BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

BACHELOR'S THESIS

Brno, 2020 Jaromír Bača

BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

END-TO-END ENCRYPTION PROTOCOL FOR IEEE 802.15.4

PROTOKOL S KONCOVÝM ŠIFROVÁNÍM PRO IEEE 802.15.4

BACHELOR'S THESIS

BAKALÁŘSKÁ PRÁCE

AUTOR Jaromír Bača

AUTOR PRÁCE

SUPERVISOR VEDOUCÍ PRÁCE	Ing. Ondřej Krajsa, Ph.D.

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Informační bezpečnost**

Ústav telekomunikací

Student: Jaromír Bača ***ID:*** 133372
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Protokol s koncovým šifrováním pro IEEE 802.15.4

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte a implementujte protokol pro bezpečnou výměnu klíčů pro šifrovací algoritmus AES128. Tento protokol bude pro komunikaci využívat linkový protokol standardu IEEE802.15.4 a síťový protokol Atmel LighWeight Mesh. Implementaci navrženého protokolu a také linkového síťového protokolu proveďte na mikrokontroléru AVR ATMega128RFA1. Změřte časy nutné pro výměnu klíče a časy nutné pro šifrování a dešifrování informace. V rámci práce proveďte bezpečnostní analýzu navrženého protokolu a porovnání s obdobnými technikami. Dále proveďte analýzu a návrh výměny klíče mezi bezdrátovým uzlem a prvkem v síti internet využívající navržený protokol.

DOPORUČENÁ LITERATURA:

[1]LAVANYA, M. a V. NATARAJAN. Lightweight key agreement protocol for IoT based on IKEv2. Computers and Electrical Engineering [online]. Elsevier, 2017, 64 [cit. 2019-09-16]. DOI: 10.1016/j.compeleceng.2017.06.032. ISSN 0045-7906.

[2] THAMES, Lane a Dirk SCHAEFER. Cybersecurity for Industry 4. 0: Analysis for Design and Manufacturing. Cham: Springer, 2017. DOI: 10.1007/978-3-319-50660-9. ISBN 9783319506593.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 25.8.2020

Vedoucí práce: Ing. Ondřej Krajsa, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRACT

This thesis explores the topic of encryption of communication between low-voltage devices that are controlled by microcontrollers. Two deRFnod development boards were used in the work, which were equipped with AVR ATmega 128 RFA1 chips, which enable wireless communication. The application was developed and tested on these devices. The final output of the work is the design of an application for asymmetric key exchange, which is based on elliptic curves. This application is implanted in Atmel LightWeight, where the issue of mutual communication between communicating points is also addressed. The generated key is also used to propagate communication using the AES encryption algorithm, which is already implemented in the used LightWeigt protocol. This encryption allows not only encryption of endpoints, but also of the communication tunnel. Such protection provides users with anonymity of data and makes it impossible or very difficult for potential attackers to physically locate devices based on knowledge of data routing on the network.

KEYWORDS

Microcontrollers, Internet of Things, lightweight mesh, asymetry cryptpgraphy, elliptic curves, AES

ABSTRAKT

Tato práce se zabývá problematikou šifrování komunikace mezi nízkonapěťovými zařízeními, které jsou ovládány pomocí mikrokontrolerů. V rámci práce byly používány dvě vývojové desky deRFnod, které byly osazeny čipy AVR ATmega 128 RFA1, které umožňují bezdrátovou komunikaci. Na těchto zařízeních probíhal vývoj a testování aplikace. Finálním výstupem práce je návrh aplikace pro asymetrickou výměnu klíčů, která je založena na eliptických křivkách. Tato aplikace je implantována v Atmel LightWeight, kde je i řešena otázka vzájemné komunikace mezi komunikujícími body. Vygenerovaný klíč je dále použit pro širfování komunikace pomocí šifrovacího algoritmu AES, který je již implementován ve využitém LightWeigt protokolu. Toto šifrování umožňuje nejenom šifrování koncových bodů, ale i komunikačního tunelu. Taková ochrana poskytuje uživatelům anonymitu dat a znemožňuje nebo velmi znesnadňuje potenciálním útočníkům zařízení fyzicky lokalizovat na základě znalosti směrování dat v síti.

KLÍČOVÁ SLOVA

Mikrokontroléry, internet věcí, lightweight síť, asymetrická kryptografie, eliptické křivky, AES

BAČA, Jaromír. *Protokol s koncovým šifrováním pro IEEE 802.15.4*. Brno, Rok, 44 p. Semestral Project. Brno University of Technology, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Advised by Ing. Ondřej Krajsa, Ph.D.

Typeset by the `thesis` package, version 3.05; `http://latex.feec.vutbr.cz`

ROZŠÍŘENÝ ABSTRAKT

Tato práce se zabývá problematikou šifrování komunikace mezi nízkonapěťovými zařízeními, které jsou ovládány pomocí mikrokontrolerů. V rámci práce byly používány dvě vývojové desky deRFnod, které byly osazeny procesory AVR ATmega 128 RFA1, které umožňují bezdrátovou komunikaci. Na těchto zařízeních probíhal vývoj a testování aplikace. Finálním výstupem práce je návrh aplikace pro asymetrickou výměnu klíčů, která je založena na eliptických křivkách. Tato aplikace je implantována v Atmel LightWeight, kde je i řešena otázka vzájemné komunikace mezi komunikujícími body. Vygenerovaný klíč je dále použit pro širfování komunikace pomocí šifrovacího algoritmu AES, který je již implementován ve využitém LightWeigt protokolu. Toto šifrování umožňuje nejenom šifrování koncových bodů, ale i komunikačního tunelu. Taková ochrana poskytuje uživatelům anonymitu dat a znemožňuje nebo velmi znesnadňuje potenciálním útočníkům zařízení fyzicky lokalizovat na základě znalosti směrování dat v síti.

Navržený algoritmus na výměnu klíčů, je založen na obecné teorii eliptických křivek, která je popsána v kapitole 4.1. Návrh algoritmu je realizován v jazyce C a v průběhu vývoje byl implementován a testován na vývojový deskách od německého výrobce Dresden Elektronik, GmbH. Vývoj aplikace probíhal v prostředí editoru Code::Block a následně byl přenášen do prostředí editoru Atmel Studiu, který umožňuje testování běhu algoritmu přímo na vývojové desce. Implementace vlastního algoritmu do síťového protokolu LightWeight není zcela úspěšná. Nepodařilo se sestavit úspěšnou komunikaci mezi dvěma komunikačními body. Ačkoliv jednotlivé komponenty algoritmu na výměnu klíčů byly v průběhu návrhu vývoje úspěšně testovány. Na důkaz tohoto tvrzení byly jednotlivé komponenty algoritmu vytvořeny jako spustitelné soubory ve fromátu .exe a jsou součástí přílohy této práce.

DECLARATION

I declare that I have written the semestral project titled "Protokol s koncovým šifrováním pro IEEE 802.15.4" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the project and listed in the comprehensive bibliography at the end of the project.

As the author I furthermore declare that, with respect to the creation of this semestral project, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno . . . .

author's signature

ACKNOWLEDGEMENT

Firstly, I would like to thank my thesis supervisor, Ing. Ondřej Krajsa, PhD., for his professional guidance. Without his tips and deep patience, this thesis would not never be completed.

Contents

Introduction

The future belongs to automation. We have all come across concepts such as smart cities, self-driving cars, or fully-automated factories that do not need human beings to operate them. These things, which sounded like science fiction 20 years ago, are becoming a reality today. In addition, due to falling hardware prices, automation is not just a privilege for large corporations but is becoming increasingly commonplace.

One of the pillars of automation is the collection and flow of data, which introduces a new concept—the Internet of Things. Under this term, we can imagine a large number of small devices that are often not even connected to the mains and are powered by batteries or solar panels. They use wireless networks to communicate with the environment, and they use wireless networks that are specially designed for these small devices due to their limited power options.

Each new technology brings advantages and disadvantages. The main problem of wireless networks is their vulnerability to attacks. Input or output data can be both tapped and altered, which can result in a number of situations, from unpleasant to fatal. The logical response is to use some kind of network security, but given the fact that most small devices are dedicated to and built on microcontrollers, it is necessary to use an adequate lightweight solution.

Structure of the thesis

This bachelor's thesis is divided into theoretical and practical parts and concludes with a summary of what results were achieved. Microcontrollers are generally described in the theoretical part. The next chapter is devoted to deRFnode development boards, which were used during the work and in the Atmel Studio environment, which was used to implement the design of the key exchange algorithm. The next chapter of the theoretical section is a brief introduction to network theory for low-power devices and the Lightweight Mesh protocol, which serves as a basis for the implementation of the algorithm. The theoretical part concludes with chapters that discuss the possibility of key exchange according to the IKEv2 standard and the theory of elliptic curves, which also describes the calculation methodology.

1 Hardware

This thesis is practically oriented and uses several special software and hardware instruments. In this chapter, the microcontrollers, which form the core of the used deRFnode 1TNP2 DBT boards, are described in detail. The conclusion of the chapter deals with the Atmel Studio 7 development environment.

1.1 Mirocontrollers

In today's world, we are surrounded by various small smart devices that can record, for example, ambient temperature, or simple machines that perform one or a limited number of defined activities. All of these devices work thanks to the small, built-in mini-computers we call microcontroller units (MCUs).

Although MCUs look like the processors known from PC assemblies, they are fully functional computers. In the case of computational operations, no other peripheral input and output devices can be used. It is sufficient to provide them with electricity. In addition to the CPU itself, they include RAM and EEPROM to store the code that the computer executes. In this thesis, the ATmega128RFR2 chip is used, which is directly embedded in the deRFnode 1TNP2 DBT development board.

They are equipped with serial ports for basic communication with peripherals, which may be other MCUs or electronic devices such as sensors or servomotors. Selected ports are grouped into interfaces, such as a Serial Peripheral Interface (SPI), which allows full-duplex data communication between two microcontrollers. On one side, there is a master that controls the slave microcontroller on the other. This interface can be used for computer-to-MCU communication, but it requires a special converter. Another well-known interface is the I^2C, also known as a Two-Wire Interface (TWI), which allows two devices to be connected in a series with two wires at a time and communicate using address data.

Among the world's leading manufacturers of MCUs are companies such as Texas Instruments, Microchip Company (formerly Atmel), Intel Corp., and Fujitsu.

1.2 deRFnode 1TNP2 DBT

This is a development board that includes a radio module and an ATmega128 chip. This board is designed for low-energy data networks such as 802.15.4. The board has a number of interfaces such as USB, JTAG[1], or TWI. In addition to being powered via a USB cable connected to a computer, the board can be powered with a 5V DC plug. The board is also equipped with a battery pack for three AAA batteries, which allow the board to operate without the need to connect to the power grid.

The board is produced in two variants: deRFgateway and deRFnode. The first is equipped with an Ethernet interface. This board can be used as a network coordinator that collects data from other nodes and sends it to a different network, in this case, to an 802.3 Ethernet network. The second type, deRFnode, can serve as a coordinator within a WSN[2] network or as a reduced-function device node. The board contains sensors that measure acceleration, temperature, and luminosity.

The main advantage of these boards is their variability. The radio module is removable from the board and can be replaced with another compatible type from Dresden Elektronik Verkehrstechni, GmbH. Another optional part is the software used; we can choose between different network stacks from the manufacturer or external developers.

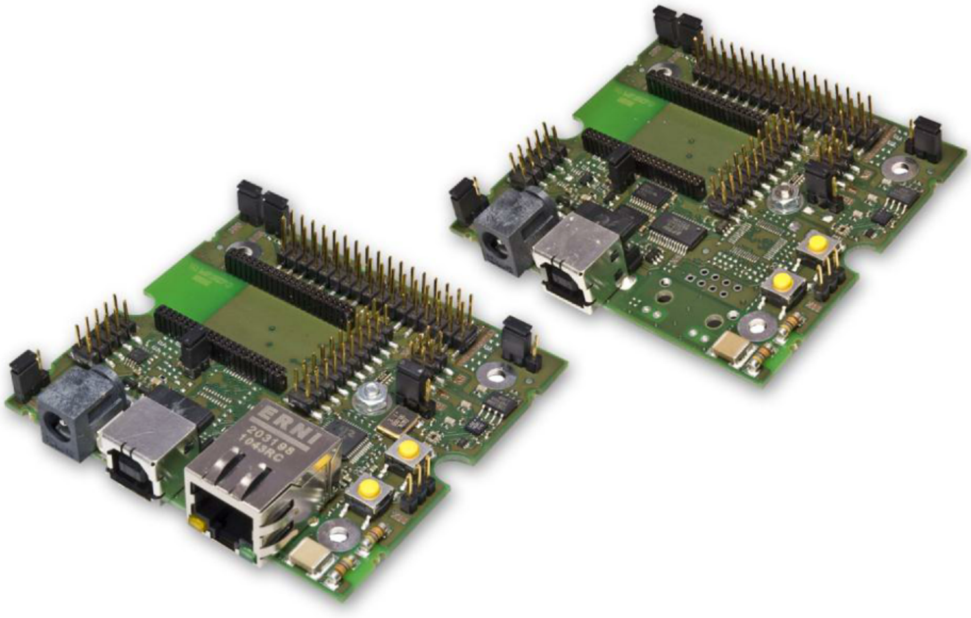


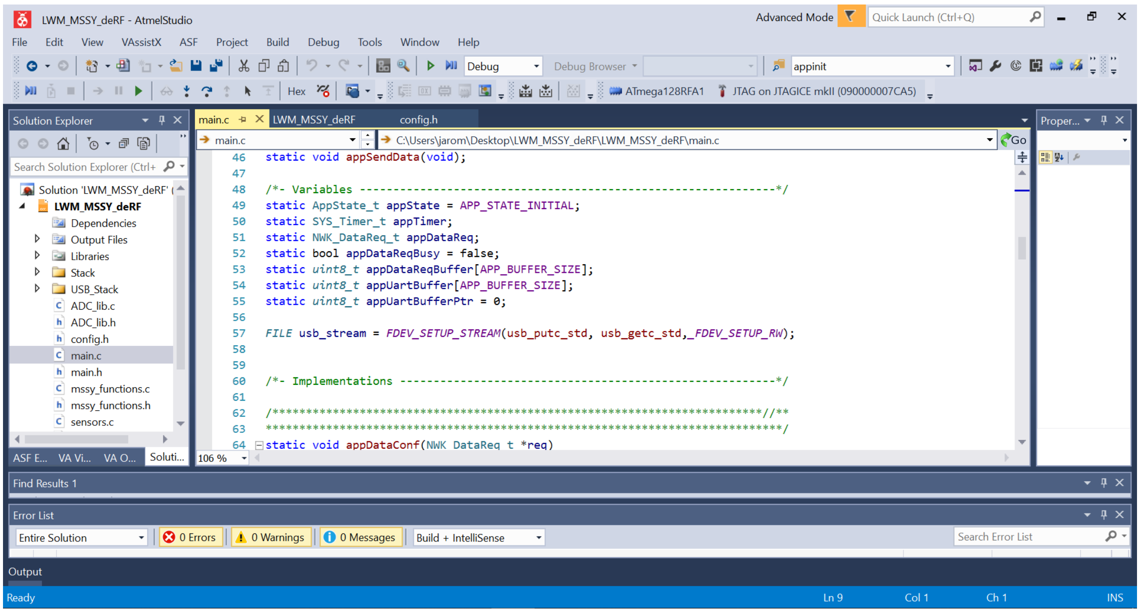
Fig. 1.1: deRFgateway and deRFnode board (without radio modules) [2]

[1] Joint Test Action Group - industry standardized connector

[2] Wireless Sensor Network

1.3 Atmel Studio IDE

The software part of this term paper was realized in the Atmel Studio 7 integrated development environment (IDE). This development environment is intended for the development and debugging of applications written in C and C++ languages. The Studio allows the programming of over 500 supported AVR and SAM microcontrollers via USB programmers, for example, Atmel ICE.



```
static void appSendData(void);

/*- Variables --------------------------------------------------------------*/
static AppState_t appState = APP_STATE_INITIAL;
static SYS_Timer_t appTimer;
static NWK_DataReq_t appDataReq;
static bool appDataReqBusy = false;
static uint8_t appDataReqBuffer[APP_BUFFER_SIZE];
static uint8_t appUartBuffer[APP_BUFFER_SIZE];
static uint8_t appUartBufferPtr = 0;

FILE usb_stream = FDEV_SETUP_STREAM(usb_putc_std, usb_getc_std,_FDEV_SETUP_RW);


/*- Implementations --------------------------------------------------------*/

/*************************************************************************//**
*****************************************************************************/
static void appDataConf(NWK_DataReq_t *req)
```

Fig. 1.2: Atmel Studio 7 IDE

2 Standard 802.15.4

The main motivation for the design of IEEE 802.15.4 was to create a communication standard for WPAN networks that would be optimized for low-energy devices for use in industrial automation. This standard serves as a basis for higher protocols, such as ZigBee, WirelessHard, and 6LoWPAN. The OSI model defines the link and physical layer parameters. Higher layer protocols are not specified.

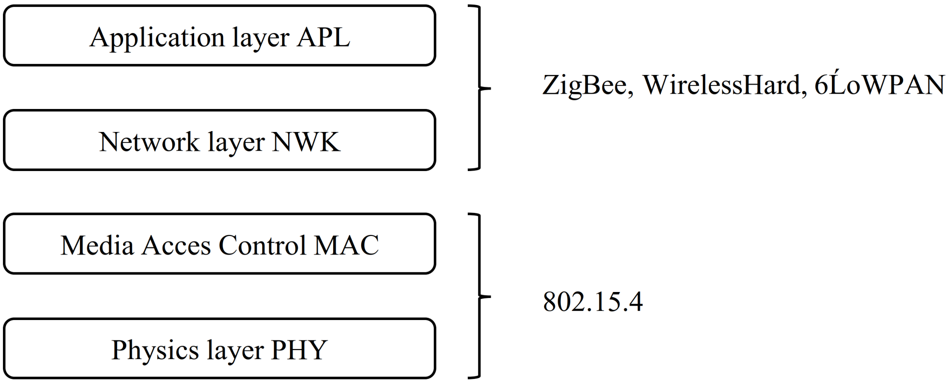


Fig. 2.1: Lyers of 802.15.4 and higher protocols

The physical layer

The general task of the physical layer is to transmit data. This layer defines the frequency band and modulation used. Within the physical layer, we distinguish a total of three frequency bands with different transmission speeds and a number of channels.

- Europe 868.0 – 868.8 MHz – pouze jeden kanál (0), 20 – 250kbit/s
- North America 902 – 928 MHz - 13 channels (1-14)
- Worldwide 2400–2483.5 MHz - 16 channels

Data link layer

The link layer ensures the correct addressing of forwarded data. Other tasks include, for example, synchronization according to the beacon frame, which it transmits at regular intervals and thus informs the user about the network presence.

2.1 Topology

The standard uses star and mesh topologies. Topologies consist of two basic types of devices: full-function device (FFD) and reduced-function device (RFD).

FFD (Fully Function device)

This device can serve either as a network coordinator, a terminal coordinator, or a terminal only. In the case of the first role, the device acts as a router and, in addition to network management, can forward data to other networks based on other standards, such as Ethernet or WiFi.

RFD (Reduced Function device)

An RFD device is a feature with reduced functionality and only works as an endpoint that receives or sends data to its coordinator, not to another point on the network.

2.1.1 Star

This type of topology consists of one network coordinator to which FFD or RFD devices can be connected but only communicate with the network coordinator.

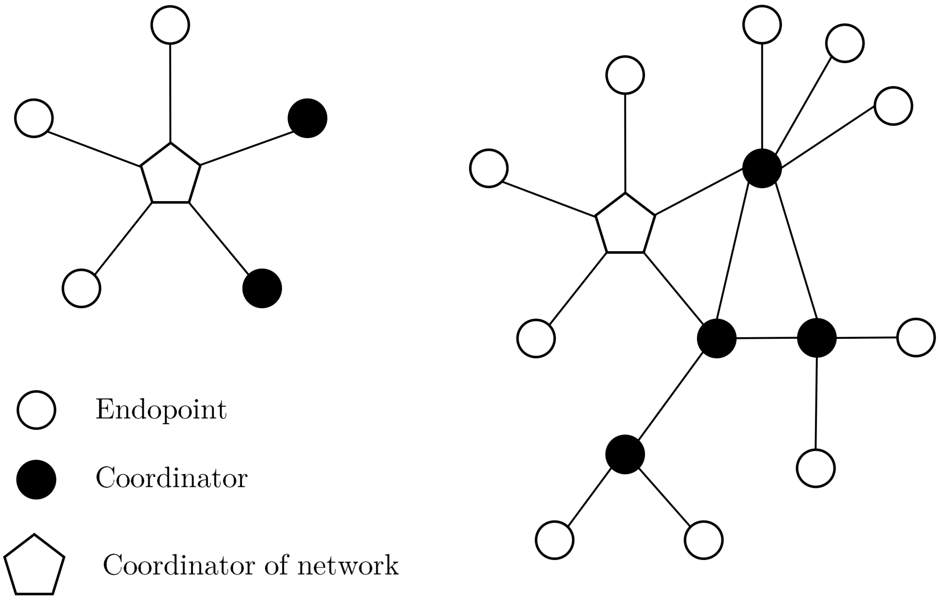


Fig. 2.2: Topologies Star and Mesh (Peer-to-peer) [1]

2.1.2 Mesh (Peer-to-peer)

If there is no requirement for the peer-to-peer topology to send data to other networks, there is no need to include a network coordinator. The advantage of this solution is the possibility of building a network with a higher range than the network coordinator radio module in the star topology. This is made possible by the ad hoc capability where the data packet is forwarded from the sender to the recipient through several intermediate nodes. However, this solution has a negative effect on the energy consumption of the system.

2.2 Atmel lightweight Mesh

For our protocol design, Atmel's Lightweight Mesh SDK was used in a low-power wireless network. It can be applied to any system or development board that works with an MCU with the hLow Power transceiver for 802.15.4, such as the ATmega128RFA1 used in the deRFnode 1TNP2 DBT board. It is possible that, based on this protocol, it could theoretically have up to 65,635 nodes [1].

3 IKEv2 - Internet Key Exchange

The purpose of this protocol is to secure communication between the two parties, not only by securing the forwarded data but also by securing the communication channel. Communication is thus secured against data theft or usage of fraudulent data.

The principles of the IKEv2 are depicted in Figure 3.1. The protocol is divided into three main parts. In the first part, there is a mutual exchange of keys based on the Diffie–Hellman algorithm. Side A proposes various combinations of security associations (SAs), which are a set of algorithms used to encrypt and authenticate the subscribers, to side B. Side B chooses the most appropriate combination of SAs based on its capabilities. In the second phase of the protocol, authentication of the parties, key exchange, and activation of the agreed encryption algorithm according to the selected SA take place. In the third phase, both parties receive an identification tag (SPI), which confirms the identity of the forwarded data. The key exchange is then used again to encrypt the transmitted data.

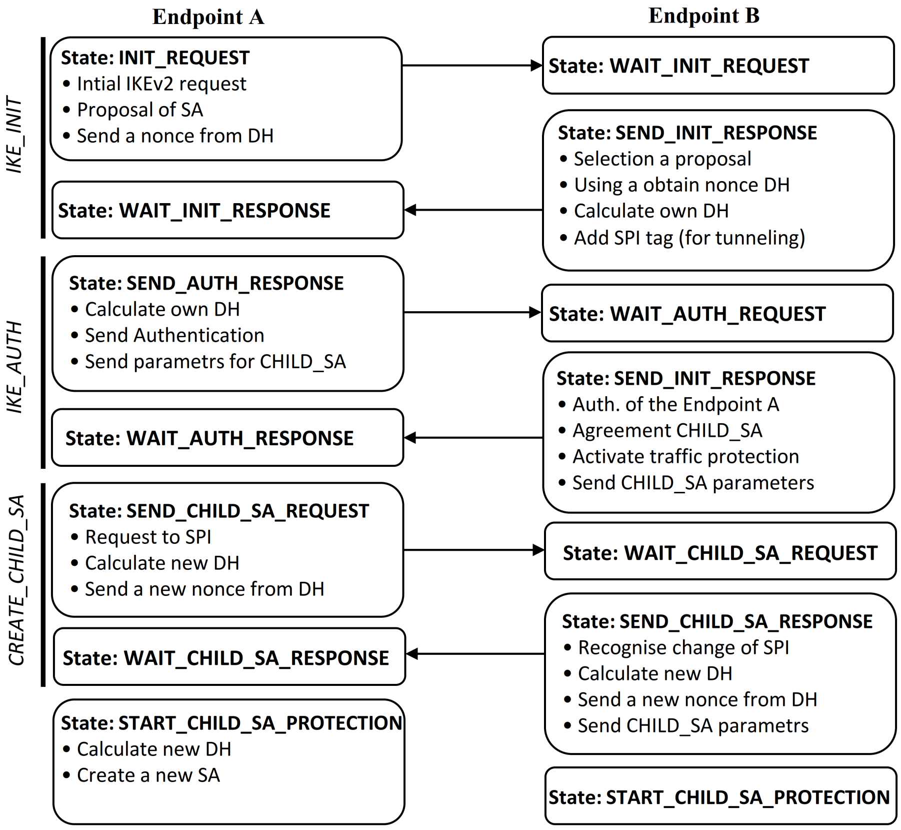
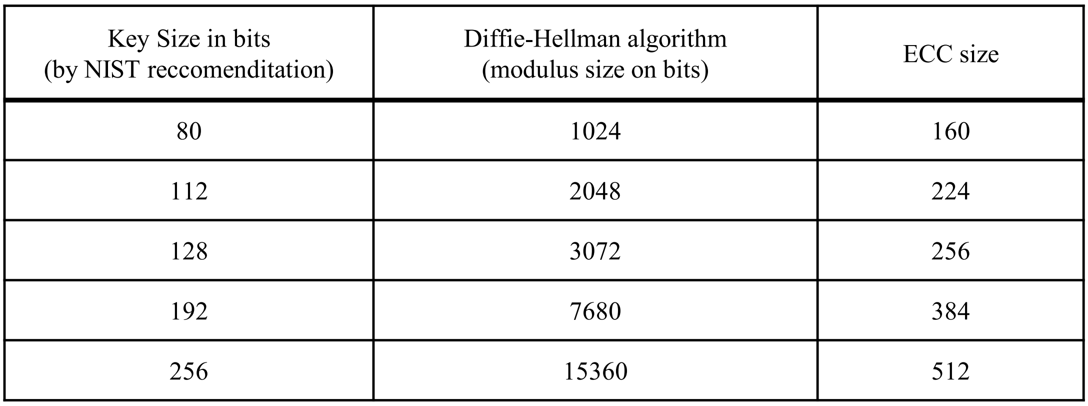

Endpoint A
Endpoint B
IKE_INIT
IKE_AUTH
CREATE_CHILD_SA
State: INIT_REQUEST
• Intial IKEv2 request
• Proposal of SA
• Send a nonce from DH
State: WAIT_INIT_REQUEST
State: SEND_INIT_RESPONSE
• Selection a proposal
• Using a obtain nonce DH
• Calculate own DH
• Add SPI tag (for tunneling)
State: WAIT_INIT_RESPONSE
State: SEND_AUTH_RESPONSE
• Calculate own DH
• Send Authentication
• Send parametrs for CHILD_SA
State: WAIT_AUTH_REQUEST
State: SEND_INIT_RESPONSE
• Auth. of the Endpoint A
• Agreement CHILD_SA
• Activate traffic protection
• Send CHILD_SA parameters
State: WAIT_AUTH_RESPONSE
State: SEND_CHILD_SA_REQUEST
• Request to SPI
• Calculate new DH
• Send a new nonce from DH
State: WAIT_CHILD_SA_REQUEST
State: SEND_CHILD_SA_RESPONSE
• Recognise change of SPI
• Calculate new DH
• Send a new nonce from DH
• Send CHILD_SA parametrs
State: WAIT_CHILD_SA_RESPONSE
State: START_CHILD_SA_PROTECTION
• Calculate new DH
• Create a new SA
State: START_CHILD_SA_PROTECTION

Fig. 3.1: IKEv2 key exchange [5]

4 Elliptic-curve Diffie–Hellman

The previous chapter described the methodology of key exchange. The key exchange itself is realized by asymmetric cryptography, where both parties independently determine the secret key from which they calculate the public key, using a one-way mathematical function. With a one-way function, it is easy to calculate the public key from the secret key, but to recalculate the secret key from the public key is mathematically very difficult. This method was discovered by Whitfield Diffie and Martin Hellman in the 1970s. Although it was later revealed that the method had been invented a few years earlier by the British intelligence and security organization GCHQ, this fact remained a secret until the 1990s, which is why this key exchange protocol is known as the Diffie–Hellman protocol.

Tab. 4.1: Key size comparison between Diffe-Hellman algorithm and ECHD [5]



Key Size in bits (by NIST reccomenditation)	Diffie-Hellman algorithm (modulus size on bits)	ECC size
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

This protocol can be used with different key lengths, as described in 4.1. Among other things, a key size equivalent is added for a better overview, as recommended by the National Institute of Standards and Technology (NIST). However, the key sizes in the classic Diffie–Hellman protocol are unsuitable for application on low-power devices because of the transmission of larger amounts of data, which negatively affects power consumption and the need for more storage space. An elliptic curve-based algorithm can be used to solve this problem. This algorithm's main advantage is a smaller key size when compared to the classic Diffie–Hellman algorithm, but it maintains the same level of security. For example, a 224-bit key created by an elliptical curve-based algorithm provides security equivalent to a 2048-bit key generated by the Diffie–Hellman algorithm.

4.1 Theory of elliptic curve

This part of the thesis briefly describes the theory of elliptic curves. Although there are other theories and computational methods, we confine ourselves to describing an elliptic curve on the GF(p) type of division ring that uses modular arithmetic. The theory described here is based on publications [8]. This theory was used as a basis for creating a key exchange algorithm in this term paper.

An elliptic curve is a plane algebraic curve defined by equation 4.1, where the values x a y represent the Cartesian coordinates of the chosen starting point, and a and b are the curve parameters.

$$y^2 = x^3 + ax + b \tag{4.1}$$

However, before calculating the other points on the curve, it is necessary to verify that the proposed point lies on the curve. This can be easily verified using modified equation 4.2, using modular arithmetic. If the right and left calculations are the same, then it is confirmed that the point lies on the curve. The value p is a prime. This verification was carried out within the framework of this term paper and implemented by software; it is included in the appendices.

$$(y^2) \bmod p = (x^3 + ax + b) \bmod p \tag{4.2}$$

The points on the elliptical curve are made up of an additive group; each additional point arises by adding the previous point and the starting point. It means, logically, that there are two different methods to sum the two points accordingly, whether the added points are the same or different.

Addition of two identical points

This method, also known as doubling, is usually used to calculate the second point in a sequence within a group, where the starting point is added to itself. The graphical solution of this method is shown in Figure 4.1.

$$S = \frac{3{x_P}^2 + a}{y_P} \bmod p \tag{4.3}$$

$$x_R = s^2 - 2x_P \bmod p \tag{4.4}$$

$$y_R = s(x_P - x_R) - y_P \bmod p \tag{4.5}$$

Using these formulas, we can calculate a common point. By calculating the first equation, we obtain the slope of the curve S. We then use this value to calculate the coordinates x_R and y_R.

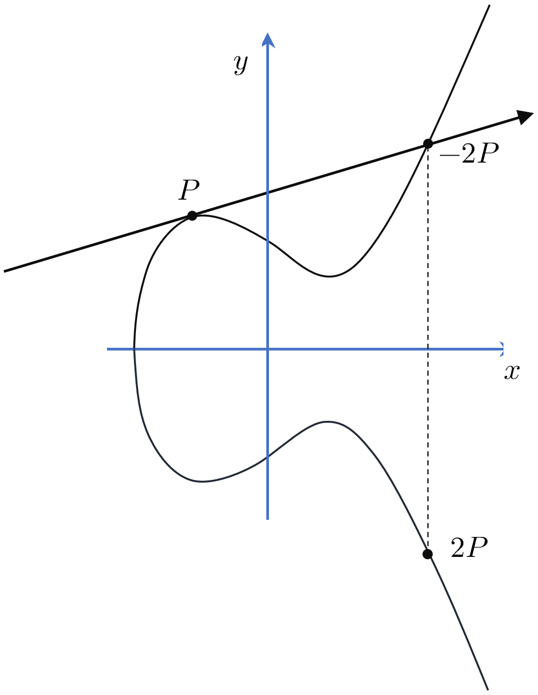


Fig. 4.1: Doubling [8]

Addition of two different points

As in the previous case, we must first calculate the slope of the curve S and then coordinates x_R and y_R. The figure below shows the graphical method of finding a common point.

$$S = \frac{y_P - y_Q}{x_P - x_Q} \tag{4.6}$$

$$x_R = s^2 - (x_P + x_Q) \tag{4.7}$$

$$y_R = s\,(x_P - x_R) - y_P \tag{4.8}$$

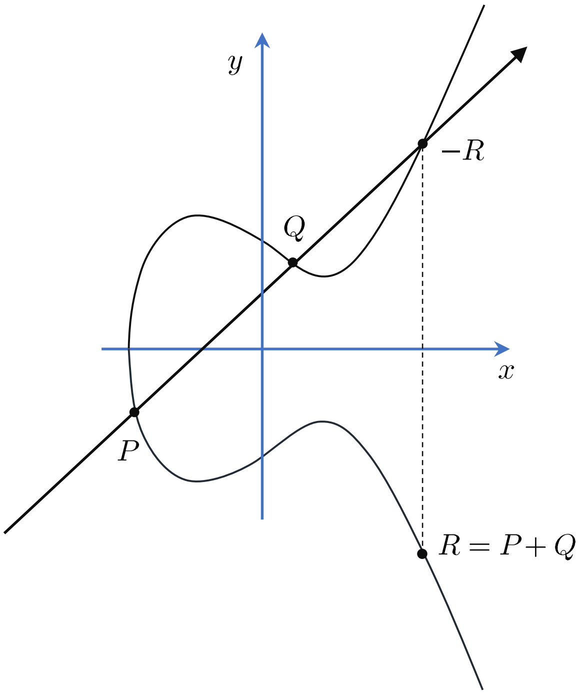


Fig. 4.2: Addition point [8]

5 Key exchange algorithm

In this chapter, we gradually describe the operation of the key exchange algorithm. The algorithm is based on elliptic curves and the Weierstrass calculation method, which was described in Chapter 4. The application is written in C language, and its development took place simultaneously on the platform Code::Blocks and Atmel Studio IDE, where the application was also tested on development boards equipped with AVR ATmega128RFA1 MCUs. The individual parts of the algorithm are described here by generalized flowcharts, which present the function of the described application, not the exact form. More detailed development diagrams are included in the appendix. They are inserted into one sheet in pdf format and can be searched using the entered term.

The key exchange is divided into three phases, which are illustrated in the figure 5.1. The task of the first phase is a random selection of input values; from these values, this phase is further calculated by the group generator and its order. These values are used for a successful key exchange between communicating points. The second phase is responsible for the selection of the key, which is again selected in a pseudo-random way; based on this key, the phase calculates a point on the elliptic curve, which is used for sharing. The third phase of the key exchange can calculate the common key from the received data, resp. a common point that serves as a key in the future or an input value to the AES encryption block within the Lightweight stack.

5.1 Phase A

This phase only works on the initiator side of the communication. Its task is to select suitable values, which then go through several stages of testing to verify their suitability. Furthermore, the algorithm is designed as a system of several loops that run until all tests run properly and until the generated values are clearly usable for a successful asymmetric key exchange.

The first part of this phase of the algorithm is the selection of a number that represents a modulo value. According to the theory, it is given that this number must be a prime number. The selection of a large prime number is realized through the bdRandomSeeded function, which generates a pseudo-random number for us, which is then tested using the bdRabinMiller function. These commands are taken from a library that allows you to work with large numbers. Thanks to it, the selection and control of random values is very effective. Similar commands for handling large numbers occur throughout the our algorithm. If it finds that the number is not

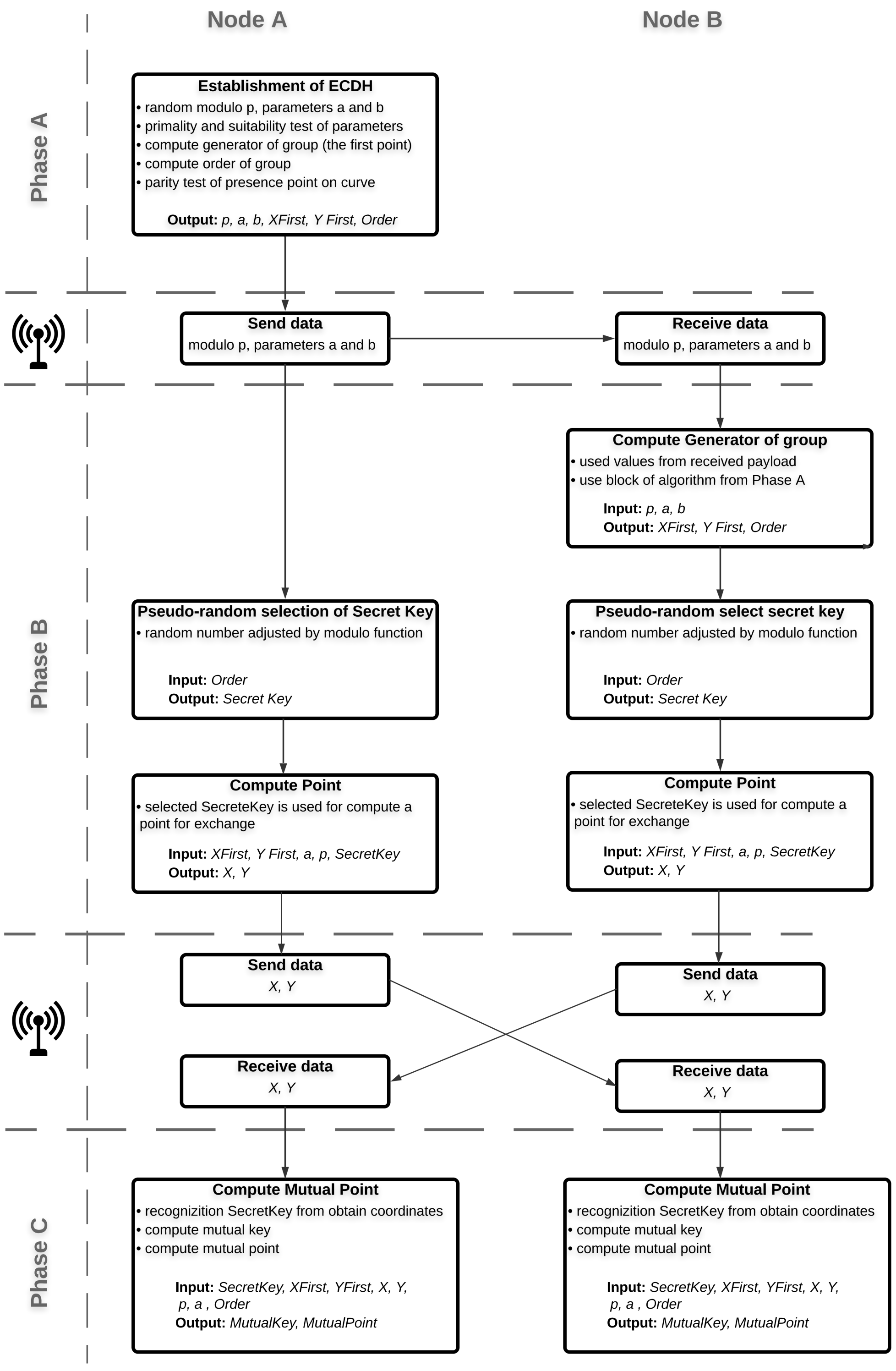

Node A
Node B
Phase A
Establishment of ECDH
• random modulo p, parameters a and b
• primality and suitability test of parameters
• compute generator of group (the first point)
• compute order of group
• parity test of presence point on curve
Output: p, a, b, XFirst, Y First, Order
Send data
modulo p, parameters a and b
Receive data
modulo p, parameters a and b
Phase B
Compute Generator of group
• used values from received payload
• use block of algorithm from Phase A
Input: p, a, b
Output: XFirst, Y First, Order
Pseudo-random selection of Secret Key
• random number adjusted by modulo function
Input: Order
Output: Secret Key
Pseudo-random select secret key
• random number adjusted by modulo function
Input: Order
Output: Secret Key
Compute Point
• selected SecreteKey is used for compute a point for exchange
Input: XFirst, Y First, a, p, SecretKey
Output: X, Y
Compute Point
• selected SecreteKey is used for compute a point for exchange
Input: XFirst, Y First, a, p, SecretKey
Output: X, Y
Send data
X, Y
Send data
X, Y
Receive data
X, Y
Receive data
X, Y
Phase C
Compute Mutual Point
• recognizition SecretKey from obtain coordinates
• compute mutual key
• compute mutual point
Input: SecretKey, XFirst, YFirst, X, Y, p, a , Order
Output: MutualKey, MutualPoint
Compute Mutual Point
• recognizition SecretKey from obtain coordinates
• compute mutual key
• compute mutual point
Input: SecretKey, XFirst, YFirst, X, Y, p, a , Order
Output: MutualKey, MutualPoint

Fig. 5.1: Key exchange algorithm scheme

prime, the loop is repeated, i.e., re-selected and tested. If the selection is successful, the selections of parameters a and b follow, representing the asymptotes of the elliptic curve. Selected values are tested according to the equation:

$$(4 * a^3 + 27 * b) \bmod p \neq 0. \tag{5.1}$$

In the case of a negative test result, the loop is repeated until suitable values are selected and until no equation results in zero. Both tests are designed as separate blocks, which the main algorithm calls as needed. Figure 5.2 shows a run of an application for testing a prime number, using a flowchart for illustration.

```
// psudo-random number
bdRandomSeeded(premod, 512, (const unsigned char*)"", 0,
   RanodmNumber);

// Rabin-Miller - 10 rounds iteration
PrimeTest = bdRabinMiller(mod, 10);

```

Listing 5.1: Method of selection and verification the prime number

In the next part of this application phase, the coordinates of the first point, which represent the group generator, are calculated. This step is mediated by a block of code, marked TheFirstPointBIGD, and it is an essential element in assembling a group of points. It is based on comparing values from two tables. The data from these tables are outputs from separate XPart and YPart applications. Flowcharts for these applications are included in the appendicies. The calculation method is illustrated in Table 5.1 and listing of algorithm. If the first match is recorded, the resulting coordinates are stored in variables Xfirst and Yfirst. These coordinates represent the first point or group generator. TheFirstPoint computes the whole group in the same way. Incremental value is added to the cycle, which increases with each point found. The result of this value is the number of points, which represents the order of the group.

The obtained coordinates are further tested. This test checks if the X and Y coordinates lie on the curve. This solution was chosen due to problems during the creation of the algorithm. Despite appropriately selected and tested input values, the calculated groups of points were non-parity and, therefore, unusable for asymmetric key exchange. , the coordinates of the first point and the order of the group. If this test is not succesfull, the aplication returns to beginning.

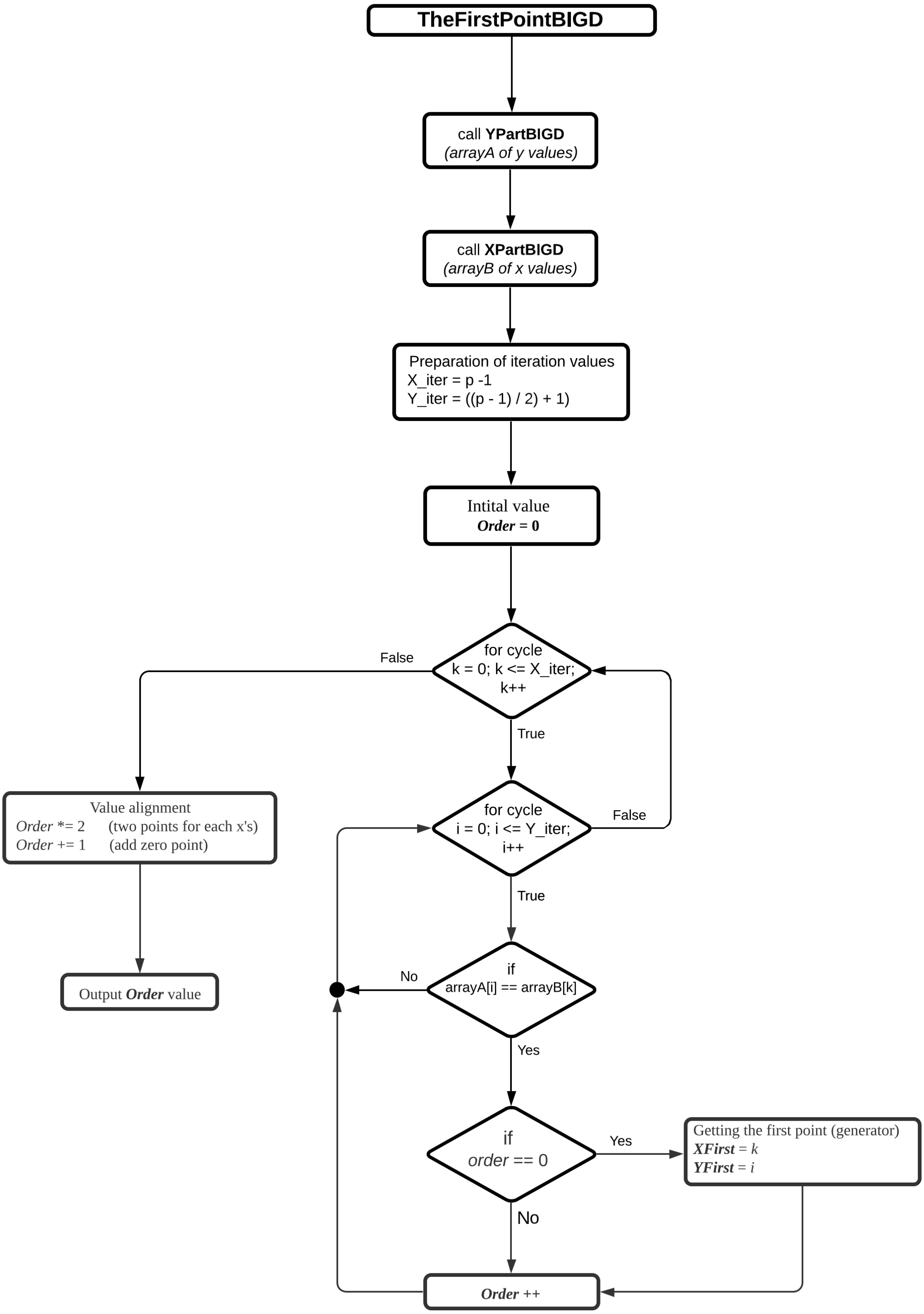
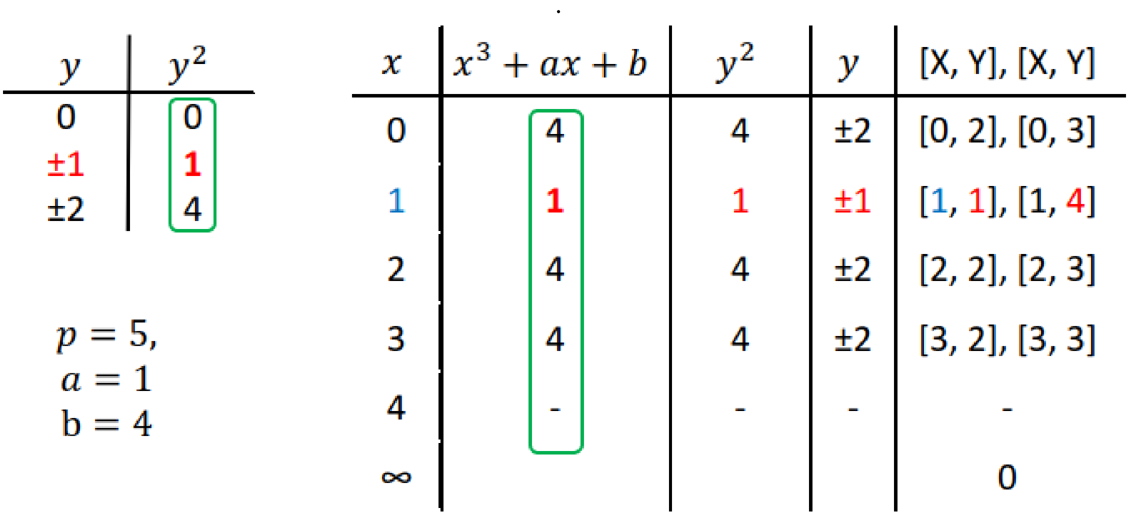


Fig. 5.2: Algorithm for compute the generator and order of the group

Tab. 5.1: Method of obtain generator (first point) of the group



y	y^2
0	0
±1	1
±2	4

$p = 5$,
$a = 1$
b = 4

x	$x^3 + ax + b$	y^2	y	[X, Y], [X, Y]
0	4	4	±2	[0, 2], [0, 3]
1	1	1	±1	[1, 1], [1, 4]
2	4	4	±2	[2, 2], [2, 3]
3	4	4	±2	[3, 2], [3, 3]
4	-	-	-	-
∞				0

5.2 Phase B

This phase has the task of selecting the key and calculating the point that represents the public key to be sent to the counterparty. The application is divided into two subversions with respect to where it is used. When used on the communication initiator side (Node A), it contains only a block for generating a pseudo-random secret key and calculating a sharing point (coordinates X and Y). The version for the communication recipient is essentially the same but also contains a block that calculates the first point and order of the group by knowing the first number and the asymptote of the curve. This solution was chosen due to save energy on the development boards. For the exchange of the initial public parameters, it is enough to send the recipient only the mentioned values of the module and parameters. This calculation is not verified in any way. There is a confidence that the values received from the initiator are usable for the calculation. If these values are forged, the key exchange cannot be successful.

The secret key selection algorithm block works with a pseudo-random number that is created by a generator that is based on the function bdRandomSeeded and srand() function which is already part of the Lightweight stack.. Within this block, the pseudo-random number is adjusted using a known group order value, which is generated in Phase A. This is because the key's value ranges from one to the group order value. In addition, this block contains anti-risk treatment so that the resulting key is not zero.

5.3 Phase C

The final phase of the algorithm follows the mutual exchange of points, which represent the public keys. This application consists of two applications, PointCompBIGD - see the flowchart bellow, and MagnifierBig. The entry to the PointComp application is our secret key and the point received from the second communication node. The secret key represents the number of iterations and the point received here represents the new generator of group. These applications are described in more detail in chapter 5.4. The output of this phase is a 128-bit key. This key is then used to encrypt the communication. The key is entered using a block taken from the LWM stack.

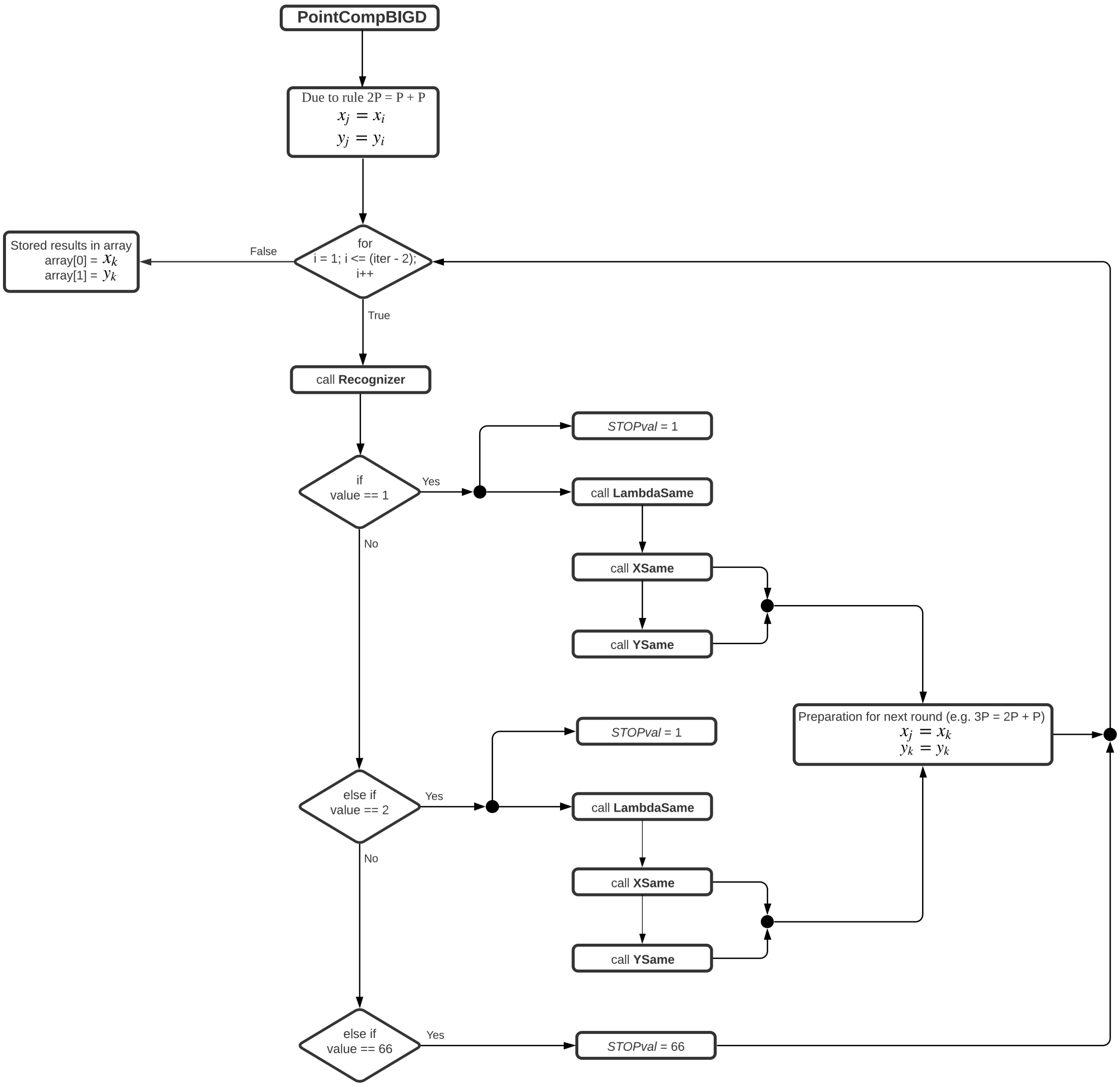


Fig. 5.3: The PointCompBIGD application

5.4 Overview of ECDH function

In this section, we will briefly introduce the blocks of the ECDH algorithm. These blocks are based on mathematical operations, which are described in the chapter 4 and also make frequently use of functions from the BIGD library [12].

ECDH_Phase_A
In the case of the ECDH_PPHASE_A_Big application, this is a summary of several sub-applications. The purpose of this application is to automatically select the modulo value by usage a pseudo-random function and then use the Miller-Rabin test to verify whether the selected value is really a prime number. In the next step, this application again uses the psudorandom functions to select the values of asymptotes, a and b. Subsequently, the first point of the group is calculated from these three values, which will be used to calculate the remaining points in the group. The ECDH_Phase_A application ends with a test of whether the selected point actually lies on the curve. If this test is evaluated incorrectly, the FazeA application is started again from the beginning. For more detail the flowchart of this phase is part of the appendices.

ECDH_PHASE_BA_Big
The task of this phase is to select the secret key, which is selected pseudo-randomly, and then calculate the coordinates of the point that will be shared with the counterparty. This aplication works with obtained values from the previous application. Part of this application is the SecreyKeyBIGD subapplication, which generates a pseudo-random key. The randomness of the selection is controlled by adopted and edited algoritm for generate pseudo-random number, where is added block of code from LWM stack. This change ensures that each iteration of the block generates a different number - see a listening of code bellow. This key is also adjusted to the size of group. In the next step, the PointCompBIGD function is called, which iteratively calculates the coordinates of the point which to be exchanged.

```c
int RanodmNumber(unsigned char *bytes, size_t nbytes, const unsigned char *seed, size_t seedlen){
  unsigned int myseed;
  size_t i;
  int offset;

  /* Use time - then blend in seed, if any */
  myseed = (unsigned)time(NULL);
  if (seed)
  {
    for (offset = 0, i = 0; i < seedlen; i++, offset = (offset + 1) % sizeof(unsigned))
      myseed ^= ((unsigned int)seed[i] << (offset * 8));
  }

  //srand(myseed);             // Original command
  srand(PHY_RandomReq());    //  Random number generator from LWM mesh
  while (nbytes--)
  {
    *bytes++ = rand() & 0xFF;
  }

  return 0;
}

 // Generator of 128 bitsnumber and RandomNumber algorithm
bdRandomSeeded(a, 128, (const unsigned char*)"", 0, RanodmNumber);
```

Listing 5.2: Generator pseudo-random numbers

ECDH_PHASE_BB_Big

This application is purposefully and principally identical to the previous BA version. This application is supplemented by the calculation of the first point and the order of the group. This is due to the fact that during the first data exchange between the boards, Node A sends only the modulo and asymptotes values to Node B.

ECDH_PHASE_C_Big

After the second data exchange, when node A and node B exchange the coordinates of the points, phase C follows, which is summarized in the application ECDH_PHASE_C_Big. This application works with our secret key and the point received from the other communication node. Using the PointCompBIGD subapplication we are able to compute a common point from these values.

checkValABBIGD

This application is based on a formula 5.1 described in chapter 5. The task of the application is to verify whether the tested X and Y coordinates actually lie on the curve. The application uses functions with the BIGD library [12] and it is therefore possible to test numbers with a bit value higher than 32 bits.

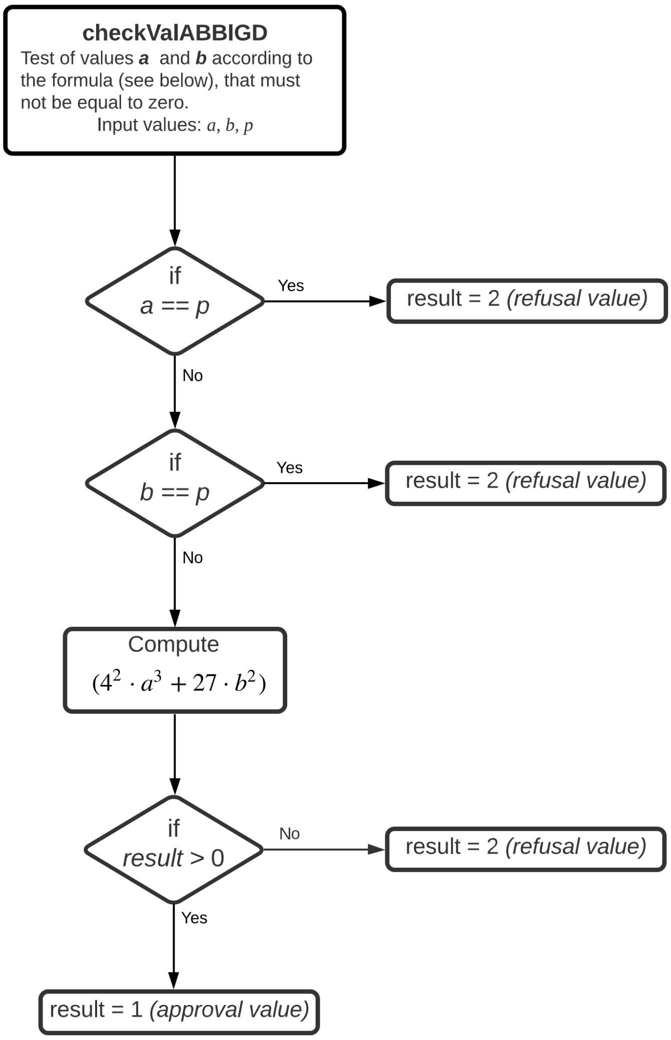

checkValABBIGD
Test of values *a* and *b* according to the formula (see below), that must not be equal to zero.
Input values: a, b, p
if $a == p$
Yes
result = 2 *(refusal value)*
No
if $b == p$
Yes
result = 2 *(refusal value)*
No
Compute
$(4^2 \cdot a^3 + 27 \cdot b^2)$
if *result* > 0
No
result = 2 *(refusal value)*
Yes
result = 1 *(approval value)*

Fig. 5.4: The checkValABBIGD application

MagnifierKeyBig

When calculating the coordinates, t is possible that the points will not have the required bit size. This problem is solved by application MagifierKeyBig. This application will increase the received values to the size that is required.

SumTwoPointsBIGD

The purpose of this application is the sum of two points. The application contains the RecogniserBIGD subroutine, which decides whether to use the method for calculating two identical points or the method for calculating two different points. The output of this application is a common point.

RecogniserBIGD

According to the rules for counting points on elliptic curves, there can be two cases where two identical or two identical points are added. This problem is solved by the PointCompBIGD application, which recognizes points and returns the value that the aplication for the sum points - PointCompBIGD, uses to select the appropriate method.

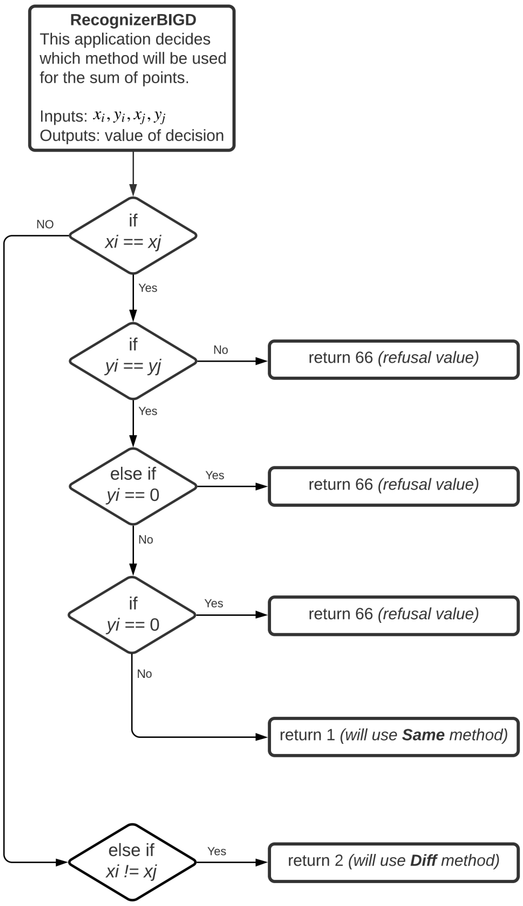

RecognizerBIGD
This application decides which method will be used for the sum of points.
Inputs: x_i, y_i, x_j, y_j
Outputs: value of decision
if xi == xj
NO
Yes
if yi == yj
No
return 66 (refusal value)
Yes
else if yi == 0
Yes
return 66 (refusal value)
No
if yi == 0
Yes
return 66 (refusal value)
No
return 1 (will use **Same** method)
else if xi != xj
Yes
return 2 (will use **Diff** method)

Fig. 5.5: RecongnizerBIGD application

PointCompBIGD

This application is similar to SumTwoPointsBIGD. However, it is supplemented by an iteration cycle that calculates a certain point in the group. This application needs to be used wisely because the number of iterations is limited to a 32-bit number for the for loop.

TheFirstPointBig

The task of this application is to calculate the proving point, as know as a group generator. The function, including the mathematical background, is explained in the chapter 5.1. The application consists of XPartBIGD and YPartBIGD applications, which calculate the X and Y tables. The data from these tables are compared and a group generator is obtained by this method.

6 Implementation ECDH on LWM

In this chapter, we will introduce our communication protocol solution, which is based on the LightWeight Stack. During the development, the manufacturer's manual was used. The main task of this protocol is the transfer of data between the individual phases of our key exchange algorithm.

We will gradually explain the individual parts of the protocol, which will be presented here in the form of code listings. The first subchapter deals with the implementation of the whole structure of ECDH, which corresponds to the scheme 5.1 in the chapter 5. The next subchapter will explain the solution of data forwarding to create a payload block

6.1 TaskHandler

The main part of the protocol that controls individual calls and terminations is the APP_Task Handler block. This block is called from the main block in infinite loop - see the listing 6.1. It was therefore necessary to design a method for switching individual code sequences. This was done using by the switch command. The individual states represent the individual phases of the key exchange application and correspond to the flowchart 5.1 in the chapter 5.1.

```
int main(void)
{
    // Serial communication for use command printf for check results
  usb_init();

  SYS_Init(); // inicialization
  while(1)
  {
    SYS_TaskHandler();

    // Inicialization block which contains our application
    APP_TaskHandler();

  }
}
```

Listing 6.1: The main block

The initial state is APP_State_Init, which starts the application and immediately switches to state the APP_STAT_PhaseA by changing the value of appState. At that stage, there is already an application for selecting and computing the initial values as modulo, a, b, etc.

Switching to phase B and phase C is realized by conditions which, using the value comparison function, wait for the value to change at the output of the previous application. This verified that the previous application was running and returned specific values.

```c
switch (appState)
{
case APP_STATE_Init:
{
  appInit();
  appState = APP_STATE_FazeA;

} break;

case APP_STATE_FazeA:
{
  ECDH_PHASE_A_BIGD(MOD, a_parameter, b_parameter, resultsA);
  appState = APP_STATE_FazeB;  // switch to phase B

} break;

case APP_STATE_FazeB:
{
  if(comaparsionA != 0){    // check of receiving data
    appState =  APP_STATE_FazeC;   // switch to phase C
    }
} break;

case APP_STATE_FazeC:

if(comaparsionB != 0){          // check of successful completion
  appState =  APP_STATE_END;  // switch to ending state
}

APP_STATE_END:
break;
```

Listing 6.2: Management of phases

6.2 Definition of payload

Before we send the data, it is necessary to define the form of the sent payload. We need to specify what data to send, what is the source of the message and what is the target. All of these specification, and not just those, are addressed by this block of code.

The block of code that we can see below 6.2 is used to transfer data from node A to node B. The data here represent three values, namely the modulo value and asymptotes of a curve. It is assumed that some values will be larger than a 32-bit number, so in the first step, objects for transferring large numbers are activated using the bdNew() function. However, this is not a definition of variables. These are defined hierarchically above in the program and thus behave as global variables. By activating the block, a certain amount of memory will be reserved. At the end of the process, it is advisable to deactivate unnecessary variables again. This eliminates the risk of memory overflow.

In the next steps, the payload parameters are defined. These parameters include the destination address to which the data will be sent. Next, we set the source and target endpoints here. For example, if a NodeA receives data, it can send it to a specific application in its memory, which is addressed by an endpoint number. In the options item it is possible to activate transmission encryption, which is based on the AES algorithm. The data parameter represents the input for the data we want to send. An important part of this block is NWK_DataReq_t, which ensures the sending of data.

In the case of the second data exchange, when both communication nodes exchange data with each other in the form of point coordinates, a similar method is used, which we can see in the code listing 6.2.

```
static void SendDataToBB(void)
{
  // temporary varible for stored data as array
  BB data[6];

  // BIG blocks activation
  for(uint8_t i = 0; i <= 5; i++){

    data[i] = bdNew();
  }

  // Insert data
  data[0] = MOD;           // MOD
  data[1] = a_parameter;   // A_parametr
  data[2] = b_parameter;   // b_parametr
  data[3] = resultsA[0];   // Xf
  data[4] = resultsA[1];   // Yf
  data[5] = resultsA[2];   // Order


  // Send data from endpoint 1 to endpoint 2
  appDataReq.dstAddr = 0xFFFF;
  appDataReq.dstEndpoint = 2;
  appDataReq.srcEndpoint = 1;
  appDataReq.options = NWK_OPT_ENABLE_SECURITY;
  appDataReq.data = data;
  appDataReq.size = 7;
  appDataReq.confirm = appDataConf;
  NWK_DataReq(&appDataReq);


  appDataReqBusy = true;

  // blocks deactivation
  for(uint8_t i = 0; i <= 5; i++){

    bdFree(&data[i]);             // block deactivation
  }
}
```

Listing 6.3: Payload for sending data to the Node B

```
static void SendDataToC(void)
{
    // temporary varible for stored data as array
    BB data[3];

  for(uint8_t i = 0; i <= 2; i++){

    data[i] = bdNew();              // block DEactivation
  }


  data[0] = resultsBA;  // X
  data[1] = resultsBA;  // Y


  appDataReq.dstAddr = 0xFFFF;
  appDataReq.dstEndpoint = 3;
  appDataReq.srcEndpoint = 2;
  appDataReq.options = NWK_OPT_ENABLE_SECURITY;
  appDataReq.data = data;
  appDataReq.size = 4;
  appDataReq.confirm = appDataConf;
  NWK_DataReq(&appDataReq);


  appDataReqBusy = true;


  for(uint8_t i = 0; i <= 2; i++){

    bdFree(&data[i]);            // block DEactivation
  }


}
```

Listing 6.4: Payload for sending data to the Node C

6.3 Data receiving

The procedure for receiving data is similar to that for sending them. In the code listing below we can see the code, which consists of commands for activating blocks for large numbers. There are also pointers that insert the received data to the specified variables. This block includes, among other things, the application which represents phase B, which is explained in more detail in the chapter 5.4. The output values from this application are written to global variables. Before the end of the block and deactivation of objects for large numbers, the output values are checked to see if the output is greater than zero. This is proof that the application generated a certain result. The resulting value of the comparison test changes the appState variable - see variable 6.1 in the main block. This change will switch the program to the next phase.

The block for receiving data from phase B is similar to the previous block, which is used for receiving data from phase A. Its task is to take over the received data from phase B and further process them using the application ECDH_PHASE_C_BIGD and obtain a common key. This block is supplemented by a function that converts the object to large numbers on char strings. It is further converted to an integer. This value is inserted into a variable named key that enters the application as a key - see listing below.

```
void NWK_SetSecurityKey(uint8_t *key)
```

Listing 6.5: The variable containing the value of key

```
static bool SendFromAToB_BB(NWK_DataInd_t *ind)  // Version BA
{

  // Activate blocks
  MOD = bdNew();
  a_parameter = bdNew();
  b_parameter = bdNew();
  resultsBB[0] = bdNew();
  resultsBB[1] = bdNew();
  ZEROB = bdNew();

  // Set zero value
  bdSetZero(ZEROB);

  // data pointed
  MOD = ind->data[0];
  a_parameter = ind->data[1];
  a_parameter = ind->data[2];

  // Run the aplication for Phase B
  ECDH_PHASE_BB_BIGD(MOD, a_parameter, b_parameter, resultsBB);

  // Variable status control for switching to the next phase
  comaparsionB = bdCompare(resultsBB[1], ZEROB);

  // Deactivate block
  bdFree(&MOD);
  bdFree(&a_parameter);
  bdFree(&b_parameter);

}
```

Listing 6.6: Block for receiving data from the Node A

```c
static bool ExchangePoint(NWK_DataInd_t *ind)
{
    // BIG blocks activation
  Xm = bdNew();
  Ym = bdNew();
  Xo = bdNew();
  Yo = bdNew();
  MOD = bdNew();
  a_parameter = bdNew();
  ZEROB = bdNew();

  // Set zero for comparsion process
  bdSetZero(ZEROB);

  // BIG block activation
  KEY = bdNew();

  // Start application of Phase C
  ECDH_PHASE_C_BIGD(Xm, Ym, Xo, Yo, MOD, a_parameter, KEY);

  // check if previous application get the key
  comaparsionB = bdCompare(KEY, ZEROB);

   // convert BIG variable KEY to char s
  bdConvToDecimal(KEY, s, sizeof(s));

  // convert char to integer
  key = atoi(s);


  // BIG blocks deactivation
  bdFree(&Xm);
  bdFree(&Ym);
  bdFree(&Xo);
  bdFree(&Yo);
  bdFree(&MOD);
  bdFree(&a_parameter);

  bdFree(&KEY);

}
```

Listing 6.7: Block for receiving data from phase B

Conclusion

The main tasks of this work were to design an algorithm for asymmetric key exchange and implement it into the Lightweight mesh network stack. The output of the work was an algorithm, which is divided into several phases and based on the theory of cryptography on elliptic curves. It is completely autonomous in the selection of input values, which changes with each established communication. This feature prevents an unauthorized third party from reusing an existing key. The algorithm is designed with the greatest possible modularity in mind. Thanks to this philosophy, the resulting code is clear and very easy for future improvements. It will not be necessary to implement the same function in more places; it will be enough to modify only the inside of the block.

Unfortunately, the second task within this bachelor's thesis was not completely realized. Although the design followed the available instructions and sample examples, the issue of radio communication between the development boards was not resolved by the time the work was submitted. The problem was likely in the APP_TaskHandler block definition, which incorrectly calls the necessary functions to transfer data between communicating points.

The main benefit of this work is acquaintance with microcontrollers and a solid knowledge of the basics of the C language. None of these areas were known to the author at the beginning of the work. The motivation for current and future improvement in these areas is practical activities in everyday life or employment.

Bibliography

[1] *ATmega256RFR2 ATmega128RFR2 ATmega64RFR2 Datasheet.* [online], 2014. In: . Atmel (now Microchip Corporation), s. 611 [cit. 2019-12-02]. Available from: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8393-MCU_Wireless-ATmega256RFR2-ATmega128RFR2-ATmega64RFR2_Datasheet.pdf

[2] *User manual - deRFnode / deRFgateway*, 2014. Dresden Elektronik, 56 s. Available from: https://www.dresden-elektronik.de/funktechnik/uploads/media/deRFnode_deRFgateway-BHB-en_10.pdf

[3] MANN, Burkhard, 2003. *C pro mikrokontroléry: ANSI-C, kompilátory C, spojovací programy - linkery, práce s ATMEL AVR a MSC-51, příklady programování v jazyce C, nástroje pro programování, tipy a triky ...* Praha: BEN - technická literatura. ISBN 80-730-0077-6.

[4] MATOUŠEK, David, 2006. *Práce s mikrokontroléry ATMEL AT89C2051: [měření, řízení a regulace pomocí několika jednoduchých přípravků].* Práce s mikrokontroléry ATMEL AVR ATmega16. Praha: BEN - technická literatura. ISBN 80-730-0048-2.

[5] LAVANYA, M. and V. NATARAJAN, 2017. *Lightweight key agreement protocol for IoT based on IKEv2.* **64**, 580-594. DOI: 10.1016/j.compeleceng.2017.06.032. ISSN 00457906. Available from: https://linkinghub.elsevier.com/retrieve/pii/S0045790617319286

[6] HEROUT, Pavel, 2011. *Učebnice jazyka C.* Dotlač 6. vyd. České Budějovice: Kopp nakladatelství. ISBN 978-80-7232-383-8.

[7] LEVICKÝ, Dušan, 2016. *Kryptografia a bezpečnosť komunikačných sietí.* Košice: Elfa. ISBN 978-80-8086-254-1.

[8] BURDA, Karel, 2015. *Úvod do kryptografie.* Brno: Akademické nakladatelství CERM, 66 s. ISBN 978-80-7204-925-7.

[9] *Cybersecurity for industry 4.0*, 2017. New York, NY: Springer Berlin Heidelberg. ISBN 978-331-9506-593.

[10] *C Program for Extended Euclidean algorithms* [online], In: . [cit. 2019-12-19]. Available from: https://www.geeksforgeeks.org/c-program-for-basic-and-extended-euclidean-algorithms-2/

[11] *Secure Hash Algorithm (SHA-1)* [online], In: . [cit. 2019-12-19]. Available from: http://www.hoozi.com/post/b3mf9/secure-hash-algorithm-sha-1-reference-implementation-in-c-c

[12] *BigDigits multiple-precision arithmetic source code* [online]. 2020 [cit. 2020-08-07]. Available from: https://www.di-mgt.com.au/bigdigits.html

List of acronyms

AES	Advanced Encrypt S
JTAG	Joint Test Action Group
AVR	Alf and Vegard's RISC processor
SHA	Secure Hash Algorithm
IEEE	Institute of Electrical and Electronics Engineers
ISP	In System Programming
WSN	Wireless Sensor Network
IoT	Interner of Things
MCU	Interner of Things
EEPROM	Electrically Erasable Programmable Read-Only Memory
ESP	Encapsulating security protocol
NHC	Next header protocol
AH	Authentication header
LKA	LightWeight key agreement
ISAKMP	Internet Security Association and Key Management Protocol

List of appendices

Flowcharts

- All Flowcharts.pdf
- ECDH Equations.pdf
- ECDH_PHASE_A_BIGD.pdf
- ECDH_PHASE_BA_BIGD.pdf
- ECDH_PHASE_BB_BIGD.pdf
- ECDH_PHASE_C_BIGD.pdf
- checkValABBIGD.pdf
- MagnifierKeyBig.pdf
- PointCompBIGD.pdf
- RecogniserBIGD.pdf
- SecretKeyBIGD.pdf
- SumTwoPointsBIGD.pdf
- TheFirstPointBIGD.pdf
- VerifyOfPointBIGD.pdf
- XPartBIGD.pdf
- YPartBIGD.pdf

Flowcharts

- PHASE A (EXE) autonomous.exe
- PHASE A (EXE) manual.exe

Source of code

- ECDH_Phase_A_Big.c
- ECDH_Phase_BA_Big.c
- ECDH_Phase_BB_Big.c
- ECDH_Phase_C_Big.c
- checkValABBig.c
- LambdaDiffBig.c
- LambdaSameBig.c
- MagnifierKeyBig.c
- PointCompBig.c
- RecognizerBig.c
- SecretKey_Big.c
- SumTwoPoints_Big.c
- TheFirstPointBig.c
- VerifyOfPointBig.c
- XDiffBig.c
- XPartBig.c

- XSameBig.c
- YPartBig.c
- YUniBig.c