jamk.fi

Development of Secure IoT Based on Modern Microcontrollers

Juraj Haluška

Bachelor's thesis
May 2017
Technology, communication and transport
Degree Programme in Software Engineering

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s) Haluška, Juraj	Type of publication Bachelor's thesis	Date May 2017
	Number of pages 57	Language of publication: English
		Permission for web publication: yes

Title of publication
Development of secure IoT based on modern microcontrollers

Degree programme
Information and Communications Technology

Supervisor(s)
Mieskolainen, Matti

Assigned by
Kotkansalo, Jouko

Description

This paper describes the development process of a secured Internet of Things (IoT) system. Firstly, it analyses the available technologies and methods for development of secure IoT. It also describes the most suitable microcontrollers on the market for this purpose and discusses the basics of software development strategies. The most popular technologies and architectures were used to develop an IoT device with encrypted data transfer.

The IoT device was developed on a prototyping board Nucleo-F767ZI which runs on a microcontroller based on ARM architecture. The remote control of the device was implemented by exposing RESTful API through the custom implementation of an HTTP server and the whole communication was secured by SSL/TLS protocol. For SSL/TLS integration, an open source library mbed TLS was used. Humidity, temperature and atmospheric pressure sensors were connected to the device and the data captured by these sensors was stored on an SD card.

This data is available through RESTful API. The software of the microcontroller is based on mbed OS operating system with RTOS functionality. For a demonstration of RESTful API, the web application was developed. This application was programmed on the top of an Angular2 framework with UI library - PrimeNG.

The result of the thesis is an intermediate embedded system, which might be easily integrated into real IoT application where secured connectivity is required.

Keywords (subjects)
IoT, mbed OS, mbed TLS, security, ARM, microcontroller, REST, HTTPS

Miscellaneous

Contents

1 Introduction ... 3
1.1 Motivation ... 3
1.2 Objective ... 3
2 Microcontrollers ... 5
2.1 What microcontrollers are ... 5
2.2 History and presence ... 6
2.2 ARM and their portfolio ... 8
2.3 ARM Cortex-M series ... 10
2.4 Cortex-M7 core ... 11
2.4.1 Instruction set ... 11
2.4.2 Interrupt controllers ... 11
2.4.3 SysTick timer ... 12
2.4.4 Extensional buses ... 12
2.4.5 Debugger interfaces ... 13
2.4.6 Computation modules ... 13
2.4.7 Memory protection unit ... 13
2.5 Choice of microcontroller ... 14
3 Software of microcontroller ... 17
3.1 Common software strategies ... 17
3.2 mbed OS ... 22
3.2.1 Task management ... 23
3.2.2 I/O and interfaces ... 26
3.2.3 Networking ... 27
4 Network security ... 30
4.1 TLS ... 30
4.1.1 Roles ... 30
4.1.2 Structure ... 32
4.1.3 Session resumption ... 33
4.2 Entropy ... 35
4.3 HTTPS ... 36
5 Implementation ... 37
5.1 System architecture and technologies ... 37
5.2 Software on Nucleo board ... 40

5.2.1 Threads 40
5.2.2 TCP Sockets 41
5.2.3 mbed TLS Integration...42
5.2.4 HTTP Server 44
5.2.5 IP whitelist and user authentication 45
5.2.6 Sensors 47
5.3 Client application 49
6.1 Issues 51
6.2 Discussion 52
6.3 Summary 53
References 54

1 Introduction

1.1 Motivation

The Internet of Things (IoT) is increasingly set to become an inherent part of a human’s everyday life. Today, there is a great number of objects that are connected to the internet already. For example, household appliances, vehicles, health and sleep monitoring devices, weather stations, heating units, etc. The connection of elementary things used every day on a network provides many benefits and simplifications of a human’s life. As the popularity of IoT is rapidly rising, concerns have been raised about security and privacy. IoT devices could monitor sensitive information which no one wants to share with the unwanted subjects. Another issue with connecting things to the internet is the taking of control over the device by an unprivileged person. In some cases, this could result in catastrophic consequences. The rising popularity of IoT has a strong foundation in recent developments in microcontroller area. In the last decade, microcontrollers based on cores with ARM architecture have become very successful in areas of IoT. One of the main features of these microcontrollers is low power consumption, which is demanding due to continuous operation of IoT devices while providing relatively high computation power. Another significant advantage of ARM-based microcontrollers is a relatively low price and the availability of many kinds of prototyping boards. These boards give the developers’ ability of rapid product development.

1.2 Objective

This thesis focuses on the development of the IoT device driven by a modern microcontroller with the emphasis on security aspects. The first part of the thesis contains an analysis of microcontrollers available on the market and gives a brief overview of their hardware and architecture. There are also mentioned common strategies, how software for microcontrollers can be designed with or without the operating system. The second part of the thesis

describes confidential information transfer over the internet using TLS. The last and main objective of this thesis is the development of an embedded system based on prototyping board Nucleo-F767ZI from STMicroelectronics. This system shall provide a web server secured by TLS, a simplistic HTTPS API, and web interface for simple control of board’s peripherals.

2 Microcontrollers

2.1 What microcontrollers are

Microcontrollers are small, integrated circuits which can be considered as self-contained systems with a CPU, memory and peripherals (Heath 2002). Microprocessors known from classic personal computers do not contain memory and peripherals, which is why they need external ones connected via some kind of a bus system. On account of the fact that microcontrollers contain everything needed to form a computer, in most cases the only one necessity to make them work is to provide a software for them (Heath 2002). “Microcontrollers are designed to perform specific tasks. Specific means applications where the relationship of input and output is defined. Depending on the input, some processing needs to be done and output is delivered” (Choudhary 2017). Given that the applications of microcontrollers are very specific, they could be very optimized, and therefore they significantly reduce sizes and costs. They do not need such fast CPU as can be found in personal computers and they also have much smaller memory requirements. Since microcontrollers do not require rich resources, their power consumption is very small. This gives the ability to run microcontrollers from batteries and in some cases, they can run on battery uninterrupted for years.

Very important components of microcontrollers are peripherals. Peripherals give microcontroller the ability to interact with the surrounding environment and they can help the CPU with some functionality. Modern microcontrollers contain plenty of peripherals which can be used optionally due to the current needs of a particular application. Examples of the most common peripherals are listed as follows (Electronics hub 2015):

- GPIO - general pin input output
- ADC - analog to digital converter
- Serial protocol controllers - SPI, UART, I2C
- Timers
- Memories - RAM, Flash, EEPROM

There are also many other peripherals available, however, their presence in microcontrollers depends on the choice of particular models and also on the manufacturer. The availability of many kinds of peripherals, low power consumption, small size and high possibility of software optimization are the main factors why use microcontrollers in IoT applications.

2.2 History and presence

One of the most popular microcontrollers in history was the Intel 8051. This microcontroller was developed in 1980 by Intel Corporation and it was one of the first 8-bit microcontrollers. It can be seen in Figure 1. that this microcontroller was designed according to Harvard architecture. It means that it had separate memories for program and data, and those memories had their own address spaces. Later, the Intel 8051 was replaced by microcontrollers from many other companies such as Atmel, Infineon, NXP, Microchip (PIC), Texas Instruments. A widely recognized microcontroller architecture is 8-bit Atmel AVR. This architecture is still recognized nowadays because of its simplicity and also because Atmel AVR microcontrollers are used in a popular hobbyist platform - Arduino.

When 8-bit microcontrollers became no longer sufficient for certain tasks, 16-bit microcontrollers were developed and replaced them. They became the largest volume MCU category in 2011 (IC Insights 2013). Nowadays, there are microcontrollers available which architectures are based on 32-bit word size. Architectures with wider word size provide fewer restraints on resources, particularly memory and the width of registers used for doing arithmetic and logical operations. This feature gives the ability to perform real-time operations on high precision data. These days, 32-bit microcontrollers are mainstream. There are still some microcontroller manufacturers, which use their own 32-bit architectures, however the majority of them rely on processor cores which are designed by ARM Holdings. ARM based microcontrollers are the most popular ones. They are very favored because of extremely low power consumption and software support from ARM Holdings. Another advantage of ARM cores is portability. Because of standards (CMSIS) defined by ARM Holdings, it is much easier to move from one MCU to another. 32-bit microcontrollers with

ARM cores provide more options for developers while they can still maintain comparable power consumption with the old-fashioned 8-bit MCUs (STMicroelectronics 2017).

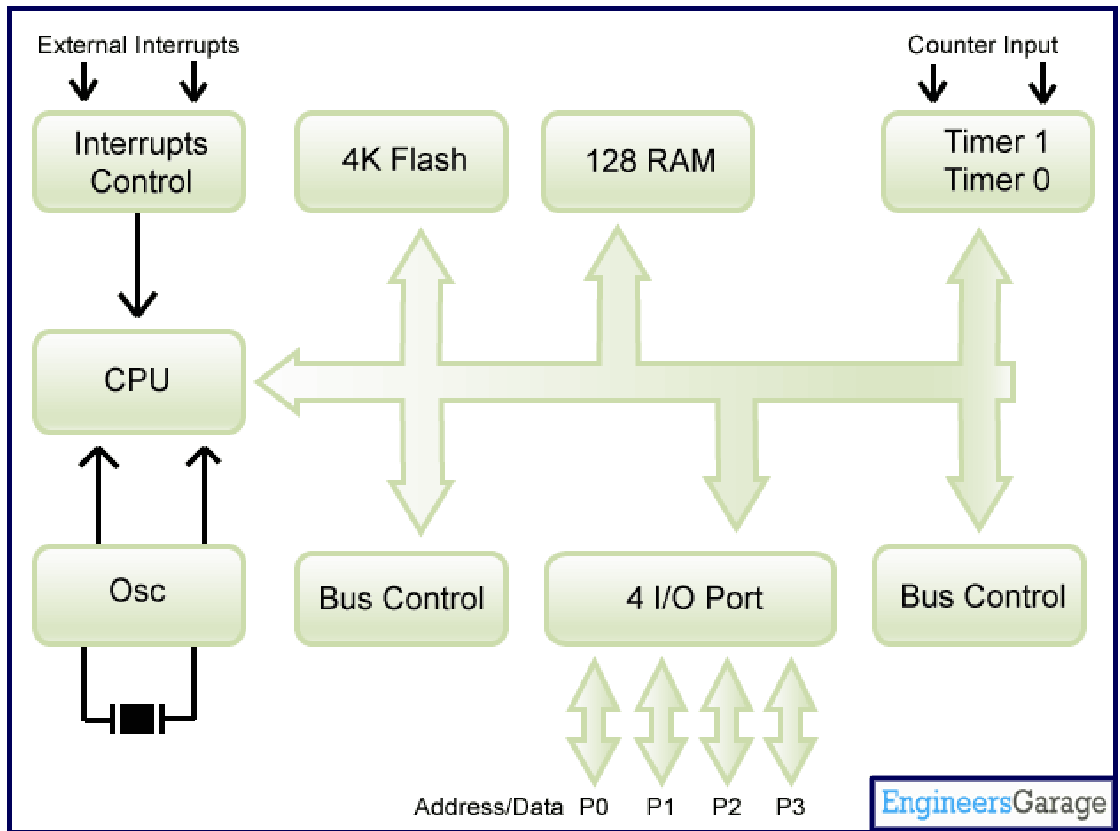

Figure 1. Block diagram of Intel 8051 (EngineersGarage 2017)

2.2 ARM and their portfolio

ARM is an acronym of Advanced RISC Machine. It is a designation of processor architecture which has become very famous in the last years. The high popularity of ARM architecture is caused by the low power requirements of processors based on it. With over 100 billion ARM processors produced as of 2017, ARM is the most widely used processor architecture in terms of quantity produced (Wikipedia 2017). ARM architecture is developed and designed by a British company called ARM Holdings. Their main business interest is in the design of ARM architecture based processor cores but they design software solutions also. ARM Holdings does not manufacture processor cores themselves, they just sell the design of cores to other companies, usually semiconductor manufacturers. Then, they embed processor cores from ARM Holdings into their own products (ARM 2017).

As mentioned earlier, ARM is RISC kind of processor. Thanks to RISC design approach, the actual implementation of a processor requires less transistors than typical CISC processors. This approach reduces costs, heat and power consumption (Wikipedia 2017). Most of the ARM cores use 32-bit word width, however, the newest variation of ARM architecture (ARMv8-A) supports also 64-bit word width. The current portfolio of ARM cores can be divided into four families based on different use cases (ARM 2017).

a) ARM Cortex-A Series

In this family, there are cores designated for high level performance applications. Cortex-A processors offer support for a range of full operating systems including Linux, as well as others requiring a memory management unit such as Android, Chrome and MontaVista (ARM 2017). The processors in this family use ARMv7-A or ARMv8-A architectures, including their variations, and some of them support multi-core designs. The intended applications of ARM Cortex-A Series are (ARM 2017):

- Smartphones
- Servers and networking

- Tablets and Readers
- Automotive
- Satellite receivers
- Wearables
- Home gateways
- Robotics

b) ARM Cortex-R Series

“The ARM Cortex-R real-time processors offer high-performance computing solutions for embedded systems where reliability, high availability, fault tolerance and/or deterministic real-time responses are needed. Cortex-R processors are used in products where performance requirements and timing deadlines must always be met. In addition, Cortex-R processors are used in electronic systems which must be functionally safe to avoid hazardous situations, for example, in medical applications or autonomous systems” (ARM 2017). Common applications of Cortex-R processors are (ARM 2017):

- Industrial
- Home
- Enterprise
- Cameras
- Storage
- Automotive
- Medical equipment
- Military

c) ARM Cortex-M Series

The processors in this family are smaller and come with the lowest power consumption of ARM products. These processors are optimized for deterministic real-time embedded processing and microcontroller application. ARM Cortex-M processors use 32-bit ARMv6-M, ARMv7-M and ARMv8-M architectures. The intended applications of processors from this family are (ARM 2017):

- Wireless sensors
- Smart watch
- Home automation
- Medical instruments
- Retail
- Industrial
- Smart watch
- Smart lighting

d) SecurCore

SecurCore is a family of 32-bit processors based on architectures ARMv6-M and ARMv7-M. They are derived from ARM Cortex-M processors, in addition, they contain anti-tampering mechanisms. Therefore, these cores are designed for use in highly secure applications. Some examples of use cases are listed below (ARM 2017):

- Advanced payment systems
- Transportation
- Electronic tickets, passports
- Smart and SIM cards

2.3 ARM Cortex-M series

As mentioned before, cores from this family are the most suitable for microcontrollers, which is the reason why this thesis gives more attention to this family. The ARM offers following cores from Cortex-M family (ARM 2017):

a) Cortex-M0, Cortex M0+, and Cortex-M23

These cores are for applications requiring minimal cost, power and area. Cortex-M0+ is the improvement of Cortex-M0 core. It is fully compatible with Cortex-M0, the difference is reduced power consumption, increased performance and added memory protection unit. It is not recommended to use older Cortex-M0 cores in new products. Cortex-M0/0+ use 32-bit ARMv6-M architecture. Cortex-M23 is the newest member of the Cortex-M family. This core uses ARMv8-M architecture, and it includes TrustZone technology.

b) Cortex-M3, Cortex-M4, and Cortex-M33

These cores are for all applications where a balance between 32-bit performance and energy efficiency is desirable. Cortex-M3 is the oldest member of the Cortex-M family. This core is useful when middle-level performance is required without the need for special features such as digital

signal processing or fast floating point operations. This features can be found in cores Cortex-M4 and Cortex-M33. Cortex-M33 also has built-in TrustZone technology (ARM 2017).

c) Cortex-M7

This is one of the newest cores available on microcontroller market. It was launched in 2014. Cortex-M7 is the most powerful core from Cortex-M family and it is fully binary compatible with Cortex-M3 and Cortex-M4 cores. It is possible to execute instructions from Cortex-M3/M4 on Cortex-M7. The main improvement is that it can run on a much higher frequency while maintaining double power efficiency in comparison with the older Cortex-M4 (ARM 2017).

2.4 Cortex-M7 core

2.4.1 Instruction set

This core supports execution of Thumb-1, Thumb-2 and special DSP instructions with saturated arithmetic. The Thumb-1 instruction set is the subset of the classic ARM instruction set. ARM instructions are 32-bit long; however, Thumb-1 are 16-bit long, which allows bigger code density. The disadvantage of Thumb-1 instructions is that some tasks which could be done by one ARM instruction take more Thumb-1 instructions. Thus, it is a compromise between the execution time and code density. Older cores were able to execute only one type of instructions, either Thumb-1 or ARM and it was necessary to switch between the two. Thumb-2 is technology which enables execution of Thumb-1 instructions with 32-bit long instructions also. This technology also adds conditional execution instruction. Thumb-2 technology combines code density of Thumb-1 instruction set and performance or ARM instruction set (Yiu 2010).

2.4.2 Interrupt controllers

In older cores, when the interrupt occurred, the software was needed to determine the starting address of an interrupt handler. This caused the

interrupt processing to be relatively slow. In every Cortex-M processor, there is NVIC which is a solution to that problem. Vectored interrupt support means that the interrupt handler address is saved in memory relating to its source. Thus, there is no need to perform additional software processing when an interrupt occurs. It is faster to process the interrupt request. Nested interrupt support adds prioritization to interrupt sources (My Hobbies 2017). "When an interrupt occurs, the NVIC compares the priority of this interrupt to the current running priority level. If the priority of the new interrupt is higher than the current level, the interrupt handler of the new interrupt will override the currently running task" (Yiu 2010). In Cortex-M7, there is available optional Wakeup Interrupt Controller. The function of this controller is to wake up core from a deep sleep when almost all processor modules are powered down (ARM 2017).

2.4.3 SysTick timer

SysTick is a configurable, countdown 24-bit timer. This timer might be used as a time base generator for real-time operating systems. Depending on the clock source, it could also serve as a counter. SysTick might be configured to generate an interrupt each time it reaches zero value; as an alternative, it can operate in polling mode. SysTick timer is optional and it is part of NVIC. (ARM 2017).

2.4.4 Extensional buses

Cortex-M7 processor is designed according to Harvard architecture. It has two separate buses, one for instructions (I-TCM) and one for data (D-TCM). Both of these buses are designed to provide tightly coupled memory for the processor core. This approach causes low-latency access, thus there is no need for cache memories. Peripherals such as GPIO, Timers and UART might be connected to the core with two AHB-Lite buses. AXI-M is an interface designed for connection of the external memory system. External memories are usually slower than those connected with TCM buses (e.g. Flash memory), thus, there is a need for cache memories, which is the purpose of I and D caches in the Cortex-M7 processor (Freescale 2015).

2.4.5 Debugger interfaces

Cortex-M7 processor has rich debugging possibilities. It contains ETM and ITM modules. ITM module enables printf style debugging, ETM gives the core real-time tracing of instruction and data. Another debugging module is DWT. “The DWT is an optional debug unit that provides watchpoints, data tracing, and system profiling for the processor” (ARM 2010). Debugging modules are connected to a debugger application via JTAG or SWD port. SWD is a newer alternative to JTAG. JTAG usually uses 5-pin port while SWD uses only two wires: one for bidirectional data, another for clock transmission. This technology is widely used in microcontrollers from STMicroelectronics and in addition to debugging, it is also used for memory flashing (ARM 2017).

2.4.6 Computation modules

Cortex-M7 is enhanced by DSP and FPU modules. The presence of these modules means that instruction set of Cortex-M7 is extended by special instructions belonging to these modules. Examples of the most common DSP instructions are multiply and accumulate. These modules are useful for fast algorithm performance in e.g. audio encoding, motor control, voice and image recognition, graphics, and data processing.

2.4.7 Memory protection unit

MPU is an inherent part of secure systems. This unit allows segmentation of memory and control access to a particular segment. “The main purpose of memory protection is to prevent a process from accessing memory that has not been allocated to it. This prevents a bug or malware within a process from affecting other processes, or the operating system itself” (Wikipedia 2017).

Modules mentioned above can be seen in following Figure 2. (ARM 2017):

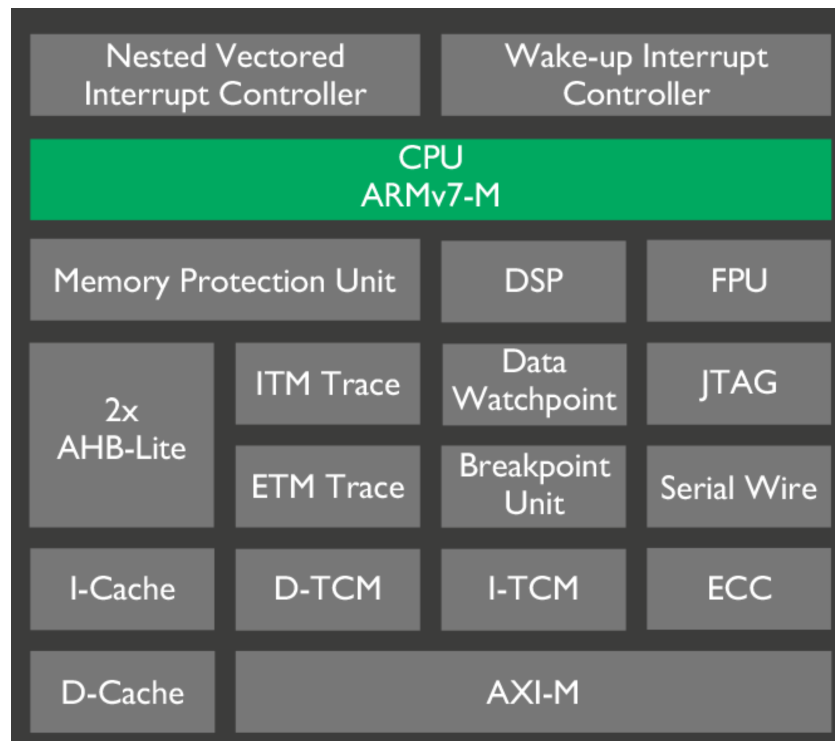

Figure 2. Cortex-M7 module diagram (ARM 2017)

2.5 Choice of microcontroller

The device developed in this thesis is required to run a web server with a secured connection. One step to get closer to secured web is by adding the encryption layer. Strong encryption algorithms need high computing rate. In the software of a web server, there is also need for fast string parsing, which is an operation consuming processor and memory. Because of high computing power, better power efficiency and availability of fast memory buses, a microcontroller with Cortex-M7 core was chosen for the practical part of this thesis.

ARM Cortex-M7 is quite a new core. As of April 2017, there are only three manufacturers which have caught up integrating Cortex-M7 in their products. The current manufacturers of Cortex-M7 based MCUs are Atmel (SAM E70, SAM S70, SAM V70), NXP (Kinetis KV5x) and STMicroelectronics (STM32 F7, STM32 H7). A significant role while choosing MCU was played by the availability of development materials and price. The chosen microcontroller is STM32F767ZI. This microcontroller is manufactured by STMicroelectronics,

and they also provide very cheap development board with it (Nucleo - F767ZI). MCU STM32F767ZI contains TRNG module, which is an integral part of systems, where encryption is involved. In addition, it also contains an Ethernet module. Therefore, there is no need to use an external Ethernet shield when implementing network connectivity. Another significant reason, why this board was chosen, is support of mbed OS. Barebone programming of ARM-based microcontrollers is not an easy task, especially for beginners. mbed OS significantly simplifies software development of embedded systems and in addition, it provides real-time operating system functionality, networking stack and security features (mbed 2017). Key features of STM32F767ZI (STMicroelectronics 2017):

- Core Cortex-M7 with 16 Kbytes I/D cache memories, frequency up to 216 Mhz, Dhrystone 2.1: 462 DMIPS/2.14 Mhz
- Up to 2 Mbytes of Flash memory organized into two banks allowing read-while-write
- SRAM: 512 Kbytes (including 128 Kbytes of data TCM RAM for critical real-time data) + 16 Kbytes of instruction TCM RAM (for critical real-time routines) + 4 Kbytes of backup SRAM
- Graphical accelerator, hardware JPEG coder/decoder, TFT and DSI display controllers
- Low power modes
- 3×12-bit 2.4 MSPS ADC - up to 24 channels, 2×12-bit D/A converters
- DMA, Ethernet, 4x I2C, 4x USART, 4x UART, 6x SPI, 3x CAN, 2x SDMMC, 2x USB controller, 18 Timers, 168 I/O ports
- True random number generator
- CRC calculation unit
- Real time clock
- 96-bit unique ID

Following figure (STMicroelectronics 2017) shows all blocks of STM32F767ZI:

Figure 3. STM32F767ZI module diagram (STMicroelectronics 2017)

3 Software of microcontroller

As mentioned in the first chapter, microcontrollers are small, self-contained systems the role of which is to perform specific tasks where the relationship between input and output is defined (Choudhary 2017). This makes the software of microcontrollers also very specific. In the past, the architecture of microcontrollers was much simpler than these days, therefore, they could be programmed in assembly language and without the need for an operating system. With the increasing complexity of microcontrollers, it has become impossible to write software purely in assembly language. These days, there are very good C/C++ compilers available, which simplifies the whole software design and enables to implement operating systems for microcontrollers. It should be mentioned that assembly language is still used, however, only in very hardware specific setup procedures, which cannot be written in C/C++.

This chapter focuses on common software strategies and possibilities of using the operating system in microcontrollers. The last section of this chapter is about mbed OS, which is a relatively new project in the world of embedded systems. mbed OS significantly simplifies and accelerates the development of ARM-based microcontrollers software.

3.1 Common software strategies

There are several strategies, how software of microcontroller can be designed. Choice of software strategy depends on complexity of particular application of microcontroller. There might be extremely simple systems, the task of which is just to respond when a button is pressed. On the other hand, there are more complex systems, e.g. systems which constantly monitor multiple parameters and need to adjust their outputs in real-time. Mentioned systems are very different, therefore their softwares are designed by different strategies. This chapter presents the most common software strategies used in MCUs.

a) Polling loop

This is the simplest strategy and it might be used in simple systems, the only task of which is to quickly react to some external events. An example of an event might be e.g. timer overflow, pressed the button, or incoming data. The principle of this strategy lies in repetitive checking if the event occurred. If the condition indicating an event is met, the microcontroller starts reacting to the event. After the reaction, the microcontroller is checking the condition again. The main disadvantage of this strategy is the waste of microcontroller's time while it is waiting for the condition to be met. Following figure is an example of simple polling loop implemented in C language.

```
while(1) {
   if (buttonPressed) {
      reaction();
      buttonPressed = !buttonPressed;
   }
}
```

Figure 4. Example of polling loop in (C language)

b) Interrupt-driven

This strategy is in most cases a better alternative to polling loop because polling (also called busy waiting) is replaced by interrupts. An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention (Wikipedia 2017). When an interrupt occurs, the processor will suspend the execution of the current code and start the execution of code associated with the source of the interrupt. This code is usually called ISR. The main advantage of this strategy is that it does not waste processing time by busy waiting. The main loop might perform some useful processing while it can be interrupted at any time when needed. This approach is also called foreground/background system. Another advantage over the polling loop is that responses to events can be prioritized thanks to the interrupt mechanism. The Figure 5. shows an example of interrupt-driven system. In the left column, there is a main task which can be interrupted by interrupt controller and routines ISR_1 or ISR_2 will be executed instead.

while(1) {
somethingUseful();
}
	void ISR_1 {
	...
	}
		void ISR_2 {
		...
		}

Figure 5. Example of interrupt-driven strategy in (C language)

c) Cyclic executive

This strategy can provide the illusion of simultaneity by taking advantage of relatively short processes in a continuous loop (Ayav 2017). Cyclic executive works only if the processes are short enough and their execution time is constant. As an example, a system with three processes (P1, P2, P3) can be considered where P1 needs to be executed every 20 ms. Cyclic executive might be used in this case only if the sum of durations of processes P1, P2, P3 is less than 20 ms. The Figure 6. shows how processes in cyclic executive are executed. The main loop of cyclic executive program can be seen in Figure 7.

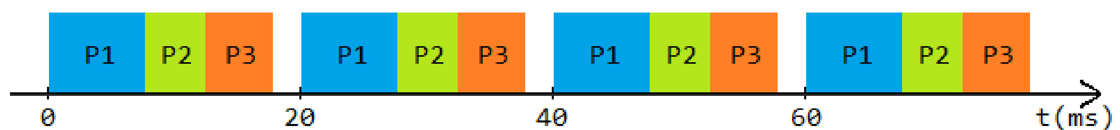

Figure 6. Diagram of process execution time in cyclic executive

```
while(1) {
    P1();
    P2();
    P3();
    synchroDelay();    // optional
}
```

Figure 7. Example of cyclic executive strategy in (C language)

d) Cooperative multitasking

Cooperative multitasking is a multitasking technique that enables two or more programs to cooperatively share the processing time and resources of the host processor (Technopedia 2017). This strategy is an extension of the cyclic

executive, however, the difference is that processes are coded in the state-driven fashion (state machines). Processes are executed the same way as in cyclic executive but each time when the process is executed, it performs only one phase which corresponds to a certain state of that process. This phase has to be short enough to allow other processes to perform their phases in acceptable time. This approach allows processes to be more complex, but it is more difficult to synchronize them. Following figure presents an example of the two independent tasks in cooperative multitasking strategy.

void P1() {
  switch (P1_state) {
    case START: … break;
    case REGULATE: … break;
    case FINISH: … break;
  }
}
	void P2() {
	  switch (P2_state) {
	    case START: … break;
	    case RECEIVE_DATA: … break;
	    case FINISH: … break;
	  }
	}

Figure 8. Example of processes in cooperative multitasking (C language)

d) RTOS

“Real time system is any system in which the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness” (Daintith, J., Wright E. 2008). The acceptable timelines are often also referred to as deadlines. An example of real time system might be a digital audio mixer. This device is used by sound engineers and musicians for live performances. Audio mixer receives an audio from multiple sources (e.g. microphones, musical instruments) and its task is to mix those sources into one or more channels. Mixed audio is then amplified and played back to the audience through speakers. To obtain an effect of a live show (that an audience will not notice a delay between what they see and what they hear), it is required to keep the delay between recording and playing back under 20 ms. Thus the time, under which the digital mixer must mix and process each bunch of samples, is 20 ms. This is the real time system the

deadline of which is 20ms. If this deadline is not met, the quality of live performance is being degraded. Real time systems could be classified into three categories:

1) Hard Real Time

In these systems, the processing has to be done before a deadline. If the deadline is not met, it will result in catastrophic consequences. As examples of hard real time might be e.g. air traffic control systems, car braking system, navigation system of missile, self-driving car (Ayav 2017).

2) Soft Real Time

Soft real time systems are those where missing a deadline causes a degradation of a system’s quality instead of a catastrophe. An example of this system might be a digital audio mixer, which was already mentioned previously (Ayav 2017).

3) Firm Real Time

These systems are similar to hard real time systems in a manner that they have hard deadlines defined, however, some low probability of missing a deadline can be tolerated. An example of this might be an automated assembly line (Ayav 2017).

From the included examples of real time systems above might be seen that they are often used in embedded applications. This is the point where real time operating systems occur. The real time operating system is a kind of operating system where tasks are scheduled in a fashion that they can meet their deadlines. The characteristics of real time system in an embedded application can be obtained by using one of the strategies mentioned in this chapter, however, real time operating systems simplify the software design. The common implementations of real time operating systems take care of process scheduling, IPC, memory allocation and memory protection, resource sharing and interrupt handling on software layer. Many of the available real

time operating systems offer additional features such as device drivers, file systems, or networking stacks. These real time operating systems are often referred to as full-featured real time operating systems. As mentioned, real time operating systems are often used in embedded applications and therefore they are optimized to run on microcontrollers. The characteristics of the operating system and optimization for microcontrollers usually make usage of real time operating systems convenient in applications where real time system characteristics are not even required. A list of the most popular real time operating systems follows below:

- FreeRTOS
- RIOT
- eCOS
- µC/OS-III
- RTLinux
- VxWorks
- Keil RTX
- QNX

3.2 mbed OS

Because of the high complexity of modern 32-bit microcontrollers, the development of barebone software for them has become a difficult task. There are several hardware-related configurations involved, e.g. clock distribution for peripherals, PLLs and frequency dividers, power distribution, bus configurations. Fortunately, there are solutions which abstract away hardware-related issues from programmers and even more, they also bring to a software features, which simplifies the whole development. One of the solutions is presented in this chapter and it is called mbed OS. mbed OS is a modern, open source operating system designated for IoT devices running on microcontrollers with ARM Cortex-M cores. It is collaboratively developed by ARM Holdings and their partners. mbed OS is a full-featured RTOS programmed in C/C++. It contains a low level security kernel called mbed uVisor, RTOS based on Keil RTX, peripheral drivers, drivers for the external devices and TCP/IP stack. For a quick and easy development of a secure IoT application, mbed OS supports many development boards from several manufacturers. These properties had a significant impact on the choice of the development board for the practical part of this thesis.

3.2.1 Task management

mbed OS provides a comprehensive API for task management. An important part of this API is RTOS, an implementation of Keil RTX which allows the creation of programs simultaneously, performing multiple functions and helps to create better structured and more easily maintainable applications (Keil 2017). RTOS in mbed OS handles the creation and destruction of threads and safe inter-thread communication as well. It is composed of the following classes (mbed 2017):

a) Thread

“The Thread class allows defining, creating and controlling thread functions in the system“ (mbed 2017). The thread can be in one of the following states (Figure 9.): Running, Ready, Waiting and Inactive. It is possible to configure the stack size and priority for each thread through their constructor (mbed 2017).

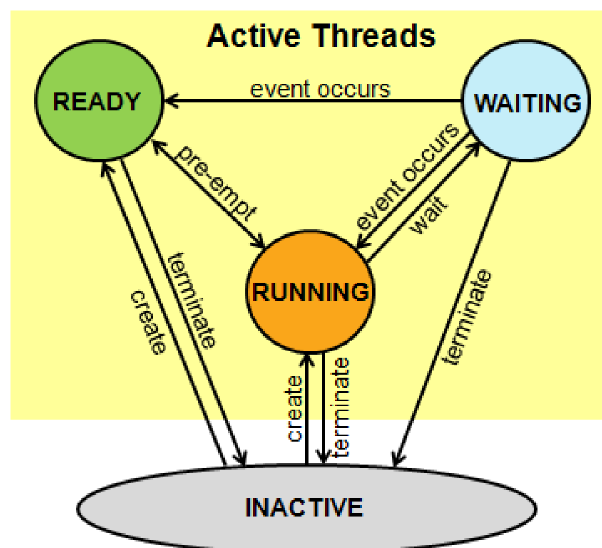

Figure 9. States of thread in mbed OS RTOS (mbed 2017)

b) Mutex

Mutex (mutual exclusion) is a locking mechanism of the operating system to protect shared resources. It ensures thread synchronization. Resource (e.g. variable, a bunch of memory, etc.) might be locked by mutex; it can be then

modified only by the thread which has locked it. Mutex operations in mbed OS are: lock, trylock and unlock. These operations can be called only by threads. The following figure (Figure 10) presents mutex mechanism and its operations.

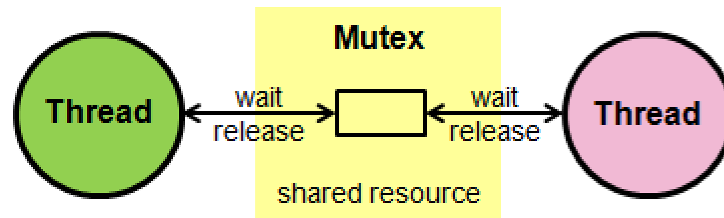

Figure 10. Mutex mechanism (mbed 2017)

c) Semaphore

Semaphore is a mechanism for sharing a pool of resources of the same kind between multiple threads. The number (size of the pool) of resources which will be protected by semaphore is defined via its constructor. There are two operations associated with the semaphore: wait and release. Calling wait operation will appropriate one resource, or it will wait if the resource is not available. The release operation will release the resource so that it can be used by the other threads. Figure 11 shows how multiple threads can control the access to shared resources using the semaphore.

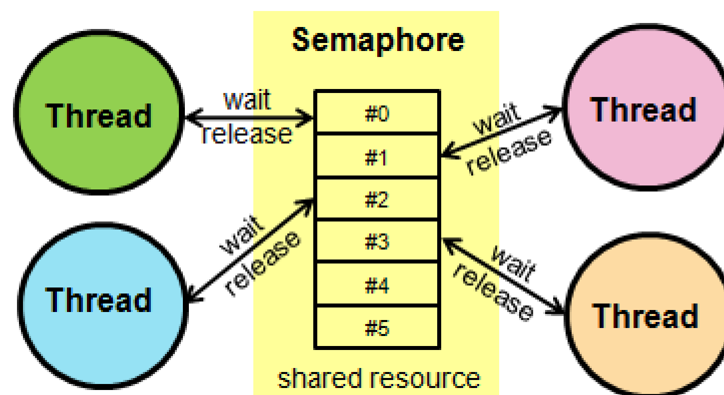

Figure 11. Semaphore mechanism (mbed 2017)

d) Signal

A signal is a simple tool for notification of events between threads. There are only two calls defined: wait and signal. The parameter of this calls is a flag

which is the 4-byte integer. If one thread is waiting for a particular flag, other threads can interrupt that waiting by calling the signal on the same flag.

e) Queue and Mail

These classes provide a safe way of data transfer between threads. The data transfers between threads are often also referred to as messages. The queue allows sending messages which are 4-byte long (pointers or integers). “Mail works like a queue, with the added benefit of providing a memory pool for allocating messages (not only pointers or integers)” (mbed 2017).

In addition to RTOS specific functions, mbed OS has other built-in functions which might be used for task management:

a) Event loop

The event loop is a mechanism that can be used to defer the execution of the code to a different context. “In particular, a common use of an event loop is to postpone the execution of a code sequence from an interrupt handler to a user context” (mbed 2017). The ISRs are not thread-safe, thus it is recommended to divide them into two sections. One section, which is a part of ISR, should contain only time critical operations. The second section, the execution of which could be scheduled (some delay tolerance), should be deferred from ISR to user context using the event loop.

b) Time-based classes

The ticker class serves for calling a function repeatedly in a specified time interval. For calling a function only once after a specified delay, there is the Timeout class available. The last time-based class for task management is a Wait. “When Wait is called inside thread, the OS scheduler will put the current thread in waiting state, allowing another thread to execute” (mbed 2017). It is worth mentioning that there are other classes which work with the time, however, their functionality is not for task management. These classes are Time and Timer. Time class provides an interface of hardware real-time clock

and it has functions for date conversions and formatting. For measurement of time intervals, there is Timer class available. Timer uses 32-bit signed integer for counting microsecond steps (mbed 2017).

3.2.2 I/O and interfaces

The following table is an outline of API abstraction in mbed OS:

Table 1. I/O API in mbed OS (mbed 2017)

Type	**Class**	**Description**
Analog I/O	AnalogIn AnalogOut	read the voltage of an analog input pin set the voltage of an along output pin
Digital I/O	DigitalIn DigitalOut DigitalInOut	read a single digital input pin write to a single digital output pin read and write to a single bidirectional digital pin
Bus I/O	BusIn BusOut BusInOut	read multiple pins as a single value write a single value to multiple pins read and write to multiple bidirectional pins
Port I/O	PortIn PortOut PortInOut	read pins from port as a single value write a single value to pins in the port read and write to multiple bidirectional pins in the port
Interrupt	InterruptIn	triggers an event when a digital input pin changes value
PWM	PwmOut	controls the frequency and duty cycle of PWM

mbed OS has also built-in drivers for following digital interfaces:

Table 2. Digital interfaces API in mbed OS (mbed 2017)

Interface	Class
UART	Serial
SPI	SPI, SPISlave
I2C	I2C, I2CSlave
CAN	CAN

3.2.3 Networking

The mbed OS has built-in drivers for Ethernet and Wi-Fi modules of many development boards. Communication with those modules is accomplished by corresponding classes - EthernetInterface and WifiInterface. As a default TCP/IP stack, mbed OS uses a lightweight open source project designed for embedded devices - lwip. On the top of the TCP/IP stack, mbed OS provides a simple socket API. “It’s a class-based interface, which should be familiar to users experienced with other socket APIs” (mbed 2017). The socket API works in the same way with both of the interface classes mentioned above. There are three important classes in socket API:

1) UDPSocket

“The UDPSocket class provides the ability to send packets of data over UDP, using the sendto and recvfrom member functions. Packets can be lost or arrive out of order, so we suggest using a TCPSocket (described below) when guaranteed delivery is required” (mbed 2017).

2) TCPSocket

“The TCPSocket class provides the ability to send a stream of data over TCP. TCP sockets maintain a stateful connection that starts with the connect member function. After successfully connecting to a server, you can use the send and recv member functions to send and receive data (similar to writing or reading from a file)” (mbed 2017).

3) TCPServer

"The TCPServer class provides the ability to accept incoming TCP connections. The listen member function sets up the server to listen for incoming connections, and the accept member function sets up a stateful TCPSocket instance on an incoming connection" (mbed 2017).

3.2.4 Development environment

There are basically three possible options, how microcontroller's software based on the mbed OS can be developed. The first development option is using the mbed CLI. "mbed CLI is the name of the ARM mbed command-line tool, packaged as mbed-cli. mbed CLI enables Git- and Mercurial-based version control, dependencies management, code publishing, support for remotely hosted repositories (GitHub, GitLab and mbed.org), use of the ARM mbed OS build system and export functions and other operations" (mbed 2017). The mbed CLI allows a simple import of libraries from the mbed repository, compilation using one of the available compilers (GCC ARM, ARM Compiler 5, IAR), and testing and source control, while the developers can still use their favorite text editor. The practical part of this thesis was developed using mbed CLI with a Geany text editor.

The second option is called mbed Online Compiler. It is the simplest way how to develop software with the mbed OS. mbed Online Compiler is in-browser IDE which provides the same functionality as mbed CLI with text editor added. Using the mbed Online Compiler avoids the need to install anything on the local computer, which is its main advantage.

The third option involves third-party toolchains. It is possible to export project files from mbed CLI and mbed Online Compiler to many of available IDEs. If the developer does not use IDE, or it is not supported, there is also the possibility to generate GNU Makefile. Then it can be used with one of the three mentioned compilers above. The list of supported IDEs (mbed 2017) is as follows:

- Keil uVision5
- IAR Systems
- Eclipse CDT/GNU ARM
- Atmel Studio
- Simplicity Studio
- DS-5
- LPCXpresso
- Kinetis Design Studio
- CooCox CoIDE
- e2studio

4 Network security

The internet is a place full of threats. The connection of objects to the internet makes them possible victims for attackers. When connecting objects to the internet, it is very important to ensure that the transmitted data cannot be read by unwanted subjects and that parties of communication are what they claim to be. There is a standardized way how to ensure security over the network. It is called Transport Layer Security (TLS) and it is used in practical part of this thesis to provide data protection and the identity of an IoT device.

4.1 TLS

4.1.1 Roles

TLS is a cryptographic protocol that provides security for devices connected to the network. TLS is a follower of the older SSL protocol and its newest version is TLS 1.2. “The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications“(RFC 5246 2008). It is commonly used to secure web protocol (HTTPS), email communication, VoIP, Instant Messaging, remote login (SSH), etc. When the connection is secured by TLS, it has one or more of the following properties (Wikipedia 2017):

a) Privacy

The privacy of data is reached by encryption. Encryption is a mechanism which encodes information or messages so that only authorized parties can access it. “Encryption works by using a mathematical formula called a cipher and a key to convert readable data (plain text) into a form that others cannot understand (ciphertext). The cipher is the general recipe for encryption, and the key makes the encrypted data unique. Only parties with the unique key and the same cipher can unscramble it” (Sans 2011). For encryption, TLS offers several symmetric cipher algorithms, e.g., AES, Blowfish, Camellia, DES/3DES, XTEA, RC4 (mbed TLS 2017).

b) Identity

The identity of communicating parties is achieved by public-key cryptography. This property is optional, however, it is usually used at least by one party. TLS also uses public-key cryptography for the exchange of symmetric encryption key. Examples of these public-key cryptographic algorithms are RSA, DSA, D-H.

c) Integrity

For integrity check, TLS inserts to each message an authentication code (MAC). It is a mechanism that makes it possible to determine whether the data has been changed by an unwanted subject. MAC is a sort of checksum and it uses hash functions (e.g., SHA-1).

“One advantage of TLS is that it is application protocol independent. Higher-level protocols can layer on top of the TLS protocol transparently” (RFC 5246 2008). Application protocol independent means that TLS is inserted between the application protocol (HTTP, SMTP, etc.) and the transport protocol (TCP). Figure 12 illustrates that TLS is represented as a part of the application layer in the common TCP/IP model.

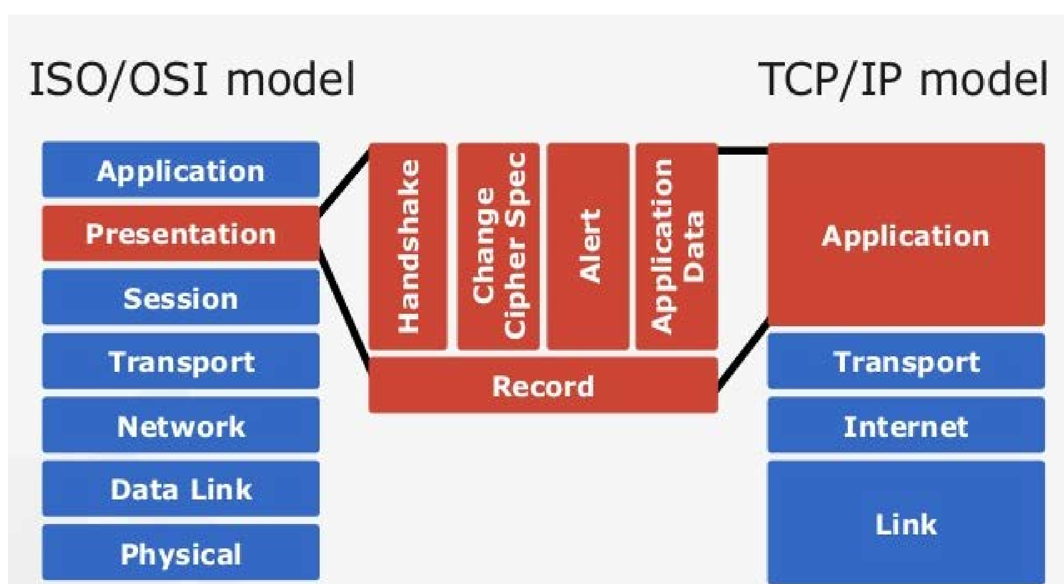

Figure 12. ISO/OSI and TCP/IP models with TLS (Luedtke 2012)

4.1.2 Structure

"The TLS protocol is composed of two layers: the TLS Record Protocol and the TLS Handshake Protocol" (RFC 5246 2008).

a) TLS Record Protocol

This layer communicates directly with the transport layer of TCP/IP model. The role of this layer is to process data from higher layers and to transmit them to the transport layer. The data processing operations performed by this layer are listed below (RFC 5246 2008):

- fragmentation/defragmentation
- compression/decompression
- MAC computation and verification
- encryption/decryption

b) TLS Handshake Protocol

When the communication between a client and server begins, they must exchange and agree on the protocol version and cryptographic algorithms. The public-key encryption is used to transfer shared keys and optionally, one or both of communicating peers are authenticated. These procedures are handled by TLS Handshake Protocol, which works on the top of TLS Record Protocol. The steps involved in TLS Handshake Protocol (RFC 5246 2008) are listed as follows:

- "Exchange hello messages to agree on algorithms, exchange random values, and check for session resumption.
- Exchange the necessary cryptographic parameters to allow the client and server to agree on a premaster secret.
- Exchange certificates and cryptographic information to allow the client and server to authenticate themselves.
- Generate a master secret from the premaster secret and exchanged random values.

- Provide security parameters to the record layer.
- Allow the client and server to verify that their peer has calculated the same security parameters and that the handshake occurred without tampering by an attacker” (RFC 5246 2008).

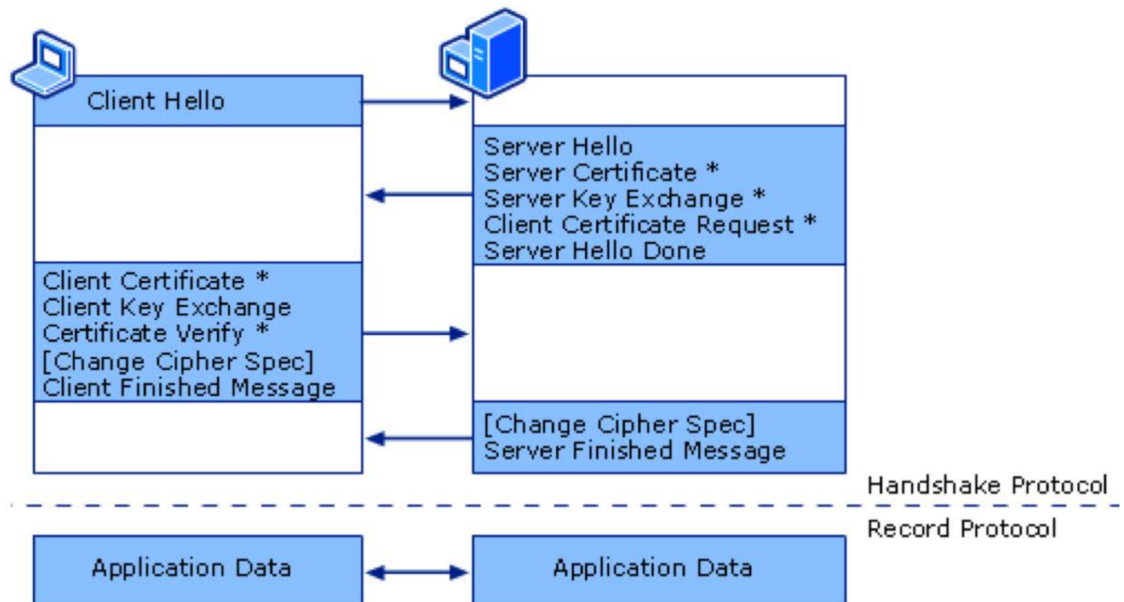

Figure 13. TLS Handshake Protocol (Zoompf 2017)

If the handshake was successful, the TLS Record Protocol will set up the exchanged parameters and the TLS session is created. Now, the application protocol (HTTP, etc.) can start using the secured TLS Record Layer directly.

4.1.3 Session resumption

The TLS Handshake Protocol involves the usage of an asymmetric cipher (public-key cryptography). Asymmetric cipher algorithms are more difficult than symmetric and require a more computing power. This brings a significant latency to the communication. For example, web browsers usually make multiple requests to the server in a row. Each request performs its own handshake and if there are more clients connecting to the same server, it will overload the server and will lead to DoS. In order to increase computational performance and to prevent DoS, TLS contains a mechanism called session resumption. This mechanism ensures a much faster handshake because

performing the asymmetric cipher algorithm at each request is no longer needed. There are two ways of session resumptions in TLS (Lin, Z. 2015).

a) Session ID resumption

“Resuming an encrypted session through a session ID means that the server keeps track of recently negotiated sessions using unique session IDs. This is done so that when a client reconnects to a server with a session ID, the server can quickly look up the session keys and resume the encrypted communication” (Lin, Z. 2015). The disadvantage of session resumption by session ID is that the server has to store information according to the particular session. This disadvantage takes place only in cases where there is a significant number of clients using the session resumption of which the server has to keep track.

b) Session ticket resumption

In order to prevent the server from keeping session information to save its memory, an alternative to session ID resumption was developed. It is called session ticket resumption and the principle of this way is that the session information is stored on the client side instead of the server. “A session ticket is a blob of a session key and associated information encrypted by a key only known by the server. The ticket is sent by the server at the end of the TLS handshake. The clients supporting session tickets will cache the ticket along with the current session key information. Later the client includes the session ticket in the handshake message to indicate it wishes to resume the earlier session. The server on the other end will be able to decrypt this ticket, recover the session key and resume the session” (Lin, Z. 2015).

4.2 Entropy

Cryptographic algorithms use random number generators for the creation of keys. The whole power of cryptographic algorithms lies in the randomness of numbers provided by random number generators. Entropy is a term which expresses unpredictability or randomness of some system. Random number generators use entropy sources to generate random numbers. If the random number generator uses weak entropy, its output can be predicted and the security of the system can be compromised by attackers. Random number generators can be divided into two categories (Hoffman, Ch. 2014):

- Pseudo Random Number Generator (PRNG)
- True Random Number Generator (TRNG)

PRNGs use deterministic algorithms with some kind of seed to generate the random number. Since the deterministic algorithms are used, the number which was generated is not truly random. The output sequence of PRNG can be predicted, which makes it unusable in cryptographic algorithms. On the other hand, PRNGs are relatively fast and they can find their usage in many other applications (games, simulations, etc.) (Hoffman, Ch. 2014).

For cryptographic purposes, TRNGs are used. TRNGs produce sequences of numbers which are unable to predict. They are also called nondeterministic RNGs. TRNG depends on some unpredictable entropy source outside of any human control (STMicroelectronics 2016). These entropy sources are usually some physical events or phenomena, which behave randomly. As an example of these sources might be cosmic radiation, radioactive decay of an atom, and radio frequency noise. Due to the physical nature of these entropy sources, they are usually presented by some electronic device. Another example might be how Linux kernel generates a random number. Linux kernel combines multiple sources of entropy such as keyboard and interrupt timings, hard drive operations, noise in audio input, etc., which are hardly-predictable (Linux kernel 2005).

In general, the generation of true random number takes more time than the

generation of the pseudo random number. This is because truly random numbers are generated by measuring entropy, not with computing by some algorithm as it is done in pseudo random number generators. Slow generation of true random numbers would be especially true in microcontrollers, which suffer from lack of entropy sources in comparison with personal computers. Fortunately, microcontrollers designated for the usage in applications where security is demanded contain hardware modules for true random number generation. The microcontroller, which was chosen for the practical part of this thesis (STM32F767ZI) contains a true random number generator based on an analog circuit. “This circuit generates a continuous analog noise that will be used on the RNG processing in order to produce a 32-bit random number. The analog circuit is made of several ring oscillators whose outputs are XORed. The RNG processing is clocked by a dedicated clock at a constant frequency and for a subset of microcontrollers, it can also be clocked with a different value of frequency” (STMicroelectronics 2016). The hardware TRNG module in STM32F767ZI is used in the practical part of this thesis as a source of random numbers for TLS.

4.3 HTTPS

The device developed in this thesis communicates with the surrounding world through HTTPS, which is HTTP protocol secured by SSL/TLS mentioned in section 4.1. The default TCP/IP port of HTTPS communication is 443 and the URI uses ‘https’ protocol identifier instead of ‘http’. HTTPS protocol was chosen for this thesis because it gives enough security and it is relatively simple to implement on embedded systems. However, the main reason why HTTPS was chosen is that it is very universal. The device can be controlled directly through the web browser or mobile application using a uniform API. The web browser application used for the control of the IoT device was implemented as a part of this thesis.

5 Implementation

5.1 System architecture and technologies

As mentioned in the previous chapter, the HTTPS protocol was chosen for the communication between the client and server. The server is implemented on a Nucleo board with mbed OS, and the board exposes its functionality to the client through the RESTful API. The data exchanged between the client and server is in JSON format. The mentioned technologies were chosen because they are relatively simple and easily integrated into many of the modern applications. The client application developed in this thesis is web-browser based, however, the usage of the mentioned technologies allows to easily implement alternative client applications on the other platforms such as Android, iOS, etc. Figure 14 represents the top layer architecture of the system:

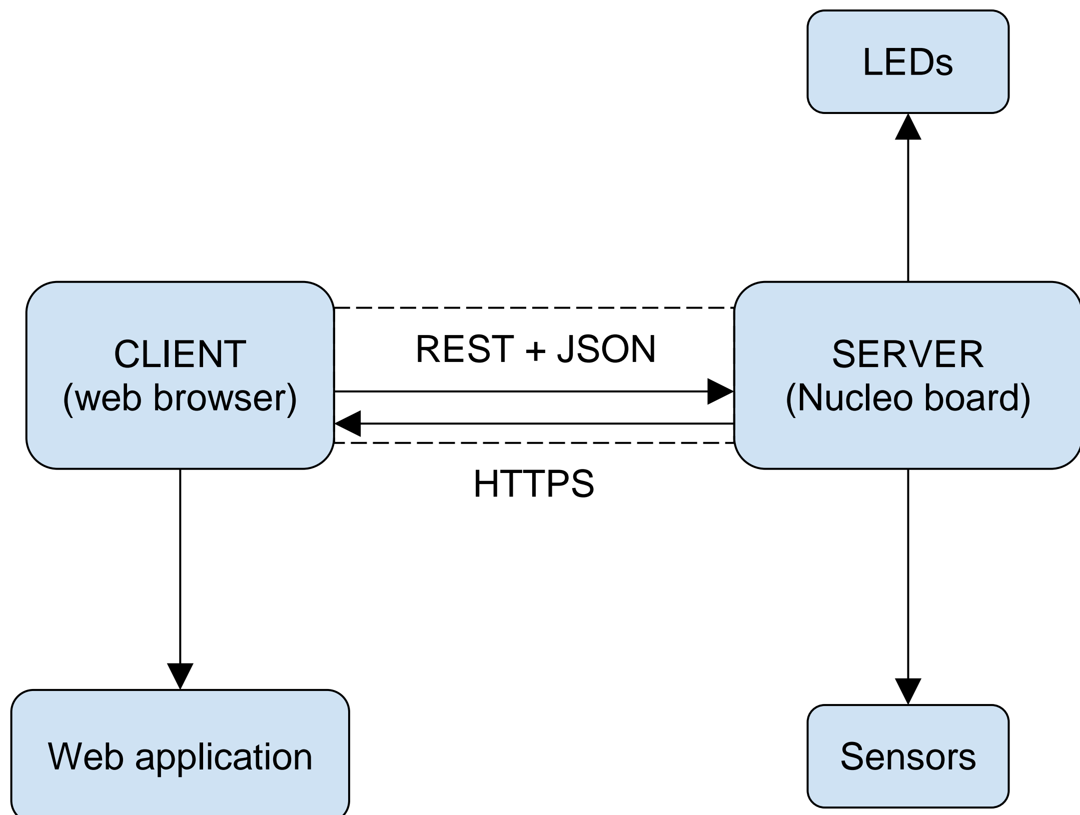

Figure 14. Diagram of system architecture

The web application is not provided to the client by a server implemented on the Nucleo board. The role of that server is just to expose its functionality by the RESTful API to the client. As shown in Figure 14, the web application is not a part of the server nor even a client. The web application might be stored on the client side, however, it might also be served to the client by an external web server.

“A RESTful API is a remote API that follows the REST style of software architecture” (McVetta 2012). REST stands for representational state transfer. The REST architecture can be used with other protocols than HTTP, however, it is mostly used with it. The REST API exposes resources of the server in a logical way, identified by the uniform resource identifier (URI). The URI structure of REST API usually represents the logical layout of the resources on the server side. For example, on a file server, it might represent the directory structure of the underlying file system. Data transferred by REST API is represented in various formats. The most common formats are JSON and XML which can easily model the objects of an underlying system. The REST API is stateless, which is a very important characteristic. Stateless means that the server does not keep information about the client. “Each request from any client contains all of the information necessary to service the request, and any session state is held in the client” (McVetta 2012). The REST API used on the web uses HTTP methods corresponding to CRUD operations. CRUD operations with corresponding HTTP methods are listed as follows (RestApiTutorial 2017):

Table 3. CRUD operations and corresponding HTTP methods

CRUD Operation	**HTTP Method**
Create	POST
Read	GET
Update	PUT / PATCH
Delete	DELETE

The following table, Table 4 shows an example on how IP addresses are manipulated in a server's whitelist using REST API (more details on whitelist implementation will be given further in this thesis).

Table 4. REST API for manipulation of IP addresses in server's whitelist

HTTP Method	**URI**	**Description**
GET	https://server/ip	Get list of all IP addresses in whitelist
POST	https://server/ip/192.168.1.1	Add IP address 192.168.1.1 to whitelist
DELETE	https://server/ip/192.168.1.1	Remove IP address 192.168.1.1 from whitelist
PUT / PATCH		not implemented

The data transferred between client and server using HTTP protocol is in the text form. The client application in this thesis is web-browser based and is programmed in JavaScript. It is very favorable to use JavaScript Object Notation (JSON) format for data exchange between the client and server because JSON is just a text format and JavaScript objects are easily converted to/from JSON. "JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate" (JSON 2017). "JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language" (JSON 2017). The implementation of JSON in the server on Nucleo board is not a full implementation according to official document (RFC 7159) describing JSON format. Many of available open-source JSON parsers are too heavy for use in microcontrollers and use dynamic memory allocation. On account of the fact that the format of all JSON

objects received by the server is known in a compile-time, a custom simplistic JSON parser was developed for the purposes of this thesis. This custom parser supports only non-nested key-value pairs, while it still maintains JSON formatting. An example of the JSON data acceptable by the server in this thesis can be seen in Figure 15:

```
{
  "led1":true,
  "led2":false,
  "led3":true
}
```

Figure 15. Example of JSON used in the server developed on Nucleo board

5.2 Software on Nucleo board

5.2.1 Threads

The software on Nucleo board performs two tasks: serving content to the client via HTTPS, and reading and storing sensor values. The thread mechanism of mbed OS was utilized to attain multitasking. Currently, there are three threads implemented in software of Nucleo board:

- Main thread
- Server thread
- Sensor thread

The main thread is a thread which must be present in every mbed OS application. This thread is automatically executed after a system boot-up by mbed OS. The only role of this thread is to initialize global variables and to execute another two threads (Server and Sensor thread). The whole networking logic is implemented in the Server thread. The sensor thread reads the sensor's values at the specific time interval and stores them in the SD card's flash memory. Each of these threads is divided into two parts - initializations and infinite loop.

5.2.2 TCP Sockets

mbed OS socket mechanism was used for networking. There are two socket classes used to accomplish client-server communication. One class represents server (TCPServer), the other one represents the client (TCPSocket). Before the sockets can be used, the Ethernet interface has to be configured and connected. These steps are performed in the initializations part of the Server thread. The configuration of Ethernet interface involves setting up of network parameters such as IP address, network mask and gateway. After the network interface has been successfully configured and connected, it can be used by the TCPServer socket by calling the open method with that network interface as a parameter. After opening this socket, it has to be bound to the configured IP address and a TCP port. As soon as the application layer used on the top of this socket mechanism is HTTPS, port 443 is used. When the TCPServer socket has been successfully bound to an IP address and a TCP port, it can start listening for a client connection. Listening is realized in the infinite loop of the Server thread. When the client has connected to the server, the accept method will set-up the TCPSocket instance for the client and save the client’s IP address in the SocketAddress structure. Now, the communication with the client can be done by calling recv and send methods on the client’s TCPSocket instance. The following chart represents the socket utilization:

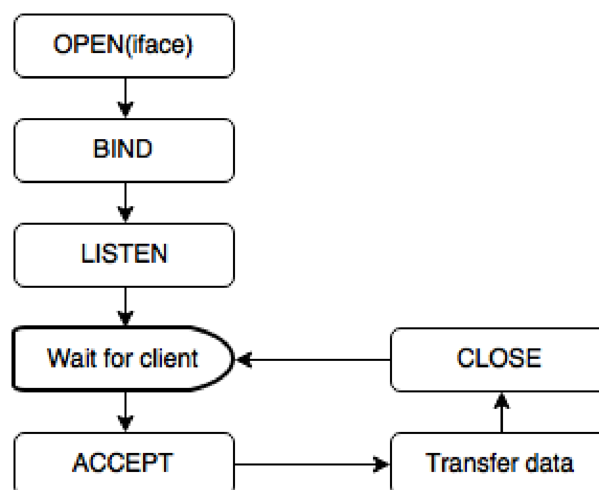

Figure 16. Flowchart of socket mechanism utilization in Server thread

5.2.3 mbed TLS Integration

The integration of mbed TLS into the server application was realized in four steps:

1) Inclusion of mbed TLS header files

The first step was to include the needed mbed TLS header files in the main source file of the server application. The included files are listed in the following table:

Table 5. Included header files with description

Header file	**Description**
ssl.h	contains the most important functions for SSL/TLS
entropy.h	implements entropy accumulator
ctr_drbg.h	functions and macros for random number generation
pk.h	functions and macros for public and private keys handling
x509_crt.h	functions and macros for certificate handling

2) mbed TLS initialization

The initialization of mbed TLS is performed in the initializations part of the Server thread. This initialization involves the following steps:

a) creation and initialization of mbed TLS structures
b) certificate and private key parsing (they are stored in an internal flash memory of microcontroller)
c) setting up parsed certificate and key to be used by TLS
d) seeding and connection of entropy source to mbed TLS drbg random number generator
e) setting up random number generator callback for TLS functions
f) configuration of mbed TLS in server mode on the transport layer
g) resetting TLS connection

The connection of hardware TRNG to mbed TLS entropy accumulator is realized by the implementation of glue functions and the indication of TRNG presence by “TRNG” flag in “targets.json” file (mbed OS file which describes each supported platform and its features). Luckily, these steps were done for Nucleo-F767ZI board by mbed OS developers, therefore the hardware TRNG will be used by mbed TLS entropy accumulator by default.

3) Implementation of callbacks for mbed TLS

As mentioned in section 4.1.1, SSL/TLS is built on top of the transport layer of the TCP/IP model. The connection between the TLS and transport layer is realized by two callback functions – the first for data receiving, the second for data sending. As soon as the socket mechanism was used, these callbacks are connected to the client instance of the TCPSocket returned by the accept function mentioned in the previous section. The type of these callbacks is defined in the “ssl.h” file. Figure 17 shows an implementation of those callback functions:

```
static int ssl_recv(void *socket, unsigned char *buf, size_t len)
{
   TCPSocket *client = (TCPSocket *)socket;
   int count = client->recv(buf, len);
   return count;
}

static int ssl_send(void *socket, const unsigned char *buf, size_t len)
{
   TCPSocket *client = (TCPSocket *)socket;
   int count = client->send(buf, len);
   return count;
}
```

Figure 17. Implementation of callbacks connecting mbed TLS and socket

4) Setting up TLS binary input/output and performing TLS handshake

After the client has been connected, the TLS connection to the client’s socket

can be realized by calling the function “mbedtls_ssl_set_bio” which is declared in “ssl.h”. This function takes as parameters the client’s socket and sends and receives the callbacks shown in Figure 17. The last required step to ensure encrypted communication is to perform a handshake by calling “mbedtls_ssl_handshake” function. If the handshake was successful, a secured channel was created between the client and server. Now, there are two functions available for the exchange of data between the client and server, which basically replaced the socket methods for data sending and receiving (send and recv).

These functions are “mbedtls_ssl_read” and “mbedtls_ssl_write”. The HTTP server was implemented on top of these functions which makes it HTTPS. Figure 18 shows the socket utilization with added TLS:

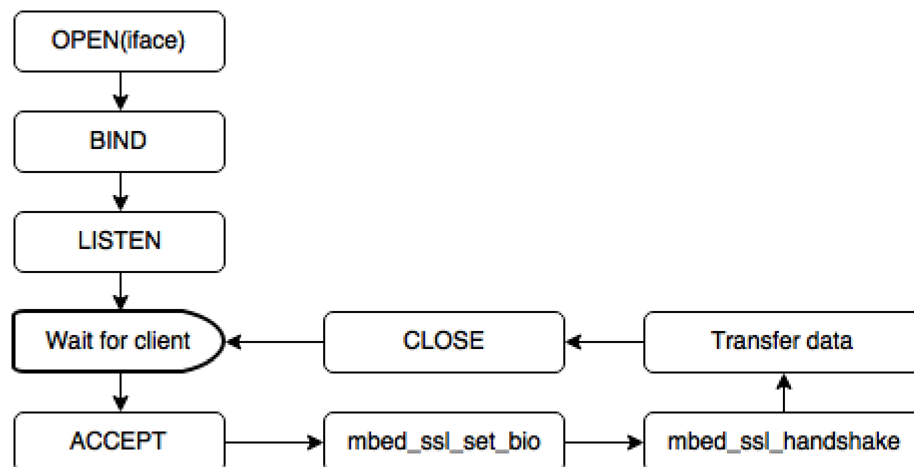

Figure 18. Flowchart of socket mechanism utilization with TLS

5.2.4 HTTP Server

The HTTP server implemented in this thesis consists of two important functions. The first function is called “http_parser” and its role is to parse the incoming request from a client and separate it into several elements. These elements are headers array, HTTP method, URI and content. After the request was parsed to these elements, the second important function is executed with those elements passed as parameters. This second function is called router and its role is to execute specific functions assigned to a particular URI path. These functions then handle the incoming requests and return the response

with HTTP status code. As an example of URI to which a specific function is assigned might be “https://server/time”. When the router receives a URI with a time path, it will execute the “timeRoute” function responsible for time handling (getting and setting). Figure 19 represents the flow of the HTTP server implemented in this thesis (this figure also represents Transfer data block in Figure 18.):

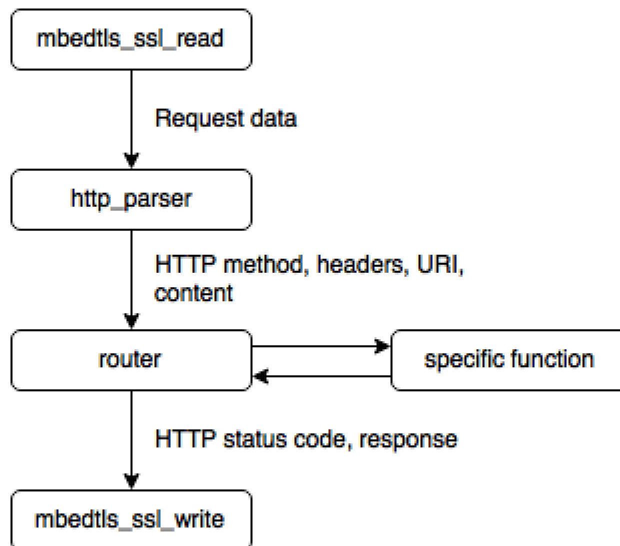

Figure 19. Flow of HTTP server implementation

5.2.5 IP whitelist and user authentication

For an increase of a system’s security, IP whitelist was implemented. IP whitelist is a list of IP addresses, the only ones allowed to communicate with the server. Incoming communication from IP addresses not listed in the IP whitelist is dropped. This measure could somewhat improve the server’s resistance to possible DDoS attacks. The capacity of the IP whitelist is configured at the compile time and is not allowed to be modified in runtime. One “master” IP address also has to be configured at the compile time. The management of other IP addresses is realized at the runtime by REST API shown in Table 4. The IP lookup in the whitelist is performed right after the client has been connected to the server. It is important to place the IP lookup before any TLS functionality, because TLS operations consume the CPU most. Figure 20 shows how the whitelist was inserted into the Server thread loop:

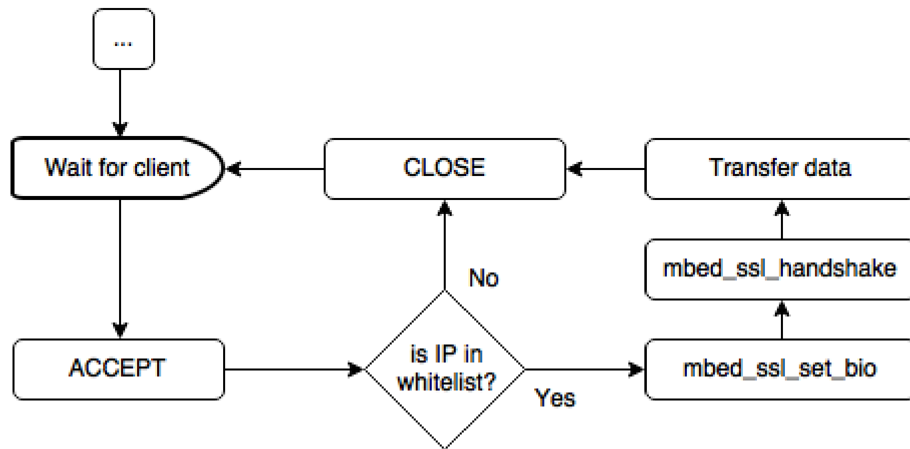

Figure 20. Socket mechanism utilization with IP whitelist

The authentication was solved using HTTP Basic Authentication mechanism (BA). BA is a very simple mechanism which does not require the server to keep session information about a logged user. The principle of this mechanism lies in appending of the BA header to each user's request which should be authenticated.

BA header contains a username and password encoded in base64 format. When the server receives any request, it will loop through the request headers and try to find a BA header. If the server finds the BA header, it will perform the user lookup in a similar way how IP whitelist works. If the request does not contain the BA header, or lookup of the username and password failed, the server will respond with HTTP status code 401 (Unauthorized).

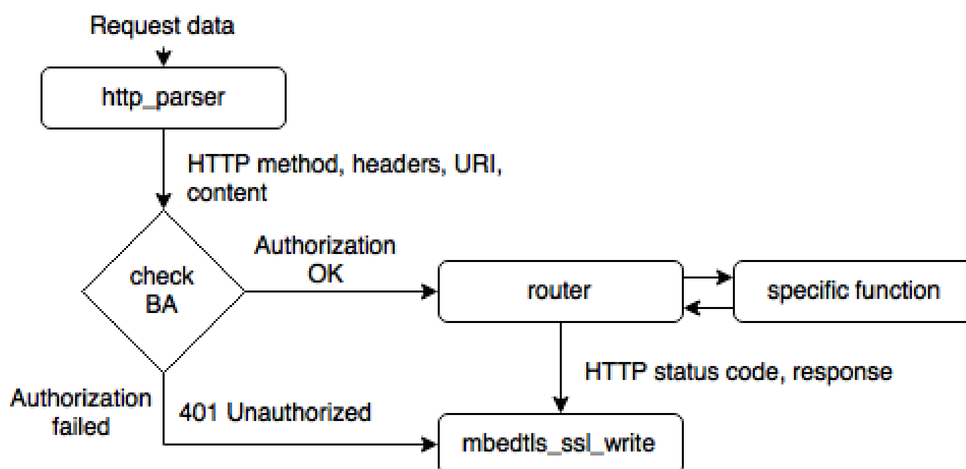

Figure 21. Added BA mechanism into HTTP server

5.2.6 Sensors

For demonstration purposes, two sensors were connected to the Nucleo board. These sensors were HTS221 (temperature and humidity) and LPS25H (atmospheric pressure and temperature). Both of these sensors are a part of the board designed for Raspberry Pi computer - Sense Hat. Even though this board is designed for the Raspberry Pi, is was convenient to use it also in this thesis because it eases the connection of sensors. Both of these sensors support the connection via I2C bus and they are connected on the same bus on the Sense Hat board. The I2C bus is master-slave and it uses addressing via transferred data through it. It gives away the need for addressing/chip-enable signals via another wires. The only necessary wires for the connection of those sensors to Nucleo board were SDA (serial data), SCL (clock), VCC and GND. SDA and SCL signals were connected to VCC through internal pull-up resistors of MCU. mbed OS contains a driver for I2C bus and also on the mbed OS repository, there were open-source drivers available for the mentioned sensors. Thanks to mbed OS, it was very simple to use those sensors without much programming and datasheet reading. Figure 22 shows how sensors are connected to the Nucleo board. Pull-up resistors are not shown because they are inside MCU.

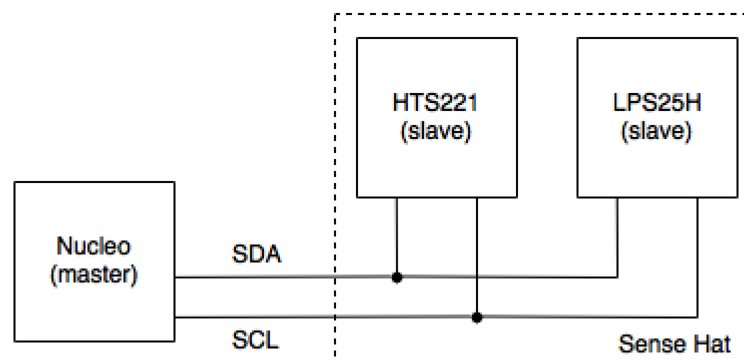

Figure 22. Connection of sensors to Nucleo via I2C bus

In the designed software in this thesis there is a separate thread designated for reading and storing values of these sensors. The values of the sensors are read in a specific time interval and stored to the file on the SD card connected through the SPI interface. mbed OS repository provides a driver for the SD card, and it has also implemented a FAT file system with POSIX compatible

interface. The data read by sensors is exposed through REST API in two ways. The first way serves records which were stored in the file on the SD card by the above-mentioned thread. The data stored in the file are in the following CSV format: “unix_time, temperature, humidity, pressure”. This format makes it convenient to read records also with some programs on a PC. The second way does not use the stored data. This way will perform a reading of the sensor’s values right after the request, thus, this way gives only the actual data. Table 6 shows the implemented API for reading data of sensors.

Table 6. Implemented API for sensors

HTTP Method	**URI**	**Description**
GET	https://server/sensor	read and get data from all sensors
GET	https://server/sensor/temp	read and get temperature
GET	https://server/sensor/humi	read and get humidity
GET	https://server/sensor/press	read and get atmospheric pressure
GET	https://server/file/[filename]	fetch the whole csv file

The following figure shows an example of the returned JSON data of the sensors by API:

```
{
   "temp":25.123150,
   "humi":35.781921,
   "press":1000.176758
}
```

Figure 23. JSON data returned by calling GET on “https://server/sensor” route

5.3 Client application

A simple and intuitive web application was developed to provide a user interface for Nucleo server. This application communicates with Nucleo board over a secured REST API, and modern web technologies were used to develop it. The heart of the whole application is the Angular2 framework. The Angular2 framework uses HTML, CSS and TypeScript language which is a modern scripting language derived from JavaScript developed by Microsoft, and its main advantage is the support for static typing. It allows developers to write cleaner and more maintainable code in comparison with JavaScript. Since web browsers do not support the execution of TypeScript, it had to be translated to JavaScript using a command line tool called Angular CLI that also contains other functionalities such as project and code generation, bundling, automated testing and web application serving.

For the user interface (UI) elements, the PrimeNG library was used. This library was created especially for the Angular2 framework. It comes with a great number of UI elements such as buttons, tables, dialogs, input fields, headers, charts, etc. PrimeNG also contains multiple graphical themes for free. The UI of the client application is basically composed of two parts: connection header and dashboard. The connection header contains three elements. The first element is an input filed for Nucleo server's URI with the save button. The second element is the credentials form for the username and password. The third element is a status indicator. When the users set up their credentials and URI of Nucleo server, the save button can be pressed. After the save button has been pressed, the user's credentials and the server's URI are stored in the browser's memory. Since this moment, every HTTP request made from this application will be directed to the server's URI and will contain the BA header with the encoded user's credentials. Pressing the save button will also perform the check request. If the user's credentials are correct and the server is alive on a particular URI, the status indicator will show the status: "OK". Figure 24 is a screenshot of the connection header:

Figure 24. Screenshot of connection header from client web application

The second part of UI is the dashboard. On top of the dashboard, there is a tab view component placed, through which the content of dashboard can be switched. The content under each tab usually represents the functionality of the Nucleo server exposed by a specific URI path. The following figure is a screenshot from the dashboard of the web application. The tab opened on the screenshot is called Access control and it utilizes the functionality of Nucleo server on “ip” path. There is the possibility to add and remove IP addresses to and from IP whitelist mentioned in section 5.2.5.

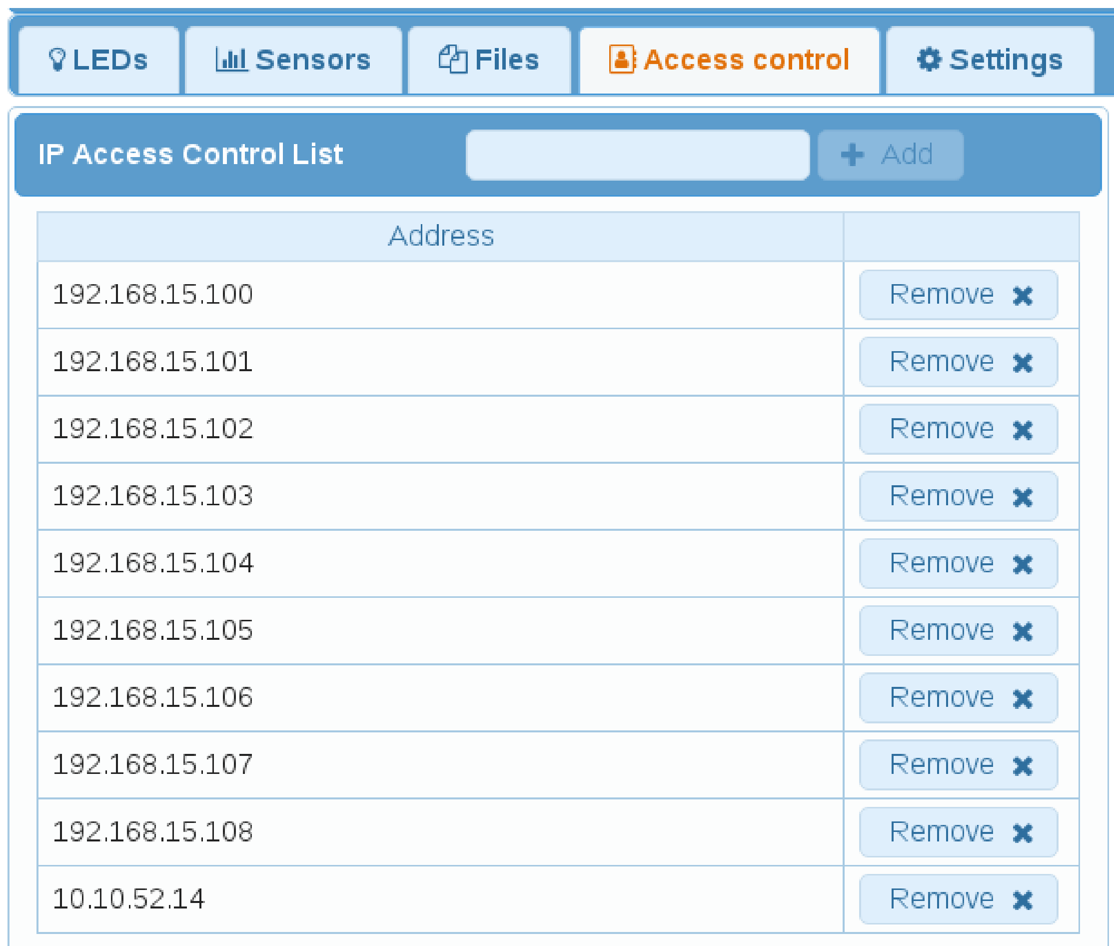

Figure 25. Dashboard with IP whitelist opened

6 Conclusion

6.1 Issues

The development of the software for the Nucleo board faced several issues. In the first version of the server's software, TLS integration did not use the session resumption. TLS handshake involves a use of public-key cryptography which causes a significant load of the CPU. When a 2048-bit private key was used, the TLS handshake with the Nucleo board took more than two seconds. It was not acceptable to perform a full TLS handshake in every request because of a huge latency. This issue was solved by adding session resumption to TLS handshake. This measure gave away the need for performing public-key computation by every request. Now, the full TLS handshake is performed only by the first request. Every other request uses cached keys, and there is no more a noticeable latency. The integration of the TLS session resumption was relatively easy, the only need was to include "ssl_cache.h" from mbed TLS library and insert three lines of code. Another issue was caused by HTTP access control mechanism (CORS) implemented in web browsers. For security reasons, the web browsers do not allow to perform HTTP requests to the server which is different from the server that provided them the main content. If requests to a different server are needed, the server must let the the browser know that it allows those requests. As soon as the content of the user application developed in this thesis was served from a different source, the web browser blocked all requests to the REST API on the Nucleo board. A solution to this issue was also relatively simple. The only necessary feature was to add special headers to the server's responses which tell the web browser that browser can access it. The added headers to the server's responses are listed as follows:

- Access-Control-Allow-Origin: *
- Access-Control-Allow-Methods: GET, POST, PUT, DELETE
- Access-Control-Allow-Headers: Authorization,...

6.2 Discussion

While the encrypted network communication was implemented in an IoT device, it still cannot be considered as fully secure. There are other possible attacks which could compromise the functionality of the IoT device. As an example might be a Distributed Denial-of-Service (DDoS) attack. The principle of this attack lies in repetitive sending of useless requests to the server by multiple hosts thus causing the overload of the server. Then, the server will become unavailable for the real user. This kind of attack was considered while implementing the software of the server. The implemented IP whitelist was an attempt to eliminate this kind of attack but in a case of a real attack, it would not help so much. This IP whitelist is working on top of the transport layer, which is not a very good approach. A slightly better way would be if the IP whitelist was implemented on the lower layer of the TCP/IP model.

Another big question for discussion would be the design of the used system architecture. In the current design, the IoT device acts like a server. This architecture might be useful in cases when there are no other devices performing similar tasks present and the device is not a part of a bigger network of IoT devices. If the device should be a part of some network (e.g. temperature sensors in every room of a building), a more clever approach would be if the single device were a client and it were pushing its data to some central server. With this question also comes another uncertainty about the chosen application protocol. There are several protocols more suitable for IoT than HTTP protocol. Examples of better alternatives would be CoAP, LWM2M, MQTT or XMPP. Since these protocols are less heavy for network transport, CPU and memory, it would not be so easy to connect a web application to control an IoT device through them. For example, CoAP protocol is very similar to HTTP protocol, however, the straight connection from the web browser to the device using CoAP is still not possible. CoAP uses UDP protocol instead of TCP. To achieve a connection between an CoAP device and a web browser, the presence of some proxy server is necessary.

6.3 Summary

The purpose of this thesis was to develop an IoT application with secured data transfer over the network. The first part of the thesis analyzed the available ARM cores on the market which are powerful enough to ensure encryption and security. This part also gave a brief overview of the ARM architecture, which needs to be understood when developing software for it. The next part of the thesis focused on software development for microcontrollers and how multiple software strategies can be utilized on it. There is also a description of mbed OS - the full-featured real-time operating system. This system was used to develop a secure IoT application on top of it. Security on the network was ensured by adding encryption and basic authentication mechanism present in the HTTP protocol. Encryption was realized by Transport Layer Security (TLS) described in Chapter 4. The result of this thesis is a working device connected to the internet through an encrypted channel. This device is controlled by REST API, and a simple web application using that API was also developed.

The most challenging part of this thesis was the theoretical research. I had to read through many documents from the ARM company. ARM architecture is relatively complicated and it was difficult for me to extract the most important information about it. Another difficult task was to find information about and describe common software strategies. However, these obstacles were overcome and now I have much stronger knowledge in the area of microcontrollers. I am particularly glad that I got to know mbed OS, which rapidly simplified the whole process of software design and programming. I am looking forward to using it again.

The source code of software for the Nucleo board and web application which was developed in this thesis can be accessed at the following link:

https://www.github.com/spacive/restiot

References

Heath, S. 2002. *Embedded Systems Design.*
Accessed on 3.4.2017. Retrieved from:
https://books.google.fi/books?id=BjNZXwH7HlkC&pg=PA11&redir_esc=y#v=onepage&q&f=false

Choudhary, H. *Difference between Microprocessor and Microcontroller.*
Accessed on 4.4.2017. Retrieved from:
https://www.engineersgarage.com/tutorials/ difference-between microprocessor-and-microcontroller

Engineersgarage, 2017. Accessed on 4.4.2017. Retrieved from:
https://www.engineersgarage.com/sites/default/files/imagecache/Original/wysiwyg_imageupload/1/8051-Architecture_4.gif

ARM, 2017. Accessed on 6.4.2017. Retrieved from:
http://www.arm.com/products/processors/cortex-a

ARM, 2010. Accessed on 19.4.2017. Retrieved from:
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439b/DDI0439B_cortex_m4_r0p0_trm.pdf

Yiu, J. 2010. *The Definitive Guide to the ARM® Cortex-M3.* Accessed on 12.4.2017. Retrieved from:
https://www.eecs.umich.edu/courses/eecs373/labs/refs/M3%20Guide.pdf

STMicroelectronis, 2017. Accessed on 12.4.2017. Retrieved from:
http://www.st.com/en/microcontrollers/stm32f767zi.html

IC Insights, 2013. *MCU Market on Migration Path to 32-bit and ARM-based Devices.* Accessed on 12.4.2017. Retrieved from:
http://www.icinsights.com/news/bulletins/MCU-Market-On-Migration-Path-To-32bit-And-ARMbased-Devices/

Ayav, T. 2017. Accessed on 18.4.2017. Retrieved from:
http://web.iyte.edu.tr/~tolgaayav/courses/ceng523/lecture2-RTOS1.pdf

Technopedia, 2017. Accessed on 19.4.2017. Retrieved from:
https://www.techopedia.com/definition/3344/cooperative-multitasking

Daintith, J.; Wright E. 2008. *A Dictionary of Computing.* Accessed on 20.4.2017. Retrieved from:
http://www.oxfordreference.com/view/10.1093/acref/9780199234004.001.0001/acref-9780199234004-e-4360?rskey=YZo8xo&result=1

Keil, 2017. *RTX Real-Time Operating System.* Accessed on 20.4.2017. Retrieved from:
http://www.keil.com/arm/rl-arm/kernel.asp

mbed, 2017. Accessed on 21.4.2017. Retrieved from:
https://docs.mbed.com/docs/mbed-os-api-reference/en/latest/APIs/tasks/rtos/#mutex

Sans, 2011. *Understanding Encryption.* Accessed on 25.4.2017. Retrieved from:
https://securingthehuman.sans.org/newsletters/ouch/issues/OUCH-201107_en.pdf

Dierks, T.; Rescorla, E. 2008. *RFC 5246.* Accessed on 25.4.2017. Retrieved from:
https://tools.ietf.org/html/rfc5246#section-1

mbed TLS, 2017. Accessed on 25.4.2017. Retrieved from:
https://tls.mbed.org/module-level-design-cipher

Luedtke, 2012. Accessed on 25.4.2017. Retrieved from:
http://image.slidesharecdn.com/20120418luedtkessltlscbcbeast-150301075556-conversion-gate01/95/beast-attack-on-ssltls-explained-6-638.jpg?cb=1425196642

Zoompf, 2017. Accessed on 26.4.2017. Retrieved from:
https://zoompf.com/wp-content/uploads/2014/10/handshake.png

Lin, Z. 2015. *TLS Session Resumption: Full-speed and Secure.* Accessed on 26.4.2017. Retrieved from:
https://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure/

Rescorla, E. 2000. *RFC 2818.* Accessed on 27.4.2017. Retrieved from:
https://tools.ietf.org/html/rfc2818

Hoffman, Ch. 2014. *How Computers Generate Random Numbers.* Accessed on 27.4.2017. Retrieved from:
https://www.howtogeek.com/183051/htg-explains-how-computers-generate-random-numbers/

Wikipedia, 2017. Accessed on 5.4.2017, 6.4.2017, 11.4.2017, 25.4.2017, 26.4.2017. Retrieved from:
https://en.wikipedia.org/wiki/ARM_Holdings
https://en.wikipedia.org/wiki/ARM_architecture
https://en.wikipedia.org/wiki/Interrupt
https://en.wikipedia.org/wiki/Memory_protection
https://en.wikipedia.org/wiki/Transport_Layer_Security

STMicroelectronics, 2016. *AN4230.* Accessed on 10.4.2017. Retrieved from:
http://www.st.com/content/ccc/resource/technical/document/application_note/4a/6a/82/05/8e/9e/4e/94/DM00073853.pdf/files/DM00073853.pdf/jcr:content/translations/en.DM00073853.pdf

STMicroelectronics, 2016. *STM32F765xx STM32F767xx STM32F768Ax STM32F769xx.* Accessed on 10.4.2017. Retrieved from:
http://www.st.com/content/ccc/resource/technical/document/datasheet/group3/c5/37/9c/1d/a6/09/4e/1a/DM00273119/files/DM00273119.pdf/jcr:content/translations/en.DM00273119.pdf

Linux kernel, 2005. Accessed on 20.4.2017. Retrieved from:
http://elixir.free-electrons.com/linux/latest/source/drivers/char/random.c

McVetta, J. 2012. *What is a RESTful API?*. Accessed on 3.5.2017. Retrieved from:
http://advanced-python.readthedocs.io/en/latest/rest/what-is-rest.html

RestApiTutorial, 2017. Accessed on 5.5.2017. Retrieved from:
http://www.restapitutorial.com/lessons/httpmethods.html

JSON, 2017. Accessed on 4.5.2017. Retrieved from:
http://www.json.org/