University of Turku
Cyber Physical Systems 2015 course
Project report – Wireless door opener

Student name: Vu Nguyen (511436)

Introduction

The aim of the project is to implement a wireless door opener based on radio communication. It is considered by many people to be inconvenient when they have to leave their cars and open the main entrance door manually, especially during the winter months. They want to stay inside their car and need a device capable of opening the door for them. Nowadays such a device can be found easily on the market. It consists of a remote control and a base station which receives data from the remote control and controls the transceivers accordingly. The wireless communication is based on 433Mhz radio communication with an own designed protocol or with rolling code. The advantage of such a device is easy to design. However, it suffers from security issues as the messages sent over the radio communication which are encrypted using rolling code can be intercepted and hacked [1, 2, 3, 4]. As a replacement, we will use AES-128 block cipher to encrypt the messages sent over then radio communication since it has been shown that AES encryption still remains secure for AES key based AES security system [5,6]. The remote control part will be used by a user to open or close the door. The base station will be used to received the encrypted messages send from the remote control and controls the door/gate accordingly. This report is going to describe the development process, design, implementation, and final testing of the project. Besides, the report can be used to replicate the project in the future after reading the Design and Implementation sections.

Project management

The project was planned to be done in a group of two students. However, one team member did not contribute any work to the project. Therefore, the whole project was done by Vu Nguyen only. The project consists of two main parts: development of the remote control and development of the base station. Regarding the remote control, embedded software was implemented for the remote control and connections between hardware components were made. The embedded software for the remote control is responsible for reading the states of the buttons (lock, unlock, open, and close), generating encrypted messages using AES-128 block cipher, and sending the encrypted messages over the air to the base station via an nRF24L01+ radio transceiver. As for the base station, hardware connections were made, and the embedded software was implemented to control the base station. The base station takes care of receiving the encrypted messages from the remote control, decrypting these message, and controlling the servo motor based on the type of the message.

The project had been planned to be completed in 2 to three weeks. However, one team member did not contribute any work to the project. As a result, it lasts longer than planned and took 1 month to finish. On average, 10 hours was spent per week doing the project. The final testing was carried out to test the operation of the system after the implementation had completed. The project report was the last phase of the project. About 4 hours was spent on writing the report. Github is used to store the source code and the reports. It can be found at:
https://github.com/quangng/wireless-door-opener

Description of the project

The system in the project consists of essentially two parts: the remote control and the base station. The remote control includes 4 buttons, an Arduino Uno platform, and an nRF24L01+ radio transceiver . The base station consists of an Intel Galileo platform, a servo motor, and a radio transceiver nRF24L01+. Figure 1 gives an big picture of the system described.

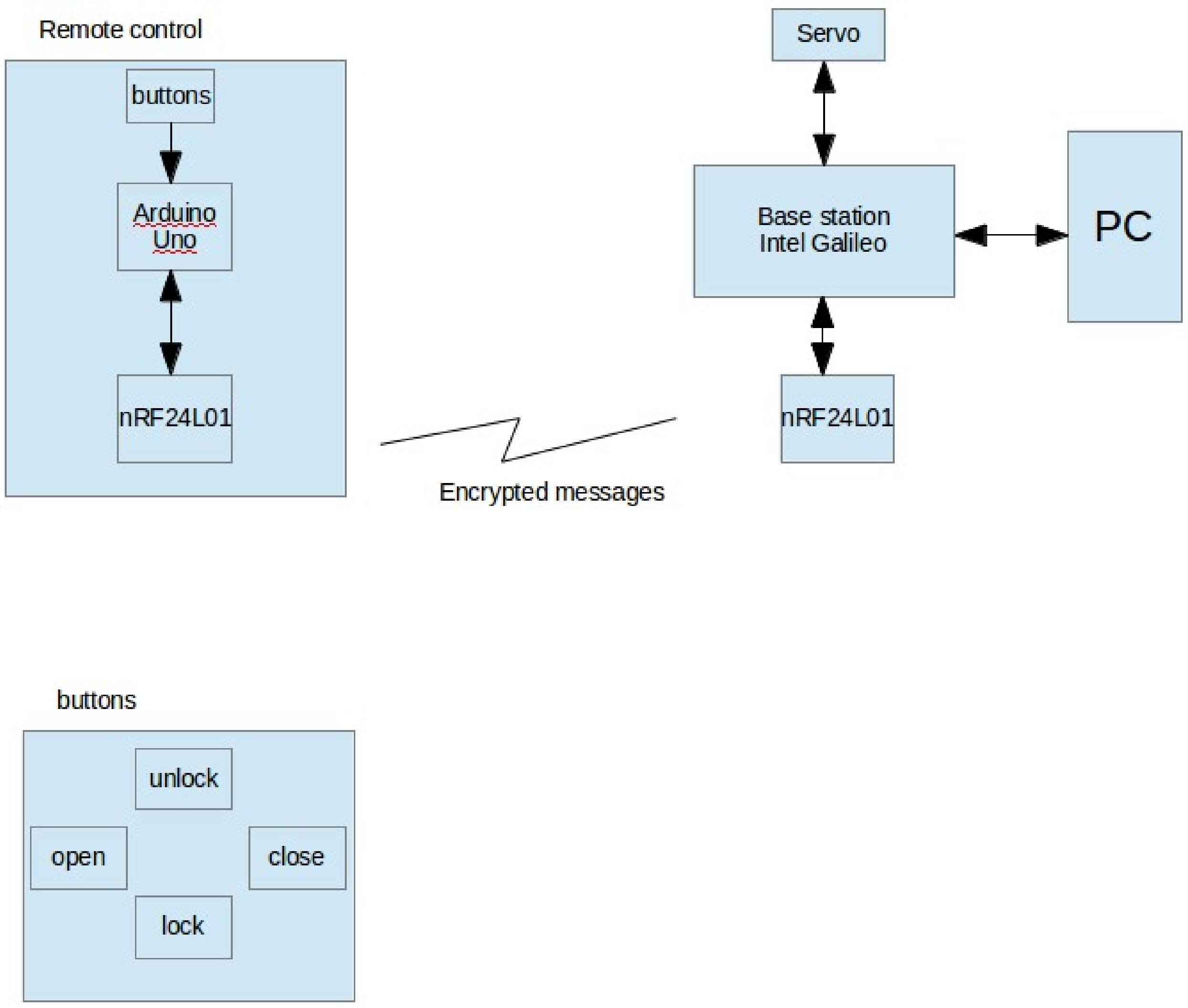

Figure 1. System block diagram of the Wireless door opener

In the remote control, there are 4 buttons: unlock, lock, open, and close. The lock button is used to disable the functionality of open and close buttons. This means that when the lock button is pressed, pressing open or close buttons will not send any message to the base station. By contrast, the unlock button is used for enabling the functionality of open and close buttons. It does exactly the opposite to the lock button. The open and close buttons, as their names imply, are used to open and close the door/gate, respectively. After the open or close button is pressed (with unlock enabled), Arduino Uno will generate an encrypted message using AES-128 block cipher and send it to the base station by means of the nRF24L01+ radio transceiver. The base station consists of a servo motor, an nRF24L01+ radio transceiver, and the base station itself which is an Intel Galileo platform. Upon receiving an encrypted message from the nRF24L01+ transceiver, Intel Galileo will decrypt the message and check the content of the decrypted message. If the message matches, the servo motor will be controlled open or close the door/gate.

Description of the equipment used

The equipment used in the project consists of hardware components for the remote control and the base station. Figure 2 shows all the components used in the project, excluding the USB cables

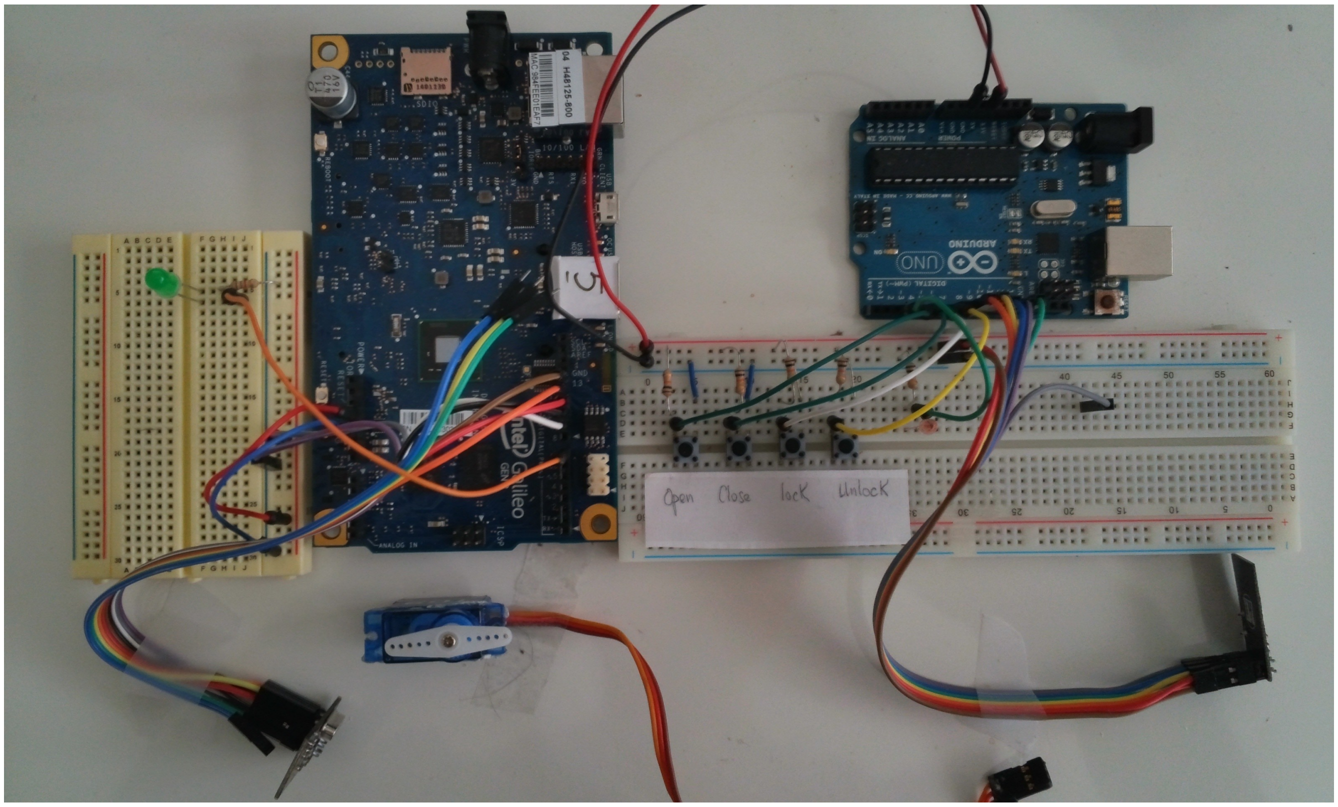

Figure 2. Hardware components used

Hardware components of the remote control

- Push button (x4)
- Arduino Uno (x1)
- nRF24L01+ radio transceiver (x1)
- Wires (male to female and male to male)
- Breadboard (x1)
- USB type B to USB type A cable (x1)
- LED (x1)
- 10K resistors (x5)

Hardware components for the base station

- Intel Galileo Gen 2 (x1)
- Breadboard (x1)
- Wires (male to female and male to male)
- nRF24L01+ radio transceiver (x1)
- Servo motor (x1)
- LED (x1)
- 10K resistor (x1)
- USB type micro B to USB type A cable (x1)
- Power supply (x1)
- SD card (x1)

Design of the project

Design of the remote control

Scheduler for embedded systems

According to the project description, the remote control has 4 buttons as an interface which a user uses to interact with the remote control. Therefore, checking the states of the buttons from the user interface should be processed as fast as possible. A solution to this is to use interrupt. Arduino Uno has two external interrupt pins: interrupt number 0 on digital pin 2 and interrupt number 1 on digital pin 3 [7]. However, there are 4 buttons which are needed to be connected to the external interrupt pins of Arduino Uno. As a result, using interrupt is not a possible option. As an alternative, we used a scheduler to schedule tasks can run in a defined interval. Another reason for choosing a scheduler is that the remote control is in practice battery powered. Hence, having a scheduler will help save power consumption since it can put the processor into sleep mode when the is no task running. A hybrid scheduler for embedded systems was chosen. The hybrid scheduler supports both preemption and non-preemption. It is already implemented and can be found from the following link.

https://github.com/blanboom/Arduino-Task-Scheduler/tree/master/TaskScheduler

The remote control has five tasks: 4 tasks for processing the states of the buttons and 1 tasks for sending the message to the remote control. The tasks are: openButtonUpdate, closeButtonUpdate, lockButtonUpdate, unlockButtonUpdate, and sendMessage. It is determine that task sendMessage tasks the longest time to execute compared to the rest of the tasks. This task is given the lowest priority. The other 4 tasks take very short time to execute. They are therefore given the highest priority. The scheduling of the tasks are given in Figure 3. T1, T2, T3, T4 represents openButtonUpdate, closeButtonUpdate, lockButtonUpdate, and unlockButtonUpdate tasks, respectively. T5 represents task sendMessage. The execution time of the is not scaled in the scheduling graph. In the real implementation, the execution time of the tasks should be a lot less than that. D1, D2, D3, D4, and D5 represent the deadlines of tasks 1, 2, 3, 4, and 5, respectively. Each task will be repeated every 100ms. The task parameters are described as follows:

Task(phi, p, D)
phi is the release time

p is the period
D is the deadline

openButtonUpdate task: T1(0, 100, 100)
closeButtonUpdate task: T2(20, 100, 100)
lockButtonUpdate task: T3(40, 100, 100)
unlockButonUpdate task: T4(60, 100, 100)
sendMessage task: T5(80, 100, 100)

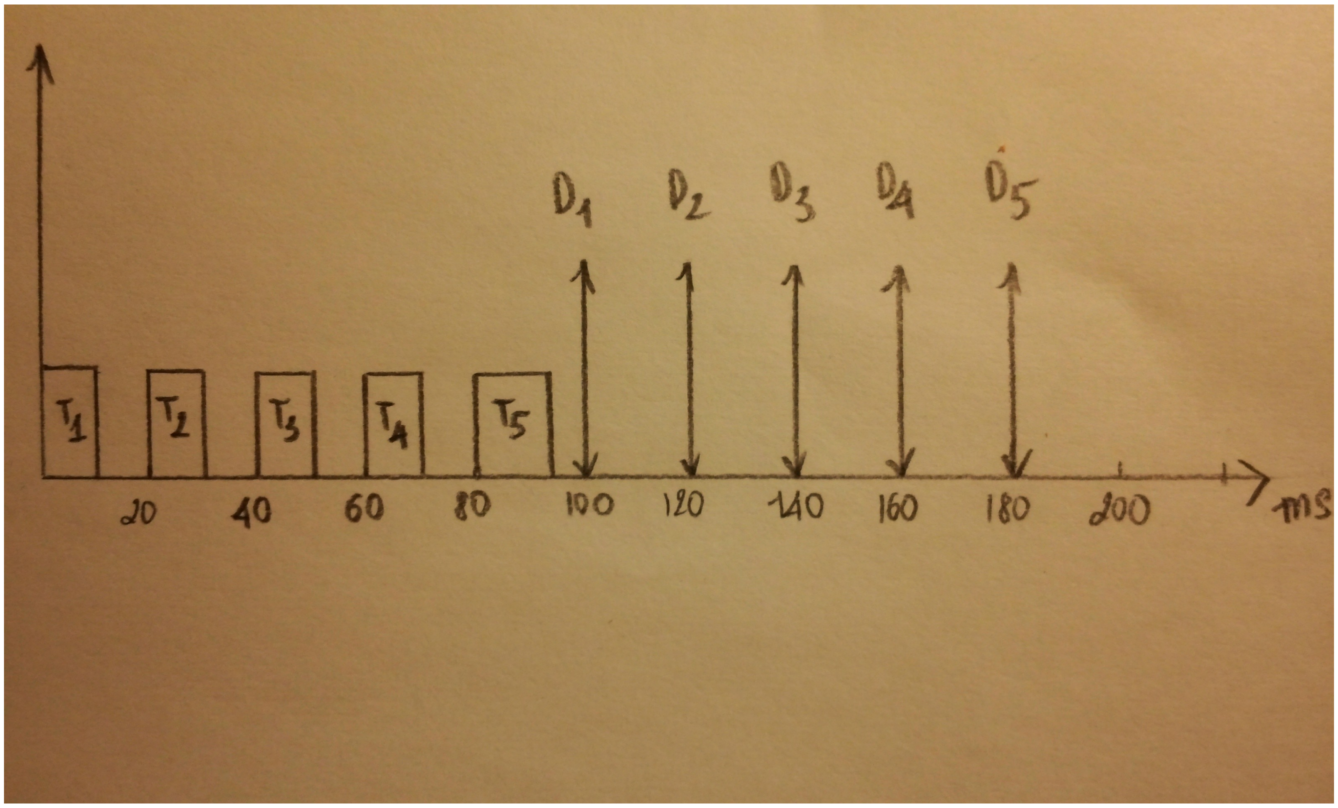

Figure 3. Task scheduling graph for the remote control

AES-128 block cipher

The AES-128 block cipher was chosen for the project for message encryption and description since it is secure as mentioned in the Introduction section. An implementation of the block cipher in the C language is given in Texas Instruments website at the following link

http://www.ti.com/tool/aes-128

This library is independent of the hardware architecture, so it was ported to use with Arduino Uno with very little modification. The original implementation of the library uses look-up tables and stores a large amount of data in volatile memory while Arduino Uno has only 2KB of SRAM [8]. This would cause memory overflow during run time. In order to use the AES-128 block cipher library with Arduino Uno, the solution to the problem was to store the look-up tables in Flash memory and then retrieve the data back to SRAM memory whenever encryption routine is called from the main function. Since Arduino Uno is based on AVR ATmege328, the AVR Libc run time library was used to implement the solution. Storing data in Flash memory is done by using the PROGMEM keyword which is a variable modifier, and it is used with datatypes defined in pgmspace.h header file. This keyboard instructs the compiler to store data into flash memory instead of SRAM where it would normally go [9]. During run time, data is written back to SRAM for calculation is doing by using pgm_read_byte macro which is provided by the AVR Libc run time library.

Hardware connections

The connection between the nRF24L01+ radio transceiver and Arduino Uno can be seen in Table 1. Figure 4 shows a real picture of nRF24L01+ radio transceiver with pin numbers and pin names.

Table 1. Connection between nRF24L01+ radio transceiver and Arduino Uno

nRF24L01+ pin number	nRF24L01+ pin name	Arduino Uno pin number	Arduino Uno pin name
1	GND		GND
2	VCC		3V3
3	CE	9	CE
4	CS	10	CSN
5	SCK	13	SCK
6	MOSI	11	MOSI
7	MISO	12	MISO

All the buttons were designed to be in pull up mode. The “open” button is connected to Arduino Uno pin 4, “close” button to Arduino Uno pin 5, “lock” button to Arduino Uno pin 6, and “unlock” button to Arduino Uno pin 7. An LED which indicates the open/close status is connected to Arduino Uno pin 3.

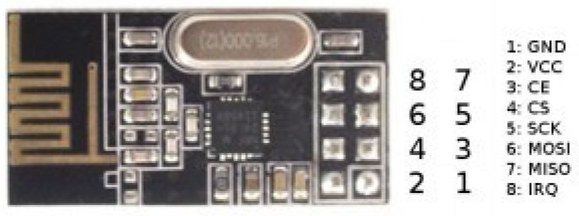

Figure 4. nRF24L01+ radio transceiver pin outs.
Reprinted from Chantrell 2013 [11]

Design of the base station

Software architectures

Since the base station is usually powered from a permanent power supply. An embedded operating system or scheduler is not needed for power saving purposes. Round-robin software architecture was used to simplify the implementation of the base station. With this software architecture, tasks or functions will be run sequentially, and the process is repeated over and over again. In particular, it checks if data is available from the radio transceiver connected to Intel Galileo Gen 2. If the message is received, it decrypts the message and compares the encrypted message with a hard-coded message. If the encrypted message matches. It will call a function to control the servo motor to open or close the the gate/door depending the type of the message. Also, the LED will be turned on to indicate the “open” message is received and turned off when the “close” message is received. When the base station is first powered up, the servo is controlled to close the door/gate by default.

AES-128 block cipher

Intel Galileo Gen 2 is a powerful 32-bit single processor embedded development board. It has 512Kb of on-die SRAM memory [12]. There is a large amount of memory available for application development on the board. Therefore, the block cipher used for the base station was kept, for the most part, the same as the one provided by Texas Instruments.

Hardware connections

The connection between the nRF24L01+ radio transceiver and Arduino Uno can be seen in Table 2. Figure 4 shows a real picture of nRF24L01+ radio transceiver with pin numbers and pin names.

Table 2. Connection between nRF24L01+ radio transceiver and Intel Galileo Gen 2

nRF24L01+ pin number	nRF24L01+ pin name	Intel Galileo Gen 2 pin number	Intel Galileo Gen 2 pin name
1	GND		GND
2	VCC		3V3
3	CE	9	CE
4	CS	10	CSN
5	SCK	13	SCK
6	MOSI	11	MOSI
7	MISO	12	MISO

Pin 5 of Intel Galileo Gen 2 is connected to the signal pin of the servo motor (via the yellow wire). Pin 7 is connected to the green LED which indicates the status of the door/gate.

Common between the remote control and the base station

The secret key used for encrypting and decrypting messages between the remote control and the base station is a key of 16 bytes size. This key is common to both the remote control and the base station. It is given in hexadecimal as follows:

```
uint8_t key[] = { 0x54, 0x68, 0x69, 0x73, 0x69, 0x73, 0x61, 0x73, 0x65, 0x63, 0x72, 0x65, 0x74,
                  0x6b, 0x65, 0x79};
```

Since the key takes only 16 bytes of SRAM. It was decided that this array stores in SRAM instead of saving in Flash memory.

Another common information is the communication address. This address is an array of 5 bytes in length. It is defined as follows:

```
uint8_t address[5] = {0xE7,0xE7,0xE7,0xE7,0xE7};
```

A simple library for nRF24L01+ radio transceiver is implemented by Kehribar from his Github repository was used in this project to save the development time. It is claimed nrf24L01_plus library is the most portable and basic library. The library can be found at:

https://github.com/kehribar/nrf24L01_plus

This library enables the auto acknowledgement and auto retransmission features of the nrf24L01+ in static length payload mode. These two features are basically the most important features of the nRF24L01_plus library. With this library, after making a transmission attempt, it is guaranteed that the slave device will receive the message properly with minimal MCU involvement. Besides, the nRF24L01_plus library automatically handle the retransmission of the same message if it gets lost in transmission, up to o limited trials with adjustable delays in between attempts [10]. This library is written in C for a different architecture. Some modifications to the library was done in the implementation phase to port it to Arduino Uno and Intel Galileo Gen 2.

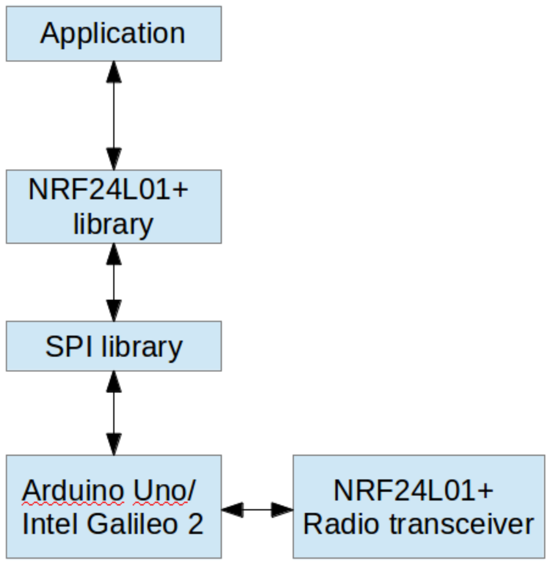

Figure 5. Abstraction layer for nRF24L01+ library

Since Arduino Uno and Intel Galileo Gen 2 offer the same SPI interface regardless of the underlying hardware architecture, this SPI library was used by the nRF24L01_plus library to communicate with nRF23L01+ radio transceiver via the development platform. Thanks to this common SPI interface, the nRF24L01library only needs to be implemented once and can be ported to both platforms.

Implementation of the project

Testing and debugging were included in the implementation phase. If something goes wrong and cannot be fixed in this phase, some modifications to the design will be made.

The remote control

AES-128 block cipher

The implementation of the AES-128 block cipher was started first since this library is independent of the underlying hardware architecture. The library is originally written in the C language while Arduino Uno IDE compiles C++ code. Therefore, in the header file of the AES-128 block cipher library, an extern keyword was inserted so that the compiler will be able to link these source code. The library consists of two files: cps_aes128.h and cps_aes128.h. In the cps_aes128.h, the extern keyword was included as follows:

```
#ifdef __cplusplus
extern "C"{
#endif

void aes_encrypt(uint8_t *state, uint8_t *key);
void aes_decrypt(uint8_t *state, uint8_t *key);

#ifdef __cplusplus
}
#endif
```

Listing 1. Code snippet of cps_aes128.h header file

To save SRAM memory space, the macro PROGMEM provided in AVR LibC <avr/pgmspace.h> was used to store data in Flash memory. The look-up tables of the block cipher library were stored in Flash memory of Atmel ATmege328. Writing data back to SRAM memory from Flash memory was done by using the macro pgm_read_byte which is also provided in <avr/pgmspace.h>. Listing 2 shows how this was done.

```
#include "cps_aes128.h"
#include <avr/pgmspace.h>

/* store 256 bytes of sbox array in Flash */
const uint8_t sbox[256] PROGMEM = {0x63, 0x7c, ...};

/* store 256 bytes of rsbox array in Flash */
const uint8_t rsbox[256] PROGMEM = {0x52, 0x09, …};

/* store 11 bytes of Rcon array in Flash */
const unsigned char Rcon[11] PROGMEM = {0x8d, 0x01,...};

…
…
```

```
…
/* read data from sbox array in Flash and write back to SRAM  */
expandedKey[ii*16 + 1] = pgm_read_byte(&sbox[expandedKey[ii*16 -
2]])^expandedKey[(ii-1)*16 + 1];
…

/* read data from rsbox array in Flash and write back to SRAM */
state[0]= pgm_read_byte(&rsbox[state[ 0]]) ^ expandedKey[(round*16)];
```

Lising 2. Store data to Flash memory and write data back to SRAM memory

One problem was encountered in the implementation of the block cipher library. This problem happened in the decryption function when data is read from Flash memory and written back to SRAM. The problem was fixed by adding a delay of 1 microsecond on the line 66 of the cps_aes128.c source file.

The main application

The main application remote-control.ino was implemented using the hybrid scheduler. It adds five tasks to the scheduler and calls the dispatchTasks function infinitely to dispatch the tasks whenever the tasks are due to run. When there is no task running, the processor will be put into sleep mode to save energy consumption. This is taken care of by the TaskScheduler library. Five tasks defined in the main application are openButtonUpdate, closeButtonUpdate, lockButtonUpdate, unlockButtonUpdate, and sendMessage. Adding the tasks to the scheduler and dispatching the tasks can be seen in Listing 3.

```
#include <TaskScheduler.h>

void setup() {
    /* Initialize the scheduler */
    Sch.init();

    /* Add tasks to the scheduler */
    Sch.addTask(openButtonUpdate, 0, 100, 1);
    Sch.addTask(closeButtonUpdate, 20, 100, 1);
    Sch.addTask(lockButtonUpdate, 40, 100, 1);
    Sch.addTask(unlockButtonUpdate, 60, 100, 1);
    Sch.addTask(sendMessage, 80, 100, 0);

    /* Start the scheduler */
    Sch.start();
}

void loop() {
    /* Dispatch the tasks */
    Sch.dispatchTasks();
}
```

Listing 3. Adding tasks to the hybrid scheduler

The sendMessage task regularly checks the status of the messageFlag flag. This flag is used to indicate if the open/close button is pressed. Once the open/close button is pressed, this task generates a message, encrypts it, and send it to the base station via the nRF24L01+ radio transceiver. Implementation of the sendMessage task is shown in Lising 4.

```
void sendMessage(void) {
  if(messageFlag) {
      if(openCloseState == BUTTON_STATE_OPEN) {
        uint8_t message[] = "0000PES2015_open";
        aes_encrypt(message, key);
        nrf24_send(message);
        while(nrf24_isSending());
      } else {
        uint8_t message[] = "000PES2015_close";
        aes_encrypt(message, key);
        nrf24_send(message);
        while(nrf24_isSending());
      }
    }
    messageFlag = false;
}
```

Listing 4. Implementation of sendMessage task

Detailed implementation of the other 4 tasks is pretty straightforward and can be referenced at

https://github.com/quangng/wireless-door-opener/blob/master/remote-control/remote-control/remote-control.ino

The base station

The base station was implemented by using round-robin software architecture. It infinitely checks if an message has been received by the nRF24L01+ radio transceiver. If the message is received, it then decrypts the message and compares if the encrypted message matches the hard-coded string. If the message is “0000PES2015_open”, it will control the servo motor to rotate 180 degrees to open the door/gate and turn the green LED on to indicate that the door/gate is opened. If the message is “000PES2015_close”, it will control the servo motor to rotate to the default position which is 0 degree and turn the green LED off to indicate that the door/gate is closed. The following code snippet shows the infinite loop that the base station is based on.

```
void loop() {
  if(nrf24_dataReady()){ //Check if data has been received
    nrf24_getData(message);
    aes_decrypt(message, key);
    char tmp[17];

    //convert message from uint8_t to char
    for (int i = 0; i < sizeof(message); i++) {
      char c = message[i];
      tmp[i] = c;
    }

    if (strcmp(tmp, "0000PES2015_open") == 0) {
      digitalWrite(LED_PIN, HIGH);
      myservo.write(180);
    } else if (strcmp(tmp, "000PES2015_close") == 0) {
      digitalWrite(LED_PIN, LOW);
      myservo.write(0);
    }
    delay(100); //Wait for servo to get to the position set
  }
}
```

Listing 5. Code snippet for the base station

Detailed implementation of the base station can be seen at the following link

https://github.com/quangng/wireless-door-opener/blob/master/base-station/base-station/base-station.ino

The block cipher AES-128 library used for the base station is the TI_AES library which is provided by Texas Instruments. One small modification was made in the header file in order for Arduino IDE to be to compile the source code which was originally written in C. The nRF24L01+ library is going to be discussed the the following section. This library is used by both Arduino Uno and Intel Galileo Gen 2 platforms.

Implementation of nRF24L01+ library

nRF24L01_plus is a portable library written in C by Kehribar. This library was modified to port it to Arduino Uno and Intel Galileo Gen 2 platforms. The nrf24.h header file of the library was modified as follows:

```
#if ARDUINO < 100
#include <WProgram.h>
#else
#include <Arduino.h>
#endif

#include "nRF24L01.h"
#include <SPI.h>

void nrf24_init(uint8_t ce_pin, uint8_t cs_pin);
uint8_t spi_transfer(uint8_t tx);
void nrf24_ce_digitalWrite(uint8_t state);
void nrf24_csn_digitalWrite(uint8_t state);
```

Listing 6. Code snippet of nrf24.h header file

Arduino Uno and Intel Galileo Gen 2 both support SPI library with the same API; <SPI.h> is included in the header to support SPI protocol communication with the nRF24L01+ radio transceiver. The four functions mentioned in Listing 6 is architecture dependent. Therefore, they were modified in the nrf24.cpp source file. Originally this library was written in C. However, the SPI library provided by Arduino IDE is written in C++. As a result, the nrf24.c file was changed to nrf24.cpp to make things work. Implementation of the hardware dependent functions are given in Listing 7 below.

```
/* init the hardware pins */
void nrf24_init(uint8_t ce_pin, uint8_t cs_pin) {
  _cepin = ce_pin;
  _cspin = cs_pin;
  pinMode(ce_pin, OUTPUT);
  digitalWrite(ce_pin, LOW);
  pinMode(cs_pin, OUTPUT);
  digitalWrite(ce_pin, HIGH);
  SPI.begin();
  nrf24_ce_digitalWrite(LOW);
  nrf24_csn_digitalWrite(HIGH);
}

/* low level interface ... */
void nrf24_ce_digitalWrite(uint8_t state) {
  digitalWrite(_cepin, state);
}

void nrf24_csn_digitalWrite(uint8_t state) {
  digitalWrite(_cspin, state);
}

uint8_t spi_transfer(uint8_t tx) {
  uint8_t rx = 0;

  rx = SPI.transfer(tx);
  return rx;
}
```

Listing 7. Code snippet for implementation of hardware dependent functions

Function nrf24_init takes chip enable (ce) pin and chip select (cs) pins as inputs. It then sets ce pin an output pin with low logic level and cs pin an output pin with high logic level to initiate SPI communication. The nrf24_ce_digitalWrite and nrf24_csn_digitalWrite calls the lower level functions which are architecture specific functions to set the states of ce and cs pins, respectively. Function spi_transfer takes care of sending data to the slave device. This is done by calling SPI functions provided by Arduino IDE development environment.

Results and conclusion

The goal of the project was to implement a wireless door opener based on radio communication. The wireless door opener was successfully tested. The distance tested between the remote control and the base station could reach as far as 20m with obstacles around. Also, thanks to the hybrid scheduler and the scheduling design, reading the button pins were processed pretty fast, giving the user a “real time” feeling when interacting with the remote control. One main drawback of the project was the block cipher mode of operation. It uses electronic codebook (ECB) mode; identical plaintext blocks are encrypted into identical cipher text blocks. Thus, it does not hide data patterns well when the encrypted messages are sent to the base station. This suggests further improvement in the future. Demonstration of the project can be seen at the following Youtube link.

https://www.youtube.com/watch?v=gQUoRQ4VVks

References

[1] Andy Greenberg. WATCH THIS WIRELESS HACK POP A CAR'S LOCKS IN MINUTES [online]; WIRED 08 April 2014.

http://www.wired.com/2014/08/wireless-car-hack/

Accessed 16 March 2015.

[2] Matthew Sparkes. Hackers can unlock your can with little more than a laptop [online]; The Telegraph 05 August 2014.

http://www.telegraph.co.uk/technology/news/11013062/Hackers-can-unlock-your-car-with-little-more-than-a-laptop.html

Accessed 16 March 2015.

[3] Erica Naone. Car Theft by Antenna [online]; MIT Technology Review 06 January 2011.

http://www.technologyreview.com/news/422298/car-theft-by-antenna/page/1/

Accessed 16 March 2015.

[4] Jam intercept and replay attack against rolling code key fob entry systems using RTL-SDR [online]; 15 March 2014

http://spencerwhyte.blogspot.ca/2014/03/delay-attack-jam-intercept-and-replay.html

Accessed 16 March 2015.

[5] Dave Neal. AES encryption is cracked [online]; theinquirer.net 17 August 2011

www.theinquirer.net/inquirer/news/2102435/aes-encryption-cracked

Accessed 16 March 2015.

[6] Alan Kaminsky et al. An Overview of Cryptanalysis Research for the Advanced Encryption Standard; Military Communications Conference 31 October 2010.

[7] Official Arduino website.

http://arduino.cc/en/Reference/attachInterrupt

Accessed 10 April 2015

[8] Offical Arduino website.

http://arduino.cc/en/main/arduinoBoardUno

Accessed 10 April 2015

[9] Offical Arduino website

http://arduino.cc/en/Reference/PROGMEM

Accessed 10 April 2015

[10] Kehribar. nrf24L01_plus library[online].

https://github.com/kehribar/nrf24L01_plus/blob/master/README.md

[11] Nathan Chantrell. Experimenting with the nRF24L01+ 2.4GHz radios [online]; Nathan's website 10 August 2013.

http://nathan.chantrell.net/20130810/experimenting-with-the-nrf24l01-2-4ghz-radios/

Accessed 10 April 2015

[12] Intel Corporation. Intel Galileo Gen 2 Development Board [online]. Intel Corporation 2014.

http://download.intel.com/support/galileo/sb/intelgalileogen2prodbrief_330736_003.pdf

Accessed 10 April 2015