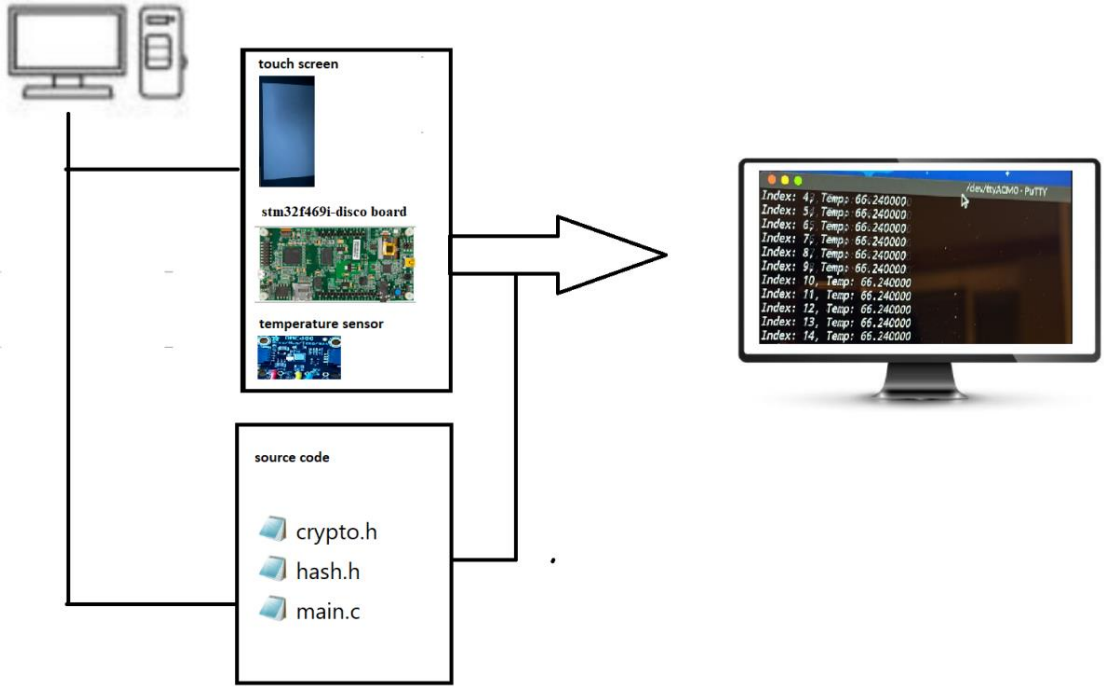

Figure 16: Application using STM32F469I-DISCO along with BME680

Upon board connection, we open a terminal console and we start writing commands.

The commands are as follows:

```
cd zephyrproject/zephyr
west build -b stm32f469i-disco samples/adamantios_demos/final/ --pristine.
west flash
```

The last command helps the code to be loaded onto the board.

The program is executed, the initial constants are defined and subsequently the threads are defined, as well.

The first thread requests memory from the second and then upon the memory is allocated, values from the sensor are retrieved. This process is done until we turn off the power of the board.

The results of the whole process are 10 Celsius values per measurement.

In order to measure the sending time in the sensor and the function k_cycle_get_32 which is located in the kernel.h library, before starting the measurement we keep the number of cycles. Then, upon measure has been taken, and the process has finished, we keep the number of cycles, once more.

We, then subtract the initial measurement of cycles from the later measurement of cycles (sub = later cycles - initial cycles), in order to get the total time it takes to the

read process to be executed (in processor clock cycles) and convert these cycles (using an online converter) in nanoseconds.

These calculations help us to get the performance of the application; using the aforementioned equipment. In the clocks file, we count as follows:
sensor read time | sensor send time | hash time (where max clock 180 Mhz == 5.5555555 nanoseconds).

For this Thesis we conducted measurements; having simulated different environments, that can be applied in different contexts and domains of IoT.
These experiments will be presented in the following sections.

6.1. Use Case Scenario 1 : Room

The first experiment included set of temperatures in the living room, during summer period (August).

Figure 17: STM32F469I-DISCO along with BME680 in a room

The values (see Table 1) are retrieved in the morning (among 1.203.024, for simplicity we have presented the average per 12' for a duration of 4 hours in the morning), in the afternoon (among 1.203.024, for simplicity we have presented the average per 12' for a duration of 4 hours in the afternoon).

Place	Period of the Day	Time of the Day	Temperature
Living Room	Morning	08:00 -12:00	23.590000
Living Room	Morning	08:00 -12:00	24.400000
Living Room	Morning	08:00 -12:00	25.220000
Living Room	Morning	08:00 -12:00	25.540000
Living Room	Morning	08:00 -12:00	25.850000
Living Room	Morning	08:00 -12:00	26.070000
Living Room	Morning	08:00 -12:00	26.200000
Living Room	Morning	08:00 -12:00	26.330000
Living Room	Morning	08:00 -12:00	26.410000
Living Room	Morning	08:00 -12:00	26.490000
Living Room	Morning	08:00 -12:00	26.540000
Living Room	Morning	08:00 -12:00	26.600000
Living Room	Morning	08:00 -12:00	28.420000
Living Room	Morning	08:00 -12:00	28.820000
Living Room	Morning	08:00 -12:00	29.110000
Living Room	Morning	08:00 -12:00	29.140000
Living Room	Morning	08:00 -12:00	29.220000
Living Room	Morning	08:00 -12:00	29.600000
Living Room	Morning	08:00 -12:00	29.920000
Living Room	Morning	08:00 -12:00	29.970000
Living Room	Afternoon	13:00 – 17:00	29.880000
Living Room	Afternoon	13:00 – 17:00	29.640000
Living Room	Afternoon	13:00 – 17:00	29.550000
Living Room	Afternoon	13:00 – 17:00	29.650000
Living Room	Afternoon	13:00 – 17:00	29.740000
Living Room	Afternoon	13:00 – 17:00	29.800000
Living Room	Afternoon	13:00 – 17:00	29.740000
Living Room	Afternoon	13:00 – 17:00	30.180000
Living Room	Afternoon	13:00 – 17:00	30.080000
Living Room	Afternoon	13:00 – 17:00	30.050000
Living Room	Afternoon	13:00 – 17:00	30.040000
Living Room	Afternoon	13:00 – 17:00	30.050000
Living Room	Afternoon	13:00 – 17:00	30.090000
Living Room	Afternoon	13:00 – 17:00	30.080000
Living Room	Afternoon	13:00 – 17:00	29.980000
Living Room	Afternoon	13:00 – 17:00	30.030000
Living Room	Afternoon	13:00 – 17:00	30.090000
Living Room	Afternoon	13:00 – 17:00	29.980000
Living Room	Afternoon	13:00 – 17:00	29.960000
Living Room	Afternoon	13:00 – 17:00	29.940000

Table 1: Temperatures retrieved in Living Room and Bathroom during the morning and the afternoon.

These measurements can help people to stabilize and control the living conditions in a house. Having in mind, sensitive group of people, such as elderly people or babies

where a significant change of temperature in the room or during having their bath could be catastrophic and may cause even death. Thus, as a future work in this experiment, we could imagine that this sensor could send a signal to a connected device (e.g. air condition installed) in order the temperature to be reset in normal levels or a message to the owner of the house in order them to open any doors or windows.

6.2. Use Case Scenario 2: Basemen

Another experiment included set of temperatures in a basement, during summer period (August).

The values (see Table 2) are retrieved in the morning (among 2.213.524, for simplicity we have presented the average per 12' for a duration of 4 hours in the morning), in the afternoon (among 2.213.524, for simplicity we have presented the average per 12' for a duration of 4 hours in the afternoon).

Place	Period of the Day	Time of the Day	Temperature
basement	Morning	08:00 -12:00	21.590000
basement	Morning	08:00 -12:00	21.400000
basement	Morning	08:00 -12:00	21.220000
basement	Morning	08:00 -12:00	21.540000
basement	Morning	08:00 -12:00	21.850000
basement	Morning	08:00 -12:00	21.900000
basement	Morning	08:00 -12:00	21.950000
basement	Morning	08:00 -12:00	22.220000
basement	Morning	08:00 -12:00	22.250000
basement	Morning	08:00 -12:00	22.070000
basement	Morning	08:00 -12:00	22.200000
basement	Morning	08:00 -12:00	22.330000
basement	Morning	08:00 -12:00	23.410000
basement	Morning	08:00 -12:00	22.200000
basement	Morning	08:00 -12:00	22.330000
basement	Morning	08:00 -12:00	23.400000
basement	Morning	08:00 -12:00	23.220000
basement	Morning	08:00 -12:00	23.540000
basement	Morning	08:00 -12:00	23.850000
basement	Morning	08:00 -12:00	23.900000
basement	Afternoon	13:00 – 17:00	24.400000
basement	Afternoon	13:00 – 17:00	25.220000
basement	Afternoon	13:00 – 17:00	25.540000
basement	Afternoon	13:00 – 17:00	24.850000
basement	Afternoon	13:00 – 17:00	24.900000
basement	Afternoon	13:00 – 17:00	24.950000
basement	Afternoon	13:00 – 17:00	25.220000
basement	Afternoon	13:00 – 17:00	25.350000
basement	Afternoon	13:00 – 17:00	25.420000
basement	Afternoon	13:00 – 17:00	25.520000

basement	Afternoon	13:00 – 17:00	25.850000
basement	Afternoon	13:00 – 17:00	26.070000
basement	Afternoon	13:00 – 17:00	26.200000
basement	Afternoon	13:00 – 17:00	26.330000
basement	Afternoon	13:00 – 17:00	26.350000
basement	Afternoon	13:00 – 17:00	26.410000
basement	Afternoon	13:00 – 17:00	26.490000
basement	Afternoon	13:00 – 17:00	26.540000
basement	Afternoon	13:00 – 17:00	26.600000
basement	Afternoon	13:00 – 17:00	26.700000

Table 2: Temperatures retrieved in basement during a day.

These measurements can help people to stabilize and control the conditions in a basement where they store for example some groceries or wine. Having in mind, wine producers that the store the collected grapes as well as tones of wine in basement, a significant change of temperature in the basement during summer or winter period could be a disaster. Thus, as a future work in this experiment, we could imagine that this sensor could send a signal to a connected device (e.g. air condition installed) in order the temperature to be reset in normal levels.

6.3. Use Case Scenario 3: factory simulation

Another experiment included set of temperatures in a room, during summer period (August); where a lighter was used near the sensor (during the morning and afternoon) to simulate significant change of temperature in a factory, when a problem regarding a machine there caused.

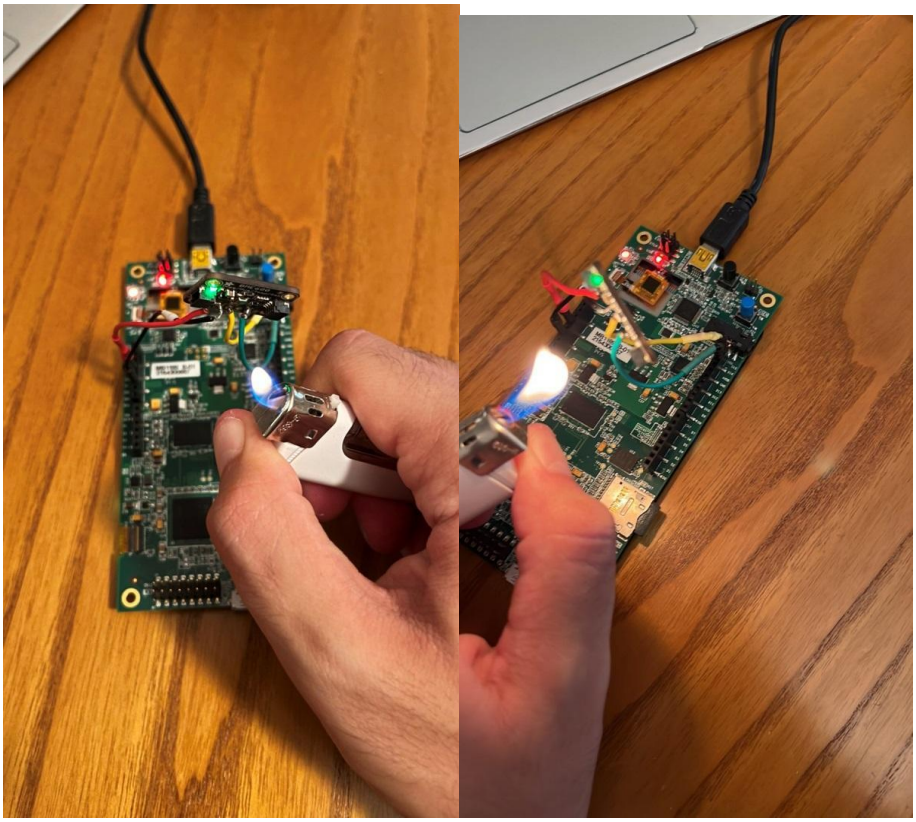

Figure 18:STM32F469I-DISCO along with BME680 hardware failure simulation

The values (see Table 3) are retrieved in the morning (among 502.761, for simplicity we have presented some average values for a duration of 4 hours in the morning), in the afternoon (among 502.761, for simplicity we have presented some average values for a duration of 4 hours in the afternoon).

Place	Period of the Day	Time of the Day	Temperature
Factory simulation	Morning	08:00 -12:00	37.600000
Factory simulation	Morning	08:00 -12:00	38.330000
Factory simulation	Morning	08:00 -12:00	38.550000
Factory simulation	Morning	08:00 -12:00	38.800000
Factory simulation	Morning	08:00 -12:00	39.820000
Factory simulation	Morning	08:00 -12:00	40.590000
Factory simulation	Morning	08:00 -12:00	42.380000
Factory simulation	Morning	08:00 -12:00	46.850000
Factory simulation	Morning	08:00 -12:00	49.270000
Factory simulation	Morning	08:00 -12:00	49.880000
Factory simulation	Morning	08:00 -12:00	50.000000
Factory simulation	Morning	08:00 -12:00	51.690000
Factory simulation	Morning	08:00 -12:00	53.930000
Factory simulation	Morning	08:00 -12:00	54.360000
Factory simulation	Morning	08:00 -12:00	53.810000
Factory simulation	Morning	08:00 -12:00	53.520000
Factory simulation	Morning	08:00 -12:00	53.180000
Factory simulation	Morning	08:00 -12:00	52.810000
Factory simulation	Morning	08:00 -12:00	52.220000
Factory simulation	Morning	08:00 -12:00	51.260000
Factory simulation	Afternoon	13:00 – 17:00	50.910000
Factory simulation	Afternoon	13:00 – 17:00	50.510000
Factory simulation	Afternoon	13:00 – 17:00	49.820000
Factory simulation	Afternoon	13:00 – 17:00	49.750000
Factory simulation	Afternoon	13:00 – 17:00	49.230000
Factory simulation	Afternoon	13:00 – 17:00	48.340000
Factory simulation	Afternoon	13:00 – 17:00	48.130000
Factory simulation	Afternoon	13:00 – 17:00	47.330000
Factory simulation	Afternoon	13:00 – 17:00	46.970000
Factory simulation	Afternoon	13:00 – 17:00	45.800000
Factory simulation	Afternoon	13:00 – 17:00	43.230000
Factory simulation	Afternoon	13:00 – 17:00	42.170000
Factory simulation	Afternoon	13:00 – 17:00	41.640000
Factory simulation	Afternoon	13:00 – 17:00	40.830000
Factory simulation	Afternoon	13:00 – 17:00	39.830000
Factory simulation	Afternoon	13:00 – 17:00	38.790000
Factory simulation	Afternoon	13:00 – 17:00	38.670000
Factory simulation	Afternoon	13:00 – 17:00	37.350000
Factory simulation	Afternoon	13:00 – 17:00	36.450000
Factory simulation	Afternoon	13:00 – 17:00	34.150000
Factory simulation	Afternoon	13:00 – 17:00	33.340000

Factory simulation	Afternoon	13:00 – 17:00	32.830000
Factory simulation	Afternoon	13:00 – 17:00	29.390000
Factory simulation	Afternoon	13:00 – 17:00	28.490000
Factory simulation	Afternoon	13:00 – 17:00	28.410000
Factory simulation	Afternoon	13:00 – 17:00	27.450000
Factory simulation	Afternoon	13:00 – 17:00	26.390000
Factory simulation	Afternoon	13:00 – 17:00	26.190000
Factory simulation	Afternoon	13:00 – 17:00	24.880000
Factory simulation	Afternoon	13:00 – 17:00	24.510000
Factory simulation	Afternoon	13:00 – 17:00	23.210000

Table 3: Temperatures retrieved in Factory simulation during a day.

These measurements can help factories or companies that are using machines that can be on fire easily (or potentially), to stabilize and control the conditions in the factory/company. This simulation showed us that from the time that the temperature is increased, it takes around 2 h and a half (in average) to get back to the normal levels. Thus, as a future work in this experiment, we could imagine that this sensor could send a message that something wrong is happening, in order for the firemen reaches the place and the machines to get powered off.

6.4. Use Case Scenario 4: greenhouse simulation

Another experiment included set of temperatures in a room, during summer period (August); where a plastic box was used to cover totally the sensor (during the day) and a hairdryer (during the morning and night) to simulate temperatures in a greenhouse, when a significant change of temperature may arise due to the temperature outside or due to an accident (e.g. low temperatures during winter or really high temperatures in summer).

Figure 19: STM32F469I-DISCO along with BME680 greenhouse simulation

The values (see Table 4) are retrieved in the morning (among 934.156, for simplicity we have presented some average values for a duration of 4 hours in the morning), in the afternoon (among 934.156, for simplicity we have presented some average values for a duration of 4 hours in the afternoon).

.

Place	Period of the Day	Time of the Day	Temperature
greenhouse simulation	Morning	08:00 -12:00	37.600000
greenhouse simulation	Morning	08:00 -12:00	38.330000
greenhouse simulation	Morning	08:00 -12:00	38.550000
greenhouse simulation	Morning	08:00 -12:00	38.800000
greenhouse simulation	Morning	08:00 -12:00	38.910000
greenhouse simulation	Morning	08:00 -12:00	38.960000
greenhouse simulation	Morning	08:00 -12:00	38.980000
greenhouse simulation	Morning	08:00 -12:00	38.770000
greenhouse simulation	Morning	08:00 -12:00	36.580000
greenhouse simulation	Morning	08:00 -12:00	36.150000
greenhouse simulation	Morning	08:00 -12:00	35.990000
greenhouse simulation	Morning	08:00 -12:00	35.620000
greenhouse simulation	Morning	08:00 -12:00	34.680000
greenhouse simulation	Morning	08:00 -12:00	34.430000
greenhouse simulation	Morning	08:00 -12:00	34.230000
greenhouse simulation	Morning	08:00 -12:00	34.240000

greenhouse simulation	Morning	08:00 -12:00	34.190000
greenhouse simulation	Morning	08:00 -12:00	34.180000
greenhouse simulation	Morning	08:00 -12:00	34.170000
greenhouse simulation	Morning	08:00 -12:00	34.150000
greenhouse simulation	Afternoon	13:00 – 17:00	70.860000
greenhouse simulation	Afternoon	13:00 – 17:00	66.890000
greenhouse simulation	Afternoon	13:00 – 17:00	66.610000
greenhouse simulation	Afternoon	13:00 – 17:00	67.570000
greenhouse simulation	Afternoon	13:00 – 17:00	65.490000
greenhouse simulation	Afternoon	13:00 – 17:00	65.470000
greenhouse simulation	Afternoon	13:00 – 17:00	64.430000
greenhouse simulation	Afternoon	13:00 – 17:00	64.040000
greenhouse simulation	Afternoon	13:00 – 17:00	63.980000
greenhouse simulation	Afternoon	13:00 – 17:00	63.820000
greenhouse simulation	Afternoon	13:00 – 17:00	62.460000
greenhouse simulation	Afternoon	13:00 – 17:00	62.400000
greenhouse simulation	Afternoon	13:00 – 17:00	61.290000
greenhouse simulation	Afternoon	13:00 – 17:00	60.920000
greenhouse simulation	Afternoon	13:00 – 17:00	60.460000
greenhouse simulation	Afternoon	13:00 – 17:00	59.830000
greenhouse simulation	Afternoon	13:00 – 17:00	59.460000
greenhouse simulation	Afternoon	13:00 – 17:00	57.300000
greenhouse simulation	Afternoon	13:00 – 17:00	56.940000
greenhousesimulation	Afternoon	13:00 – 17:00	54.390000
greenhouse simulation	Afternoon	13:00 – 17:00	39.310000
greenhouse simulation	Afternoon	13:00 – 17:00	37.410000
greenhouse simulation	Afternoo	13:00 – 17:00	36.410000

	n		
greenhouse simulation	Afternoon	13:00 – 17:00	35.410000
greenhouse simulation	Afternoon	13:00 – 17:00	35.120000
greenhouse simulation	Afternoon	13:00 – 17:00	35.040000
greenhouse simulation	Afternoon	13:00 – 17:00	34.580000

Table 4: Temperatures retrieved in Factory simulation during a day.

These measurements can help people to monitor and control plants and animal products throughout the farm-to-plate cycle. In this context, we can imagine smart agriculture and smart food chain, reducing economic costs and increasing the food and production safety. This simulation showed us that from the time that the temperature is increased, it takes around 1hour and a half (in average)to get back to the normal levels. Thus, as a future work in this experiment, we could imagine that this sensor could send a signal to a connected device (e.g. air condition installed) in order the temperature to be reset in normal levels or a message to the owner of greenhouse in order them to come and open any doors or windows.

6.5 Use Case Scenario 5: Car

Another experiment included set of temperatures in a car, during summer period (August).

Figure 20: STM32F469I-DISCO along with BME680 in a car

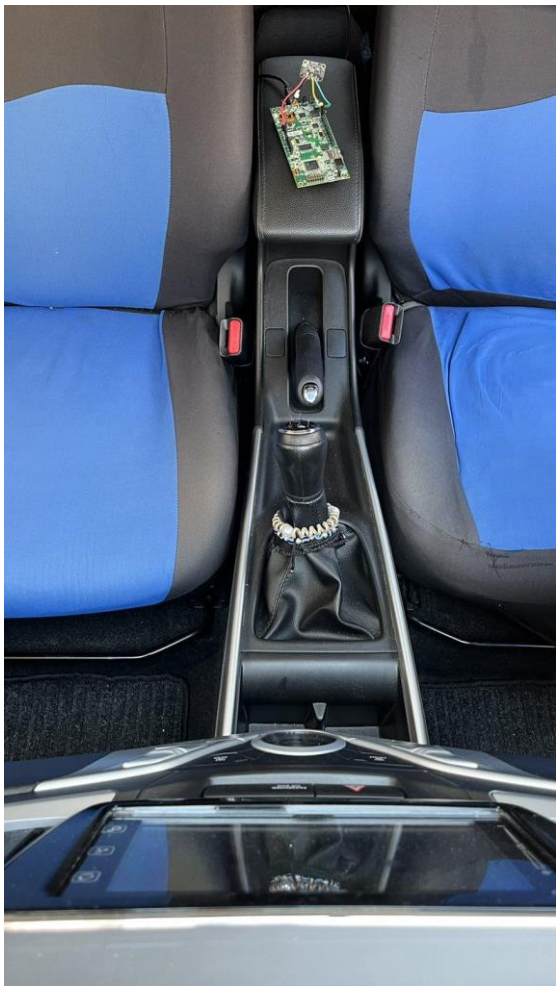

Figure 21: STM32F469I-DISCO along with BME680 in a car

The values (see Table 5) are retrieved in the morning (among 978.911, for simplicity we have presented some average values for a duration of 4 hours in the morning), in the afternoon (among 978.911, for simplicity we have presented some average values for a duration of 4 hours in the afternoon).

Place	Period of the Day	Time of the Day	Temperature
car	Morning	08:00 -12:00	24.400000
car	Morning	08:00 -12:00	25.220000
car	Morning	08:00 -12:00	25.540000
car	Morning	08:00 -12:00	25.850000
car	Morning	08:00 -12:00	26.070000
car	Morning	08:00 -12:00	26.200000
car	Morning	08:00 -12:00	26.330000
car	Morning	08:00 -12:00	26.410000
car	Morning	08:00 -12:00	26.490000
car	Morning	08:00 -12:00	26.540000
car	Morning	08:00 -12:00	26.600000
car	Morning	08:00 -12:00	28.420000
car	Morning	08:00 -12:00	28.820000
car	Morning	08:00 -12:00	29.110000

car	Morning	08:00 -12:00	29.140000
car	Morning	08:00 -12:00	29.220000
car	Morning	08:00 -12:00	29.600000
car	Morning	08:00 -12:00	29.920000
car	Morning	08:00 -12:00	29.400000
car	Morning	08:00 -12:00	29.220000
car	Afternoon	13:00 – 17:00	30.840000
car	Afternoon	13:00 – 17:00	30.850000
car	Afternoon	13:00 – 17:00	30.870000
car	Afternoon	13:00 – 17:00	30.880000
car	Afternoon	13:00 – 17:00	30.890000
car	Afternoon	13:00 – 17:00	30.900000
car	Afternoon	13:00 – 17:00	30.910000
car	Afternoon	13:00 – 17:00	30.930000
car	Afternoon	13:00 – 17:00	30.960000
car	Afternoon	13:00 – 17:00	30.930000
car	Afternoon	13:00 – 17:00	31.750000
car	Afternoon	13:00 – 17:00	31.790000
car	Afternoon	13:00 – 17:00	32.900000
car	Afternoon	13:00 – 17:00	33.420000
car	Afternoon	13:00 – 17:00	33.460000
car	Afternoon	13:00 – 17:00	35.290000
car	Afternoon	13:00 – 17:00	36.230000
car	Afternoon	13:00 – 17:00	36.390000
car	Afternoon	13:00 – 17:00	36.400000
car	Afternoon	13:00 – 17:00	36.760000

Table 5: Temperatures retrieved in basement during a day.

These measurements can help people to stabilize and control the conditions in a car. Having in mind, people that drive a lot during the day (e.g. transporters, taxi drivers, etc.), a significant change of temperature in their vehicles may cause respiratory problems or even death. Thus, as future work in this experiment, we could imagine that this sensor could send a signal to the driver (e.g. message in their cell-phone) in order to stop driving until the temperature to be reset in normal levels.

Chapter 7. Conclusions & Future Work

7.1. Conclusion

There are billions of devices connected making up the famous Internet of Things (IoT), and their number is increasing considerably. Internet can be used in various areas to its advantage human life and occupation. IoT is changing our world and the way we live However, IoT does not have a single architecture and this is where different types of malicious attacks at different levels of IoT are noticed.

The IoT devices are more vulnerable to attacks because safety measures cannot be taken sufficiently. Having studied the existing solution, the implementation that accompanies this Thesis tried to develop practical methods for the reliable execution of software based on "lightweight" execution and trust techniques. Moreover, dynamic measurement and control flow integrity methods with device connection key and code /data address encryption have been implemented, to ensure the integrity of the software at runtime on the IoT device.

More specifically, in Chapter 1, we provided a detailed Introduction; presenting the problem, the objective of this thesis, its contribution and its structure.

In Chapter 2, we described the Development environment, as well as the board that has been used for the implementation of the solution. In Chapter 3, we have presented the State of the Art, regarding IoT (namely, the development of Internet of Things and what is its network architecture), IoT Applications (e.g. Health, Smart Cities) and Security Issues regarding IoT (namely, fundamental concepts of the Security of Information Systems, the risks from malicious programs in combination with IoT, etc.).

In Chapter 4, we have presented zephyr OS and its contribution in this thesis. In Chapter 5, we have provided an extensive analysis of the Implemented solution (structure, code, etc.) as well as why C/C++ language is selected for the implementation. In Chapter 6, we exhibited the measurements and the results taken by the execution of the implemented solution and the contribution of this application throughout everyday-life scenarios.

In conclusion, this Thesis attempted to constitute a State of the Art forSecurity in IoT devices, proposing a reliable execution of software based on "lightweight" execution; supporting data address encryption for integrity of the software at runtime on the IoT devices that can be used by any people, as well as different scientific institutions and companies.

7.2. Future Work using advanced material and updated Programming code

The air quality – indoor and outdoor, is equally important. The research showed that we spend maximum time at indoor such as houses, hospitals, schools, etc. where various gases such as CO2, CO, Benzene, toluene and VOCs with humidity are well monitored.

The long term effect of these gases on the humans is possible to cause respiratory infections, lung cancer and heart diseases [39,40,41]. Therefore, indoor as well as outdoor air pollution detection is equally important.

Gas sensors provide information regarding air quality and allow people to take specific actions at the right place and time.

As a future work, we can imagine to make our application work in the context of pollutants in a factory; using a combination of sensors (see Figure X) and different sources of pollutants such as CO2, CO, Benzene, toluene and VOCs.

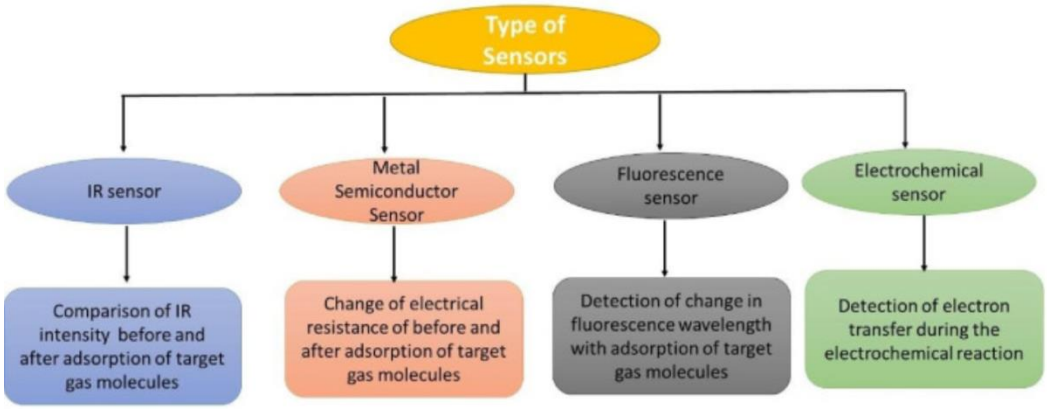

Figure 22: Types of Sensors

References

[1] https://visualstudio.microsoft.com

[2] https://www.st.com/content/st_com/en.html

[3] https://www.st.com/en/evaluation-tools/32f469idiscovery.html

[4] Gubbi et al. (2013) Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. Future Generation Computer Systems, pp.1645–1660.

Available from: https://www.sciencedirect.com/science/article/abs/pii/S0167739X13000241, Last accessed August 2022

[5] Towards a definition of the Internet of Things - Revision 1 - published on May 2015, Available from https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf , Last accessed August 2022.

[6] https://el.wikipedia.org/wiki/Διαδίκτυο_των_πραγμάτων, Last accessed August 2022

[7] https://www.tutorialspoint.com/internet_of_things/index.htm, Last accessed August 2022

[8] Dorsemaine B., et al., IEEE 2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies

Available from: https://ieeexplore.ieee.org/document/7373221, Last accessed August 2022

[9] Πάγκαλου Γ., Μαυρίδη Ι. (2002) Ασφάλεια Πληροφοριακών Συστημάτων και Δικτύων. Θεσσαλονίκη: Εκδόσεις Ανικούλα.

[10] Κάτσικα Σ., Γκρίτζαλη Δ., Γκρίτζαλη Σ. (2004) Ασφάλεια Πληροφοριακών Συστημάτων. Αθήνα: Εκδόσεις Νέων Τεχνολογιών.

[11] https://iot-analytics.com/10-internet-of-things-applications, LastaccessedAugust 2022

[12] https://aioti.eu/wp-content/uploads/2017/03/AIOTIWG01Report2015-Applications.pdf Last accessed, August 2022

[13]G. Kornaros, “Hardware-Assisted Machine Learning in Resource-Constrained IoT Environments for Security: Review and Future Prospective,” in IEEE Access, vol. 10, pp. 58603-58622, 2022, doi: 10.1109/ACCESS.2022.3179047.

[14] G. Kornaros, O. Tomoutzoglou, D. Mbakoyiannis, N. Karadimitriou, M. Coppola, E. Montanari, I. Deligiannis, G. Gherardi, “Towards Holistic Secure Networking in Connected Vehicles through Securing CAN-bus Communication and Firmware-over-the-Air Updating”, Journal of Systems Architecture (2020), vol. 109, pp. 101761,

ISSN 1383-
7621,doi: https://doi.org/10.1016/j.sysarc.2020.101761 (http://www.sciencedirect.com/science/article/pii/S1383762120300552)

[15]G. Trouli and G. Kornaros, "Automotive Virtual In-sensor Analytics for Securing Vehicular Communication," in IEEE Design & Test, vol.37, issue 3, pp. 91-98, print ISSN: 2168-2356,onlineISSN: 2168-
2364,https://ieeexplore.ieee.org/document/9001022, June2020,DOI: 10.1109/MDAT.2020.2974914

[16]O. Vermesan, M. Coppola, M. D. Nava, A. Capra, G. Kornaros, R. Bahr, E. C. Darmois, M. Serrano, P. Guillemin, K. Loupos, L. Karagiannidis and S. McGrath, “New Waves of IoT Technologies Research – Transcending Intelligence and Senses at the Edge to Create Multi Experience Environments”, online PDF , Chapter3 in book "Internet of Things – The Call of the Edge - Everything Intelligent Everywhere", River Publishers, DK, October 2020 (ISBN: 9788770221962, e-ISBN - 9788770221962 - Open Access), online https://european-iot-pilots.eu/internet-of-things-the-call-of-the-edge-everything-intelligent-everywhere/

[17]M. Coppola and G. Kornaros, “Automation for Industry 4.0 by using Secure LoRaWAN Edge Gateways”, in L. Andrade, F. Rousseau, (eds), Multi-Processor System-on-Chip, vol. 2., ISTE Ltd, London, and Wiley, New York, March 2021, https://iste.co.uk/book.php?id=1739, ISBN : 9781789450224

[18]G. Kornaros, E. Wozniak, O. Horst, N. Koch, C. Prehofer, A. Rigo, M. Coppola, "Secure and Trusted Open CPS Platforms", in book "Handbook of Research on Solutions for Cyber-Physical Systems Ubiquity", Editors: Norbert Druml, Andreas Genser, Armin Krieg, Manuel Menghin and Andrea Hoeller, IGI Global book series Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) (ISSN: 2327-3453; eISSN: 2327-3461), 2017

[19]F. Kolimbianakis and G. Kornaros, “Software-defined hardware-assisted isolation for trusted next-generation IoT systems”, In Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22), Association for Computing Machinery, New York, NY, USA, 139–146. 2022, https://doi.org/10.1145/3477314.3508378

[20]S. Leivadaros, G. Kornaros and M. Coppola, “Secure Asset Tracking in Manufacturing through Employing IOTA Distributed Ledger Technology”, in The 21th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID’21), 2nd Workshop on Secure IoT, Edge and Cloud systems (SIoTEC), May 10-13, 2021, Melbourne, Victoria, Australia

[21]D. Bakoyiannis, O. Tomoutzoglou, G. Kornaros and M. Coppola, “From Hardware-Software Contracts to Industrial IoT-Cloud Block-chains for Security, Privacy and Authenticity”, in Smart Systems Integration Conference, Virtual Edition 2021, April 27th – 29th, pp. 1-4, doi: 10.1109/SSI52265.2021.9467030

[22]G. Kornaros, D. Bakoyiannis, O. Tomoutzoglou, M. Coppola and G. Gherardi, "TrustNet: Ensuring Normal-world and Trusted-world CAN-bus Networking," 2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), Beijing, China, 2019, pp. 1-6. doi: 10.1109/SmartGridComm.2019.8909715

[23]D. Mbakoyiannis, O. Tomoutzoglou, and G. Kornaros, "Secure Over-the-air Firmware Updating for Automotive Electronic Control Units", Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), pp. 174—181, Limassol,Cyprus,2019,doi:10.1145/3297280.3297299,url: http://doi.acm.org/10.1145/3297280.3297299

[24]G. Kornaros and S. Leivadaros, "Securing Dynamic Firmware Updates of Mixed-Critical Applications", 3rd IEEE International Conference on Cybernetics (CYBCONF), 2017, pp. 1-7, doi:10.1109/CYBConf.2017.7985807

[25] https://zephyrproject.org/

[26] https://en.wikipedia.org/wiki/Zephyr_(operating_system)

[27] Smalley, Stephen & Carter, James. (2018). Security in Zephyr and Fuchsia. 10.13140/RG.2.2.15496.57603.

[28] https://www.zephyrproject.org/zephyrs-security-assessment/

[29] https://docs.zephyrproject.org/3.0.0/reference/usermode/memory_domain.html

[30] https://kernsec.org/wiki/index.php/Kernel_Self_Protection_Project

[31] https://www.comparitech.com/blog/information-security/what-is-sha-2-how-does-it-work/

[32] https://www.thesslstore.com/blog/difference-sha-1-sha-2-sha-256-hash-algorithms/

[33]https://www.brainbell.com/tutors/c/Advice_and_Warnings_for_C/Principles_Of_Reuse.html

[34] https://www.ibm.com/docs/en/cics-ts/6.1?topic=zos-flexibility-programming-language

[35] https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/

[36] https://www.log2base2.com/C/basic/introduction-to-c-language.html#:~:text=C%20is%20a%20portable%20programming,language%20or%20platform%20independent%20language.

[37] https://www.geeksforgeeks.org/clarity-and-simplicity-of-expressions/

[38] https://docs.zephyrproject.org/3.0.0/reference/libc/index.html

[39] Andrzej Chmielewski Monitoring, Control and Effects of Air Pollution Book, in Tech 9789533075266, Croatia (2011) Google Scholar

[40] Who Newsletter - Ten threats to global health in 2019

https://www.who.int/news-room/spotlight/ten-threats-to-global-health-in-2019

[41]W.Y. Yi, K.M. Lo, T. Mak, K. SL, Yee Leung, Mei Ling Meng, A survey of wireless sensor network based air pollution monitoring systems Sensors, 153 (2015), pp. 1392-31427