UNIVERSITY OF SPLIT

FACULTY OF ELECTRICAL ENGINEERING, MECHANICAL ENGINEERING AND NAVAL ARCHITECTURE

SECURE BLE

Mentor: Mario Čagalj

Students: Ana Pandža

Marin Peko

Split, January 2019

Contents

1 Introduction

In the last few decades, wireless communication has been growing and developing to that point that today we cannot imagine everyday life without it. Because of that, there is a need to make sure that wireless communication and data transfer is secured and protected. Alongside wireless communication came Internet of Things, i.e. IoT, - network of devices that share data over Internet. That network includes sensors, vehicles, home systems and so on. While some data is public and there is no harm if third party gets it, e.g. current temperature on Times Square, what about sensitive data, private and personal information that can be used against someone? We need to make sure that only those who have rights can read that information.

The goal of this project is to establish secure wireless communication between Arduino Uno microcontroller and Android application over Bluetooth Low Energy – BLE. Secure communication implies covering both, privacy and authentication. Privacy aspect of security is achieved by using Elliptic-Curve Diffie-Hellman key exchange protocol and authentication aspect is achieved by using Station to Station protocol.

In this paper, an overview of system architecture and implementation of that system is given. Also, main concepts used within this project, along with protocols and technologies are explained.

2 Bluetooth Low Energy

This chapter gives an overview of the technology that is used in communication between Android Uno microcontroller and Android application. This technology is called Bluetooth Low Energy, popularly known BLE.

BLE is a wireless personal area network technology designed to provide lower power consumption, so it is suitable for devices with low memory and energy such as sensors and monitors. [1]

2.1 Specifications

BLE specifications are:

- > 100 m range
- 1 Mbit/s over the air data rate
- not specified number of active slaves
- 6 ms latency

2.2 Security

BLE connections are specific because of the pairing process between devices so that secure connection can be established. This process varies between BLE version 4.2 and older ones (versions 4.1 and 4.0). In BLE 4.2 devices use Elliptic-Curve Diffie-Hellman key exchange protocol as a part of their pairing process. However, the problem with using only Elliptic-Curve Diffie-Hellman key exchange protocol is that it does not give the user a way to verify the authenticity of the connection. Therefore, it is still vulnerable to MITM attacks.

Security issues present when working with BLE are:

- passive eavesdropping
- MITM attacks
- identity tracking

2.3 BLE stack

In this subchapter two top layers of BLE stack will be explained. These layers are the most important ones while establishing connection between two devices.

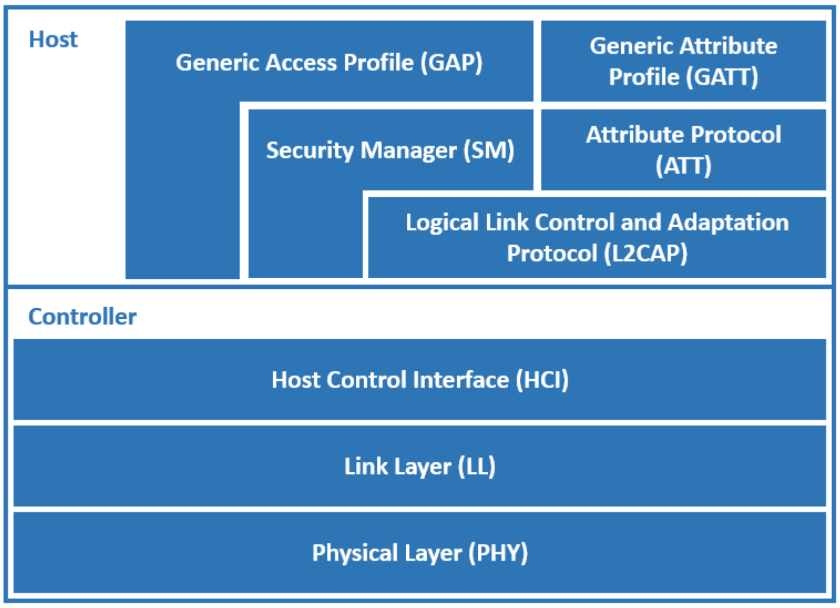

Host
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Security Manager (SM)
Attribute Protocol (ATT)
Logical Link Control and Adaptation Protocol (L2CAP)
Controller
Host Control Interface (HCI)
Link Layer (LL)
Physical Layer (PHY)

Figure 2-1 BLE stack

Generic Access Profile or GAP is the layer of the BLE stack which determines the network topology of the BLE system. Based on this layer, there are two roles in the BLE communication: BLE GAP Central and BLE GAP Peripheral. The GAP Central is typically the device which initiates the connection with the GAP Peripheral. Once the two devices are connected, they will perform a pairing process where they will exchange the information necessary to establish an encrypted connection.

Generic Attribute Profile or GATT is the layer of the BLE stack used by the application for data communication between the two connected devices. Data is passed and stored in the form of characteristics which are stored in memory of BLE device. When two devices are connected, they obtain one of two roles: GATT Server and GATT Client. The GATT Server is the device containing the characteristic database that is being read or written by a GATT Client. We can consider characteristics as groups of information called attributes.

A typical characteristic is composed of the following attributes:

- characteristic value
- characteristic declaration
- client characteristic configuration
- characteristic user description [3]

2.4 HM-10

For the purpose of this project, BLE Module HM-10 is being used. This is the module for embedded system to get BLE wireless communication with BLE capable devices. It is configurable with AT, i.e. Attention, command-set and allows transparent data communication via serial UART (default baudrate 9600 bps). Since HM-10 is a BLE 4.0 module, it’s pairing process does not include Elliptic-Curve Diffie-Hellman key exchange protocol.

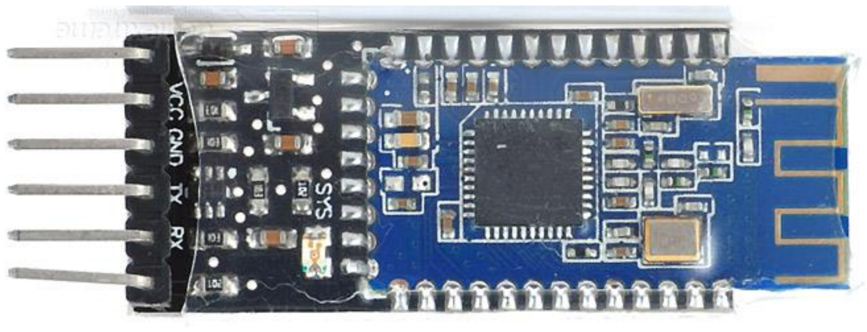


Figure 2-2 HM-10

Table 2-1 shows how to wire up HM-10 BLE module to Arduino Uno microcontroller.

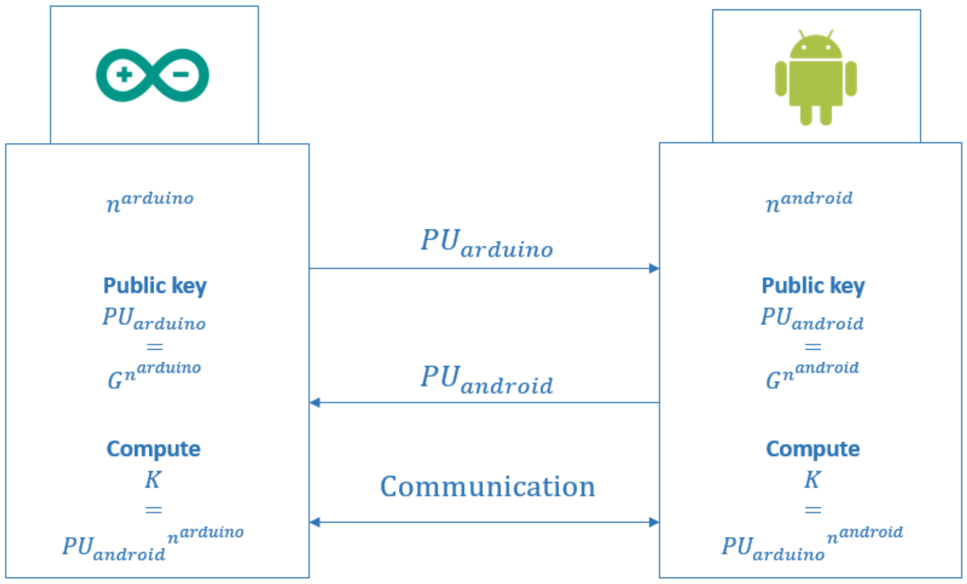
ARDUINO UNO	HM-10 BLE MODULE
D2	TX
D3	RX
GND	GND
3.3 V	VCC

Table 2-1 Wiring Arduino Uno and HM-10

3 Elliptic-Curve Diffie-Hellman key exchange protocol

Elliptic-Curve Diffie-Hellman (ECDH) is a key agreement protocol that allows two parties, each having an elliptic-curve public-private key pair, to establish a shared secret over an insecure channel [4]. In this project, those parties are Arduino microcontroller and Android application. Generated shared secret is used for encryption of messages sent over BLE.

Figure 3-1 shows how two entities (in this case, Arduino Uno microcontroller and Android application) communicate with each other in order to establish the shared secret.



$n^{arduino}$
Public key
$PU_{arduino} = G^{n^{arduino}}$
Compute
$K = PU_{android}^{n^{arduino}}$
$PU_{arduino}$
$PU_{android}$
Communication
$n^{android}$
Public key
$PU_{android} = G^{n^{android}}$
Compute
$K = PU_{arduino}^{n^{android}}$

Figure 3-1 ECDH key exchange protocol

Arduino Uno microcontroller first generates its private key $PR_{arduino} = n^{arduino}$, an integer less than order *n*, and its public key $PU_{arduino} = G^{n^{arduino}}$ which belongs to a point on elliptic curve. Android application does the same thing.

After exchanging each other's public key, Arduino Uno microcontroller and Android application generate shared secret

$$K_{arduino} = PU_{android} \cdot PR_{arduino} = PU_{android} \cdot n^{arduino}, \text{ i.e.}$$

$$K_{android} = PU_{arduino} \cdot PR_{android} = PU_{arduino} \cdot n^{android}, \text{ where}$$

$$K_{arduino} = K_{android}.$$

Generated shared secret ensures encrypted communication between two entities.

However, since Elliptic-Curve Diffie-Hellman key exchange protocol ensures only privacy, and not authentication, which is why its usage is vulnerable to Man in The Middle (MITM) attacks, new protocol is being introduced on top of the Elliptic-Curve Diffie-Hellman key exchange protocol. This protocol is called Station-to-Station protocol and more words about it will be given in the following chapter.

4 Station-to-Station protocol

As already said, Station-to-Station protocol is being used on top of the Elliptic-Curve Diffie-Hellman key exchange protocol in order to cover authentication aspect of secure communication.

Figure 4-1 shows how Station-to-Station protocol encapsulates Elliptic-Curve Diffie-Hellman protocol.

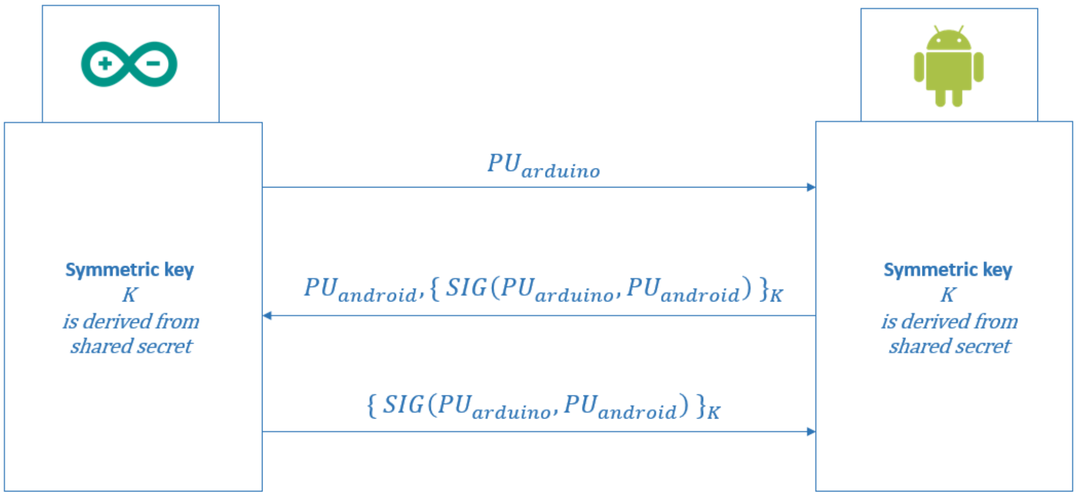


Figure 4-1 Station-to-Station protocol

First, one of the entities (in this case it is Arduino Uno microcontroller) sends his public key across the communication channel. Then, the second entity (in this case it is Android application) returns his public key along with the symmetrically encrypted signature of concatenated public keys of both entities. Signature is encrypted with symmetric i.e. session key which is derived from previously generated shared secret. At the end, first from two entities also sends his symmetrically encrypted signature of concatenated public keys of both entities.

Symmetric, i.e. session key, is created by applying SHA-256 cryptographic hash function on the shared secret generated during Elliptic-Curve Diffie-Hellman key exchange protocol. Due to hash functions' one-wayness, nobody should be able to break it in order to expose shared secret. For symmetric encryption Advanced Encryption Standard, i.e. AES, is used with the key length of 256 bits.

Once both entities are authenticated, secure communication between them over Bluetooth Low Energy can start.

5 State machine

State machines, in general, represent a set of complex rules and conditions. For the purpose of this project, state machine is defined in order to simplify execution of both, Elliptic-Curve Diffie-Hellman key exchange protocol and Station-to-Station protocol, and also to follow their states and transitions. The same state machine needs to be implemented on both sides of communication, Arduino Uno microcontroller and Android application.

Figure 5-1 represents the state machine used within this project.

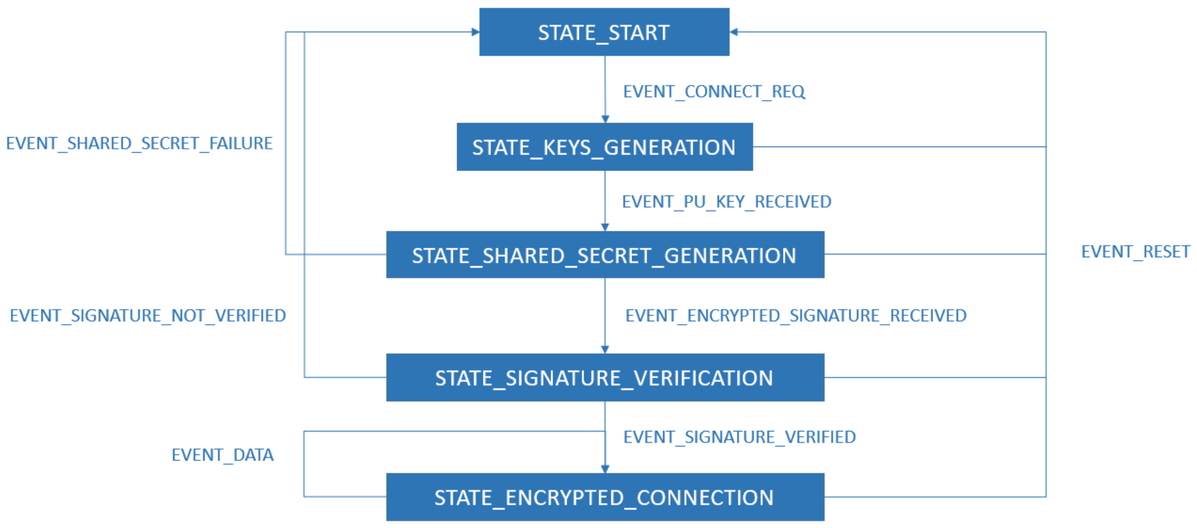

STATE_START
EVENT_CONNECT_REQ
STATE_KEYS_GENERATION
EVENT_SHARED_SECRET_FAILURE
EVENT_PU_KEY_RECEIVED
STATE_SHARED_SECRET_GENERATION
EVENT_RESET
EVENT_SIGNATURE_NOT_VERIFIED
EVENT_ENCRYPTED_SIGNATURE_RECEIVED
STATE_SIGNATURE_VERIFICATION
EVENT_SIGNATURE_VERIFIED
EVENT_DATA
STATE_ENCRYPTED_CONNECTION

Figure 5-1 State machine

In Table 5-1 detailed description of each and every state in the state machine is given along with the list of actions executed on the entry to that specific state.

STATE	DESCRIPTION	ON ENTRY
STATE_START	Initial state of state machine.	• clear Elliptic-Curve Diffie-Hellman private and public keys, and symmetric i.e. session key
STATE_KEYS_ GENERATION	On connection request received from another entity state machine switches to this state.	• generate Elliptic-Curve Diffie-Hellman private and public keys • send previously generated public key to another entity
STATE_SHARED_ SECRET_GENERATION	On receipt of public key from another entity (Arduino Uno microcontroller or Android application) state machine is switched to this state.	• generate shared secret from public key received from another entity
STATE_SIGNATURE_ VERIFICATION	Once the encrypted signature is received state machine switches to this state.	• decrypt received payload with symmetric i.e. session key and verify signature received from another entity
STATE_ENCRYPTED_ CONNECTION	Once the verification of exchanged signatures is finished, encrypted communication can begin. During the time secure connection lasts, state machine remains in this state.	• decrypt received payload with shared secret

Table 5-1 States

Table 5-2 shows all the events used within the state machine.

EVENT	DESCRIPTION
EVENT_CONNECT_REQ	Indicates start of Elliptic-Curve Diffie-Hellman key exchange protocol along with Station-to-Station protocol.
EVENT_PU_KEY_RECEIVED	Indicates receipt of Elliptic-Curve Diffie-Hellman public key from another entity.
EVENT_ENCRYPTED_SIGNATURE_ RECEIVED	Indicates receipt of encrypted signature from another entity.
EVENT_SHARED_SECRET_FAILURE	Indicates failure while generating shared secret.
EVENT_SIGNATURE_VERIFIED	Indicates that received signature from another entity is successfully verified, i.e. another entity is successfully authenticated.
EVENT_SIGNATURE_NOT_VERIFIED	Indicates that received signature from another entity is not verified, i.e. another entity is not authenticated.
EVENT_DATA	Indicates that another entity is sending data encrypted with shared secret.
EVENT_RESET	Indicates reset of all data used within state machine. This includes private and public keys from Elliptic-Curve Diffie-Hellman key exchange protocol as well as symmetric i.e. session key from Station-to-Station protocol.

Table 5-2 Events

6 Messages

Since only limited number of bytes (approximately 20 – 23 bytes, depending on how long are other information sent over the GATT layer of the BLE stack) can be transferred across the BLE channel at once, new message format is defined:

```
$<MESSAGE_TYPE>=<MESSAGE_CONTENT>;
```

Each message transferred over BLE channel shall begin with dollar sign and end with semicolon. Type of the message is required part of the message while content is optional. Type and content parts of the message are separated by equality sign.

In Table 6-1 shows all message types used within this project along with their description.

MESSAGE TYPE	DESCRIPTION
CONNECT	Initiates Station-to-Station and Elliptic-Curve Diffie-Hellman key exchange protocol.
PU	Carries Elliptic-Curve Diffie-Hellman public key from one entity to another.
FAILURE	Indicates unsuccessful generation of a shared secret.
SIG	Carries symmetrically encrypted signature from one entity to another.
SIGVER	Indicates successful verification of received signature.
SIGNVER	Indicates unsuccessful verification of received signature.
DATA	Carries data encrypted with shared secret from one entity to another.
RESET	Resets state machine, i.e. Station-to-Station (symmetric i.e. session key) and Elliptic-Curve Diffie-Hellman key exchange protocol (private and public keys).

Table 6-1 Message types

7 System implementation

This chapter gives an overview of system implementation, including both Arduino Uno microcontroller's and Android application's part, as well as an overview of technologies used within this project.

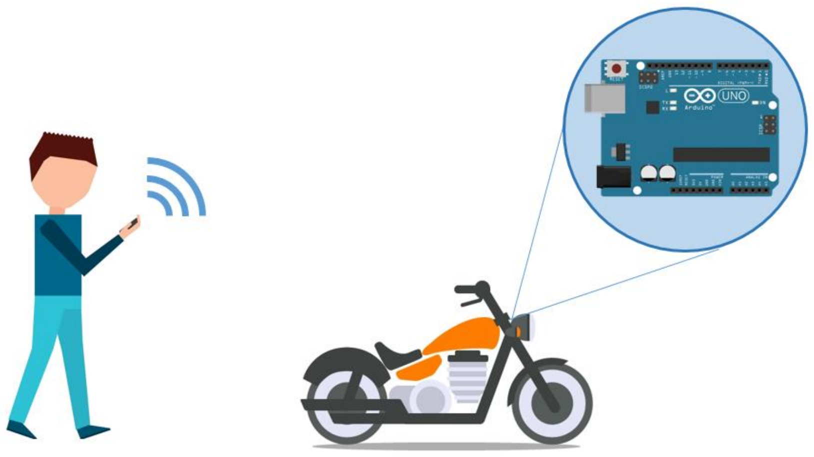


Figure 7-1 System architecture

7.1 Arduino Uno microcontroller

Arduino Uno microcontroller part of the system is implemented by using C/C++ programming languages. While implementing the system, Platformio IDE extension along with Visual Studio Code is being used.

Cryptographic and mathematical parts of implementation, such as generating Elliptic-Curve Diffie-Hellman key pair or shared secret, are realized by using an open source Arduino Cryptography Library [7].

Implementation consists of 4 main classes:

- *MessageParser*
- *StateMachine*
- *ECDHKeyExchange*
- *STS*

MessageParser class is responsible for parsing content received over Bluetooth Low Energe, i.e. BLE, communication channel. Content is parsed according to the message format specified in the previous chapter.

StateMachine class contains definition of states, events and conditions for switching the state machine from one state to another. This class encapsulates both, *ECDHKeyExchange* and *STS* class, which are responsible for generating Elliptic-Curve Diffie-Hellman key pair and shared secret, verifying the signature and symmetrical encryption / decryption.

Arduino Uno specifications	
Microcontroller	Atmega328P
Flash Memory	32 KB
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz

Table 7-1 Specifications

While implementing Arduino Uno part of the project, key thing is to have microcontroller specifications in mind because those specifications could affect performance of the software later on. In Table 7-2 performance of some operations used within this project is given.

Operation	**Time**
Generating ECDH key pair	3259 ms
Generating shared secret	3241 ms
AES256 key setup	437 us
AES256 encryption (per byte)	24 us
AES256 decryption (per byte)	46 us
SHA256 hashing	2835 us

Table 7-2 Performances

7.2 Android application

Android application part of the system is implemented by using Xamarin Forms and C# programming language. Unlike the Arduino Uno microcontroller part where an external open source cryptography library is being used, in the Android application C# built-in cryptography library is being used.

Secure communication with Arduino Uno microcontroller is initiated from Android application side. First, user of Android application needs to enable Bluetooth on his phone and scan for nearby BLE devices. Once the scanning is over, user chooses one of the devices from the list of all scanned devices. When the device is successfully selected, list of all the services and characteristics associated to that device is being shown. Furthermore, characteristic responsible for exchanging messages between two BLE devices is chosen.

8 Conclusion

Nowadays, when number of applications in technology is growing, every communication, especially wireless one, is required to be secured. Security, which is not so easy to achieve, implies both, privacy and authentication.

In this paper, it is explained how to establish secure communication between two entities over Bluetooth Low Energy. In our case, those entities are Arduino Uno microcontroller and Android application but the same concept can be applied to any other entity in the modern world. Some of the means in achieving security within this project were Elliptic-Curve Diffie-Hellman key exchange protocol and Station-to-Station protocol. These protocols cover already mentioned aspects of security, privacy and authentication.

At the end, this project represents one of many ways on how to achieve secure communication between two entities over the Bluetooth Low Energy channel. Thus, it gives solution to many real-world problems, especially in the field of IoT.

LITERATURE

[1] „Bluetooth Low Energy“, from the Internet https://en.wikipedia.org/wiki/Bluetooth_Low_Energy (16th January 2019)

[2] „A Basic Introduction to BLE Security“, from the Internet https://www.digikey.com/eewiki/display/Wireless/A+Basic+Introduction+to+BLE+Security (16th January 2019)

[3] „Generic Attribute Profile (GATT)“, from the Internet http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_40_00_45/docs/blestack/ble_user_guide/html/ble-stack-3.x/gatt.html (16th January 2019)

[4] „Elliptic-Curve Diffie-Hellman“, from the Internet https://en.wikipedia.org/wiki/Elliptic-curve_Diffie-Hellman (16th January 2019)

[5] „Station-to-Station protocol“, from the Internet https://en.wikipedia.org/wiki/Station-to-Station_protocol (16th January 2019)

[6] C. Boyd, A. Mathuria – „Protocols for Authentication and Key Establishment“

[7] „Arduino Cryptography Library“, from the Internet https://rweather.github.io/arduinolibs/index.html (16th January 2019)