RĪGAS TEHNISKĀ UNIVERSITĀTE

ELEKTRONIKAS UN TELEKOMUNIKĀCIJU FAKULTĀTE

[](https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwj9zKXEs6fgAhWF1iwKHceJAlAQjRx6BAgBEAU&url=https://lv.wikipedia.org/wiki/Att%C4%93ls:RTU_logo_2017.svg&psig=AOvVaw2YftzNrZHl3gHZWIEXy3Ih&ust=1549552627423236)

Datu pārraide bezvadu sensoru tīklos

***Laboratorijas darbs Nr.6***

I REGV0

Ruslans Babajans 171REB152

Rīga, 2021

**Darba uzdevums:**

Laboratorijas darbā tiek modificēts kods lai iekļautu datu šifrēšanu. Šīm nolūkam projektam tika pievienota small\_aes bibliotēka.

Izmantotā aparatūra: DISCO-L072CZ-LRWAN1 izstrādes platforma.

Izstrādātais c++ kods:

// UID of used kits are: 3145789, 3342368

#define MY\_MAC\_ADDRESS myUidValue

#define OTHER\_MAC\_ADDRESS 3342368

#include "mbed.h"

#include "PinMap.h"

#include "GenericPingPong.h"

#include "sx1276-mbed-hal.h"

#include "main.h"

#include "small\_aes.h"

#ifdef FEATURE\_LORA

/\* Set this flag to '1' to display debug messages on the console \*/

#define DEBUG\_MESSAGE 1

/\* Set this flag to '1' to use the LoRa modulation or to '0' to use FSK modulation \*/

#define USE\_MODEM\_LORA 0

#define USE\_MODEM\_FSK !USE\_MODEM\_LORA

#define RF\_FREQUENCY 868300000 // Hz

#define TX\_OUTPUT\_POWER 14 // 14 dBm

#if USE\_MODEM\_LORA == 1

#define LORA\_BANDWIDTH 125000 // LoRa default, details in SX1276::BandwidthMap

#define LORA\_SPREADING\_FACTOR LORA\_SF7

#define LORA\_CODINGRATE LORA\_ERROR\_CODING\_RATE\_4\_5

#define LORA\_PREAMBLE\_LENGTH 8 // Same for Tx and Rx

#define LORA\_SYMBOL\_TIMEOUT 5 // Symbols

#define LORA\_FIX\_LENGTH\_PAYLOAD\_ON false

#define LORA\_FHSS\_ENABLED false

#define LORA\_NB\_SYMB\_HOP 4

#define LORA\_IQ\_INVERSION\_ON false

#define LORA\_CRC\_ENABLED true

#elif USE\_MODEM\_FSK == 1

#define FSK\_FDEV 25000 // Hz H=1

#define FSK\_DATARATE 50000 // bps

#define FSK\_BANDWIDTH 100000 // Hz TX\_BW

#define FSK\_AFC\_BANDWIDTH 103473 // Hz RX\_BW

#define FSK\_PREAMBLE\_LENGTH 5 // Same for Tx and Rx

#define FSK\_FIX\_LENGTH\_PAYLOAD\_ON false

#define FSK\_CRC\_ENABLED true

#else

#error "Please define a modem in the compiler options."

#endif

#define RX\_TIMEOUT\_VALUE 0 // in ms

#define MAX\_RX\_LENGTH 64

#define MAX\_HOP\_COUNT 3

#define LENGTH\_OFFSET 11

/\*

\* Global variables declarations

\*/

typedef enum

{

IDLE,

RX\_COMPLETE,

RX\_TIMEOUT,

RX\_ERROR,

TX\_START,

TX\_WAITING\_COMPLETE,

TX\_COMPLETE,

TX\_TIMEOUT,

} AppStates\_t;

volatile AppStates\_t State = IDLE;

/\*!

\* Radio events function pointer

\*/

static RadioEvents\_t RadioEvents;

/\*

\* Global variables declarations

\*/

SX1276Generic \*Radio;

typedef \_\_packed struct {

uint8\_t length;

uint8\_t hopCounter;

uint32\_t destinationAddress;

uint32\_t sourceAddress;

uint32\_t nonceCounter;

uint8\_t reserved1;

uint8\_t reserved2;

uint8\_t payload[16];

} myPacket\_t;

myPacket\_t myRxPacket, myTxPacket;

volatile uint8\_t lastRxLength;

AES myEncryptionAES;

uint8\_t mySecretEncryptionKey[] = "mySecretPassword";

uint32\_t buttonPressCounter;

InterruptIn mybutton(USER\_BUTTON);

void myButtonInterruptFunction(){

if(State == IDLE){

State = TX\_START;

buttonPressCounter++;

}

}

int SX1276PingPong()

{

uint8\_t i;

aesSetKey(&myEncryptionAES,mySecretEncryptionKey, 16, SMALL\_AES\_ENCRYPTION);

uint8\_t testText[] = "mySecretMessage!";

uint8\_t ciphertext[16];

aesEncrypt(&myEncryptionAES, testText, ciphertext);

for(i=0; i<16; i++){

dprintf("0x%02X ",ciphertext[i]);

}

uint32\_t \* addressUID = (uint32\_t \*)(0x1FF80050 + 0x14);

uint32\_t myUidValue = \*addressUID;

dprintf("myUidValue = %u", myUidValue);

srand(myUidValue);

dprintf("sizeof(myRxPacket) = %u", sizeof(myRxPacket));

dprintf("MY\_MAC\_ADDRESS = 0x%08X", MY\_MAC\_ADDRESS);

mybutton.fall(&myButtonInterruptFunction);

Radio = new SX1276Generic(NULL, MURATA\_SX1276,

LORA\_SPI\_MOSI, LORA\_SPI\_MISO, LORA\_SPI\_SCLK, LORA\_CS, LORA\_RESET,

LORA\_DIO0, LORA\_DIO1, LORA\_DIO2, LORA\_DIO3, LORA\_DIO4, LORA\_DIO5,

LORA\_ANT\_RX, LORA\_ANT\_TX, LORA\_ANT\_BOOST, LORA\_TCXO);

dprintf("SX1276 Ping Pong Demo Application" );

dprintf("Freqency: %.6f", (double)RF\_FREQUENCY/1000000.0);

dprintf("TXPower: %d dBm", TX\_OUTPUT\_POWER);

#if USE\_MODEM\_LORA == 1

dprintf("Bandwidth: %d Hz", LORA\_BANDWIDTH);

dprintf("Spreading factor: SF%d", LORA\_SPREADING\_FACTOR);

#elif USE\_MODEM\_FSK == 1

dprintf("Bandwidth: %d kHz", FSK\_BANDWIDTH);

dprintf("Baudrate: %d", FSK\_DATARATE);

#endif

// Initialize Radio driver

RadioEvents.TxDone = OnTxDone;

RadioEvents.RxDone = OnRxDone;

RadioEvents.RxError = OnRxError;

RadioEvents.TxTimeout = OnTxTimeout;

RadioEvents.RxTimeout = OnRxTimeout;

if (Radio->Init( &RadioEvents ) == false) {

while(1) {

dprintf("Radio could not be detected!");

wait( 1 );

}

}

Radio->SetChannel(RF\_FREQUENCY );

#if USE\_MODEM\_LORA == 1

if (LORA\_FHSS\_ENABLED)

dprintf(" > LORA FHSS Mode <");

if (!LORA\_FHSS\_ENABLED)

dprintf(" > LORA Mode <");

Radio->SetTxConfig( MODEM\_LORA, TX\_OUTPUT\_POWER, 0, LORA\_BANDWIDTH,

LORA\_SPREADING\_FACTOR, LORA\_CODINGRATE,

LORA\_PREAMBLE\_LENGTH, LORA\_FIX\_LENGTH\_PAYLOAD\_ON,

LORA\_CRC\_ENABLED, LORA\_FHSS\_ENABLED, LORA\_NB\_SYMB\_HOP,

LORA\_IQ\_INVERSION\_ON, 2000 );

Radio->SetRxConfig( MODEM\_LORA, LORA\_BANDWIDTH, LORA\_SPREADING\_FACTOR,

LORA\_CODINGRATE, 0, LORA\_PREAMBLE\_LENGTH,

LORA\_SYMBOL\_TIMEOUT, LORA\_FIX\_LENGTH\_PAYLOAD\_ON, 0,

LORA\_CRC\_ENABLED, LORA\_FHSS\_ENABLED, LORA\_NB\_SYMB\_HOP,

LORA\_IQ\_INVERSION\_ON, true );

#elif USE\_MODEM\_FSK == 1

dprintf(" > FSK Mode <");

Radio->SetTxConfig( MODEM\_FSK, TX\_OUTPUT\_POWER, FSK\_FDEV, 0,

FSK\_DATARATE, 0,

FSK\_PREAMBLE\_LENGTH, FSK\_FIX\_LENGTH\_PAYLOAD\_ON,

FSK\_CRC\_ENABLED, 0, 0, 0, 2000 );

Radio->SetRxConfig( MODEM\_FSK, FSK\_BANDWIDTH, FSK\_DATARATE,

0, FSK\_AFC\_BANDWIDTH, FSK\_PREAMBLE\_LENGTH,

0, FSK\_FIX\_LENGTH\_PAYLOAD\_ON, 0, FSK\_CRC\_ENABLED,

0, 0, false, true );

#else

#error "Please define a modem in the compiler options."

#endif

dprintf("Wireless Sensor Networks LAB.4");

Radio->Rx( RX\_TIMEOUT\_VALUE );

while( 1 )

{

#ifdef TARGET\_STM32L4

WatchDogUpdate();

#endif

switch( State )

{

case IDLE:

// Do nothing - wait for button interrupt

sleep();

break;

case RX\_COMPLETE:

if (myRxPacket.length == lastRxLength){

dprintf("Rx complete!");

dprintf("myRxPacket.length = %u",myRxPacket.length);

dprintf("myRxPacket.hopCounter = %u",myRxPacket.hopCounter);

dprintf("myRxPacket.sourceAddress = 0x%08X",myRxPacket.sourceAddress);

dprintf("myRxPacket.destinationAddress = 0x%08X",myRxPacket.destinationAddress);

if(myRxPacket.destinationAddress == MY\_MAC\_ADDRESS){

dprintf("This packet is for us!");

myRxPacket.hopCounter = 0;

uint8\_t plaintextBuffer[17];

uint8\_t ivCipherText[16];

aesEncrypt(&myEncryptionAES, (uint8\_t\*)&myRxPacket, ivCipherText);

for(i=0; i<16; i++){

plaintextBuffer[i]=myRxPacket.payload[i] ^ ivCipherText[i];

}

plaintextBuffer[16]=0; // Add Null at the end of text manually

dprintf("Payload is %s\n",plaintextBuffer);

}

else{

dprintf("Packet was not for us (different destinationAddress)");

if (myRxPacket.sourceAddress == MY\_MAC\_ADDRESS){

dprintf("This packet was sent for us, so we do not forward it again");

}

else{

if (myRxPacket.hopCounter >0){

dprintf("Hop counter > 0, forwarding packet");

myRxPacket.hopCounter--;

uint16\_t random\_wait\_ms = rand() % 1000;

dprintf("Waiting %d ms before forwarding", random\_wait\_ms);

wait(random\_wait\_ms \* 0.001);

Radio->Send( &myRxPacket, myRxPacket.length );

State = TX\_WAITING\_COMPLETE;

break;

}

}

}

}

else{

dprintf("Packet length error!");

}

Radio->Rx( RX\_TIMEOUT\_VALUE ); // Put transceiver back to Rx

State = IDLE;

break;

case RX\_TIMEOUT:

Radio->Rx( RX\_TIMEOUT\_VALUE ); // Put transceiver back to Rx

dprintf("Rx Timeout happened\n");

State = IDLE;

break;

case RX\_ERROR:

Radio->Rx( RX\_TIMEOUT\_VALUE ); // Put transceiver back to Rx

dprintf("Rx CRC Error happened\n");

State = IDLE;

break;

case TX\_START:

myTxPacket.hopCounter = 0;

myTxPacket.sourceAddress = MY\_MAC\_ADDRESS;

myTxPacket.destinationAddress = OTHER\_MAC\_ADDRESS;

myTxPacket.length = 32;

myTxPacket.nonceCounter++;

uint8\_t plaintextBuffer[16];

sprintf((char\*)plaintextBuffer, "CNT = %u", buttonPressCounter);

uint8\_t ivCipherText[16];

aesEncrypt(&myEncryptionAES, (uint8\_t\*)&myTxPacket, ivCipherText);

for(i=0; i<16; i++){

myTxPacket.payload[i]=plaintextBuffer[i] ^ ivCipherText[i];

}

myTxPacket.hopCounter = MAX\_HOP\_COUNT;

Radio->Sleep( ); // First we need to put chip from Rx to Sleep state

Radio->Send( &myTxPacket, myTxPacket.length );

dprintf("Message sent! buttonPressCounter was %d", buttonPressCounter);

State = TX\_WAITING\_COMPLETE;

break;

case TX\_WAITING\_COMPLETE:

sleep();

break;

case TX\_COMPLETE:

dprintf("Message sending complete!");

Radio->Rx( RX\_TIMEOUT\_VALUE ); // Put transceiver back to Rx

State = IDLE;

break;

case TX\_TIMEOUT:

dprintf("TX\_TIMEOUT happened!");

Radio->Rx( RX\_TIMEOUT\_VALUE ); // Put transceiver back to Rx

State = IDLE;

break;

default:

State = IDLE;

break;

}

}

}

void OnTxDone(void \*radio, void \*userThisPtr, void \*userData)

{

Radio->Sleep( );

State = TX\_COMPLETE;

if (DEBUG\_MESSAGE)

dprintf("> OnTxDone");

}

void OnRxDone(void \*radio, void \*userThisPtr, void \*userData, uint8\_t \*payload, uint16\_t size, int16\_t rssi, int8\_t snr)

{

Radio->Sleep( );

lastRxLength = size;

if (size > MAX\_RX\_LENGTH){

lastRxLength = 64;

}

memcpy( &myRxPacket, payload, size );

State = RX\_COMPLETE;

if (DEBUG\_MESSAGE)

dprintf("> OnRxDone: RssiValue=%d dBm, SnrValue=%d", rssi, snr);

dump("Data:", payload, size);

}

void OnTxTimeout(void \*radio, void \*userThisPtr, void \*userData)

{

Radio->Sleep( );

State = TX\_TIMEOUT;

if(DEBUG\_MESSAGE)

dprintf("> OnTxTimeout");

}

void OnRxTimeout(void \*radio, void \*userThisPtr, void \*userData)

{

Radio->Sleep( );

State = RX\_TIMEOUT;

if (DEBUG\_MESSAGE)

dprintf("> OnRxTimeout");

}

void OnRxError(void \*radio, void \*userThisPtr, void \*userData)

{

Radio->Sleep( );

State = RX\_ERROR;

if (DEBUG\_MESSAGE)

dprintf("> OnRxError");

}

#endif

Graphical user interface, text

Description automatically generated

1. att. Pārbaude ka small\_aes bibliotēkas šifrēšanas objekts strādā

Graphical user interface, text, application, chat or text message

Description automatically generated

2. att. Pirmā iekārta sūt šifrēto testa ziņu “1234567890ABCDE”

Graphical user interface, text, application, chat or text message

Description automatically generated

3. att. Otrā iekārta sūt šifrēto testa ziņu “1234567890ABCDE”, kuru arī saņem pēc pāradresācijas

Graphical user interface, text, application, chat or text message

Description automatically generated

4. att. Pirmā iekārta sūt šifrēto ziņu kas satur pogu nospiešanas skaitu

Graphical user interface, text, application, chat or text message

Description automatically generated

5. att. Otrā iekārta sūt šifrēto ziņu kas satur pogu nospiešanas skaitu

Iešifrēšanas un atšifrēšanas lakā lēcienu skaitītājam tika uzstādīta 0 vērtība, lai ziņa būtu iešifrēta un atšifrēta ar vienu un to pašu headeri. 2. un 3. att. var redzēt ka katru reizi šifrēta ziņa atšķiras pie nemainīga plainteksta. 4. un 5. att. var redzēt ka ziņa tiek veiksmīgi atšifrēta pat ja ziņu lēcienu skaitītajam vērtība ir mainījies.

**Secinājumi:**

Šajā darbā tika izveidots kods kas iekļauj datu šifrēšanas algoritmu. Ieviestais datu šifrēšanas algoritms ir “counter mode encryption/decryption”, kas iešifrē/dešifrē ziņu pa blokiem. Darbā laika tiek pārbaudīta algoritma darbspēja, kas paradīja ka ziņas tika veiksmīgi iešifrētas un dešifrētas.