Wrocław University of Science and Technology

Faculty of Information and Communication Technology

List 1 and List 2 Report

Hakan Yildizhan 270056
Gabriel Wechta 250111

Supervisor:
Wojciech Wodo, PhD

November 13, 2022

Report
Embedded Security

Contents

0 | Introduction

List 1 and List 2 topics focus on infrared technology namely sending and receiving NEC frames, A5/1 stream cipher, and simple usages of Arduino board.

This report's purpose is to present how the solutions for List 1 and List 2 were developed. It will have less formal construction. We will focus more on how we developed a solution to a given problem, rather than describe standards and technologies. We will also describe workstation and provide images of our breadboards and devices.

1 | List 1

1.1 Task formulation

"IrDA and oscilloscope showcase. *Saleae* download page for logic analyzer software. We used infrared receiver *TSOP31236* datasheet and some code for *Arduino*. Traces of IrDA communication from the remote as picked up by the receiver will be send to respective students via email for decoding.

Make sure you can identify protocol used for data transmission and you are able to decode the data, as well as, create your own valid frame and transmit it. The ultimate goal is to perform replay attack - be able to transmit exactly the same data as grabbed in the traces."

1.2 Traces

We used Saleae Logic Analyzer ver.1.2.40 for examining two received traces. Those traces contained logic data for two sample IrDA frames. Following images show output from the Saleae Logic Analyzer.

Figure 1.1: Sample 1

Figure 1.2: Sample 2

Things to notice:

- Samples are different but are pretty similar.
- High state is a base state.
- At the beginning there is 9ms leading pulse burst.
- Then a 4.5ms space.
- After that there are two lengths of high impulses, short – 0.5ms, long – 1.6ms, and one length of low impulse – 0.62ms.

Additionally, after main frames there are two or more much shorter frames (see 1.3). After examination we found out that this is NEC Infrared Transmission Protocol [1].

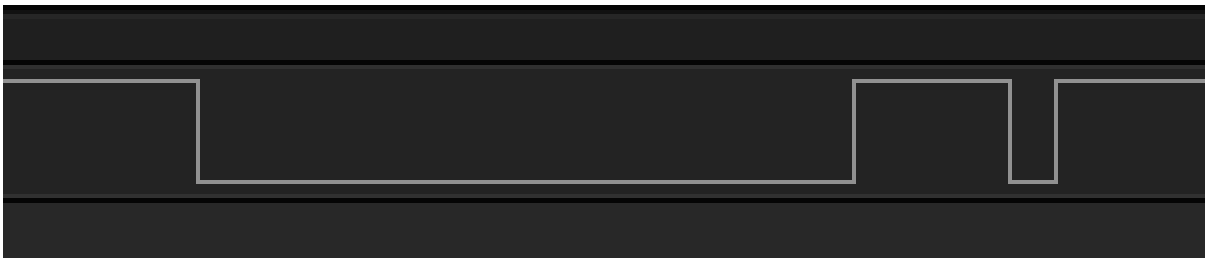

Figure 1.3: Repeat code

1.3 NEC

The NEC IR transmission protocol uses pulses to encode the message bits. Each pulse burst is 562.5µs in length, at a carrier frequency of 38kHz (26.3µs). Logical bits are transmitted as follows:

- **0** – a 562.5µs pulse burst then 562.5µs space, with a total transmit time of 1.125ms
- **1** – a 562.5µs pulse burst then 1.6875ms space, with a total transmit time of 2.25ms

The message contains 32 bits.

- 9ms – pulse leading begin of message transmission
- 4.5ms – space
- **8-bit** – address
- **8-bit** – inverse of the address
- **8-bit** – command
- **8-bit** – inverse of the command
- 562.5µs pulse – end of message transmission

Although it takes different time to encode **0** and **1**, because of the fact that each frame uses the same number of **0** and **1** (due to logical inversion of bits in address and command), NEC frame has constant duration time which is equal to 67.5ms.

When the key on the remote controller is still being pressed, repeat code will be sent. Repeat code consists of the following.

- 9ms – leading pulse burst
- 2.25ms – space
- 562.5µs – pulse to end repeat code frame

1.4 Decoding samples

We used NEC specification to decode received samples. We simply analyzed it by hand, bit by bit.

- `sample1.logicdata` – `01001100 10110011 10000001 01111110` (`0x4CB3817E`)
- `sample2.logicdata` – `01001100 10110011 01100001 10011110` (`0x4CB3619E`)

Clearly, address in both messages is the same, command differ by more than just one bit and the rule of inversion is preserved.

1.5 Circuit description

We are going to describe only the circuit elements that are fundamental to the presented repeat attack. We use 5V power supply.

- TSOP31236 – IR receiver. According to the datasheet can be connected to 5V circuit without any additional resistor.
- IR LED – Unknown model number, but luckily we were able to calculate required resistor. Forward voltage of typical IR LED is 0.8–1.6 [V]. We assumed that save forward current is 20mA and picked forward voltage in the middle of expected range so equal to 1.2V. We used Kirchoff's Second Law and Ohm's Law to derive equation 1 for needed resistance. Where U_z – circuit voltage, U_f – forward voltage, I_f – forward current. In our case this equation yields 180Ω.
- 220Ω resistor – closest value we have to the one we should have.
- Arduino Uno – programmable single-board microcontroller.
- Saleae Logic State Analyzer 8CH 24MH – for analyzing produced frames.

$$R = \frac{U_z - U_f}{I_f} \tag{1}$$

The main idea is to use IR transmitter (IR LED) to reproduce analyzed NEC messages. For the sake of proving that transmitted frames are actually what we intent to send, we capture them with IR receiver (TSOP31236) connected to Saleae Logic Analyzer. On Arduino we run code reproducing those two NEC messages (see code below).

For the photo of described circuit see image 1.4.

1.6 Results

Messages caught by IR receiver and nicely visualized by Saleae Logic Analyzer software can be seen in images 1.5 and 1.6. Maybe it is not clearly visible at the first glance but obtained frames and those produced by us are very similar.

```
#include <IRremote.h>  // library for sending NEC frames

IRsend irsend;  // create sending object
unsigned int buf[3];
unsigned long sample1_hex = 0x4CB3817E;  // Hex value of the first sample
unsigned long sample2_hex = 0x4CB3619E;  // Hex value of the second sample

void setup() {
    Serial.begin(9600);
    buf[0] = 9000;  // Mark 9ms
    buf[1] = 2250;  // Space 2.25ms
    buf[2] = 560;   // Burst
}

void replay_sample(unsigned long sample_hex){
    irsend.sendNEC(sample_hex, 32);  // send main frame
    delay(40);
    irsend.sendRaw(buf, 3, 38);  // send first repeat code
    delay(96);
    irsend.sendRaw(buf, 3, 38);  // send second repeat code
}

void loop() {
    delay(1000);
    replay_sample(sample1_hex);

    delay(1000);
    replay_sample(sample2_hex);
}
```

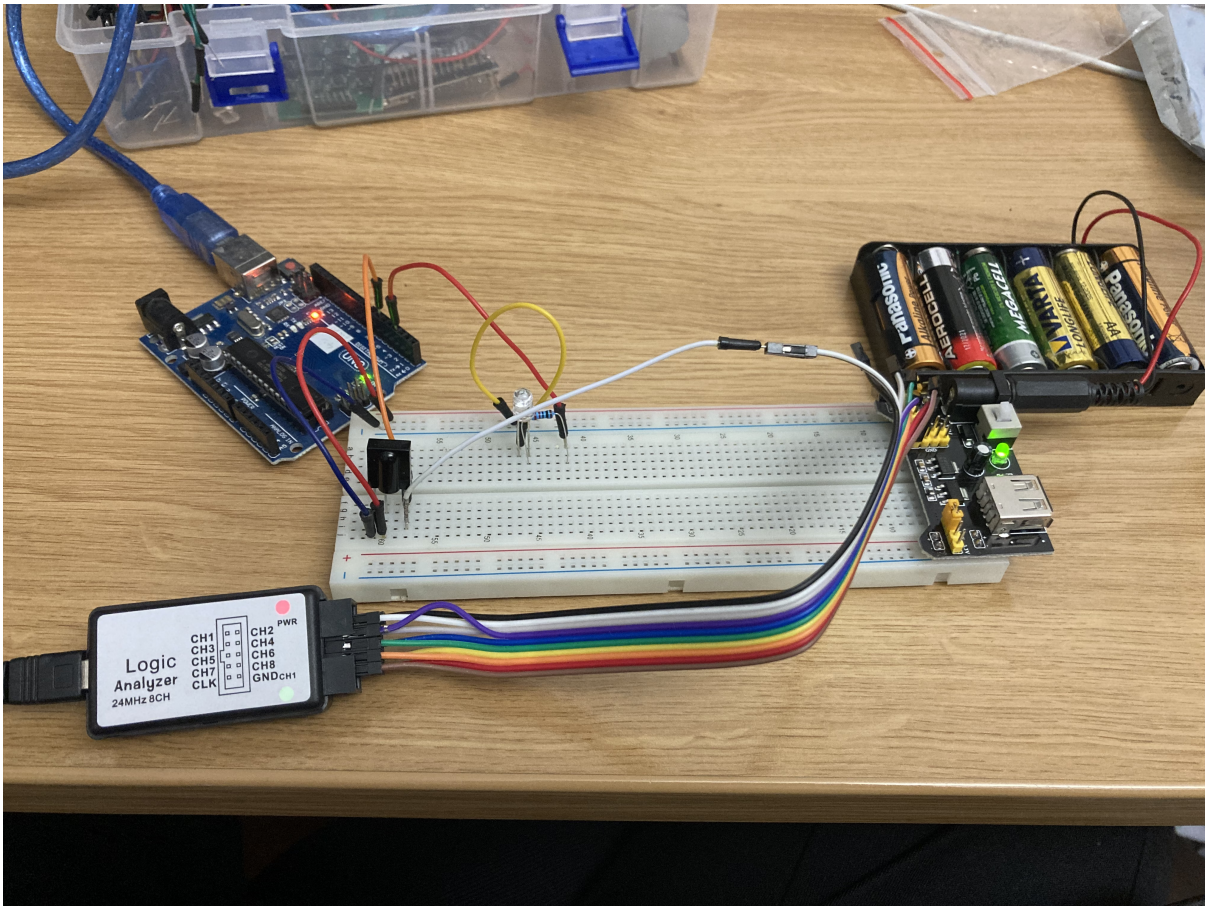

Figure 1.4: Circuit for replay attack

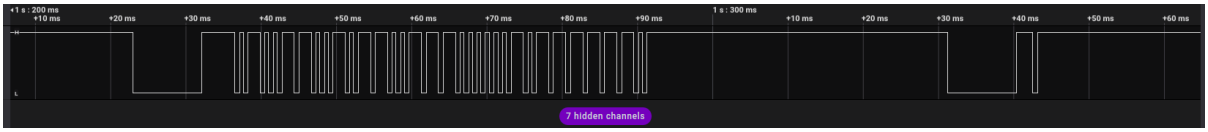

Figure 1.5: Replay sample 1

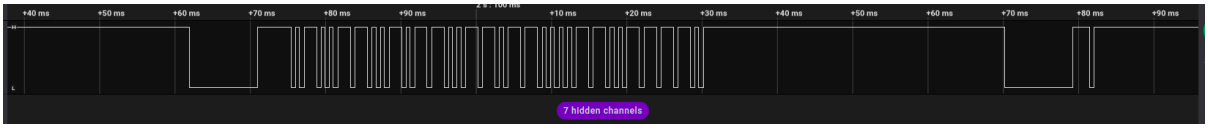

Figure 1.6: Replay sample 2

2 | List 2

2.1 Task formulation

"Securing IrDA transmission with A5/1. Using Arduino platform and the IrDA hardware from the showcase build a transmission link protected with A5/1 implementation of stream cipher. There are two cases to implement:

- The two communicating PCs separately produce the keychains from A5/1 and pass it to Arduino boards using UART. The encrypted message is sent between the Arduinos over IrDA link (this is case for people who have got two pieces of Arduino).
- Arduino board is connected via UART link with PC for keystream transmission. The keystream is received from the sending PC and forwarded in parallel with the cryptogram via IrDA link in a broadcast mode. In this case logical signal analyzer is helpful to capture the data on the transmitter side You can compare plain text and encrypted signal."

2.2 A5/1 Cipher

A5/1 is a stream cipher for mainly GSM communication, which was developed in 1987. The algorithm of this cipher was discovered via reverse engineering in 1999. This stream cipher follows the following algorithm:

- A5/1 cipher is initialized with a 64-bit (randomly selected) sequence and a public 22-bit nonce.
- Three bit registers or "LFSR"s of lengths 19, 22 and 23 are initialized. See image 2.1 for the structure of these registers.
- For each LFSR, there are special bits - bit 8, bit 10 and bit 10 respectively, along with "XOR bits" - for LFSR 1 - bits 13, 16, 17, 18; for LFSR 2 - bits 20 and 21; and for LFSR 3 - bits 20, 21 and 22.
- For each LFSR, starting from bit 0, XOR each bit with bit i of the 64-bit key, and place the resulting bit at the beginning of the LFSR, shifting all remaining bits to the right. Follow the same procedure with the 22-bit nonce.
- Find the "majority bit" across all three LFSRs; the bit that appears the most among the special bits of the LFSRs.
- For 100 times, for each LFSR, if its special bit equals to the majority bit, XOR the XOR bits, placing the result at the beginning and shifting all remaining bits to the

right. If the special bit does not equal to the majority bit, skip.

- Follow the same procedure for an additional 228 times, this time, XOR'ing the last bits of all three LFSRs and adding the resulting bit to a 228-bit keystream array.
- The resulting keystream can be used to encrypt the message by way of bitwise XOR operations.

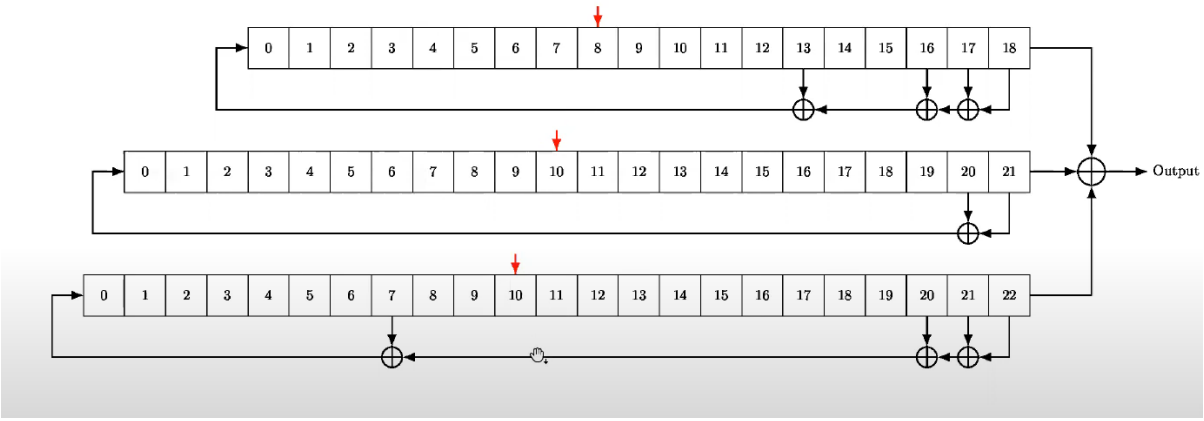

Figure 2.1: A5/1 Cipher Bit-Registers

2.3 Circuit description

Following items were used in the circuit:

- TSOP31236 IR receiver - x2
- IR LED diode - x2
- 220Ω resistor - x2
- Arduino UNO - x2
- Breadboard - x2
- USB Type A/B cable - x2

See image 2.2 for how the circuit was constructed.

2.4 Code implementation

The following sketch was utilized to realize the scenario:
Two Arduino libraries were developed for this purpose:

- A51Cipher: This library takes care of the key cipher generation, as well as encrypting any message passed in the binary format.
- STIR: The Serial-to-Infrared communication library. This library is responsible for receiving and forwarding Serial as well as IR messages to and from the Arduinos connected PC.

```
#include <A51Cipher.h>
#include <STIR.h>

A51Cipher cipherLib;
STIR commLib(0,0);
bool keyStream[228];
const uint8_t ROLE = 0x1;
bool cipherKeyIsSent = false;

void setup()
{
    cipherLib = A51Cipher();
    if (ROLE == 1)
    {
        cipherLib.createCipherKey(keyStream);
    }

    STIRConfig config(0, 0, 0, 2, 3, ProcessIncoming::WRITETOSERIAL);
    commLib = STIR(config, ROLE);
    commLib.beginListen();
}

void loop()
{
    if (ROLE == 1 && !cipherKeyIsSent)
    {
        commLib.sendBinary(keyStream);
        cipherKeyIsSent = true;
    }

    commLib.communicationLoop();
    if (commLib.bufferMessageFromPCSize > 0)
    {
        bool* encryptedMessage = cipherLib.encryptMessage(commLib.
    bufferMessageFromPC);
        commLib.sendBinary(encryptedMessage);
    }
}
```

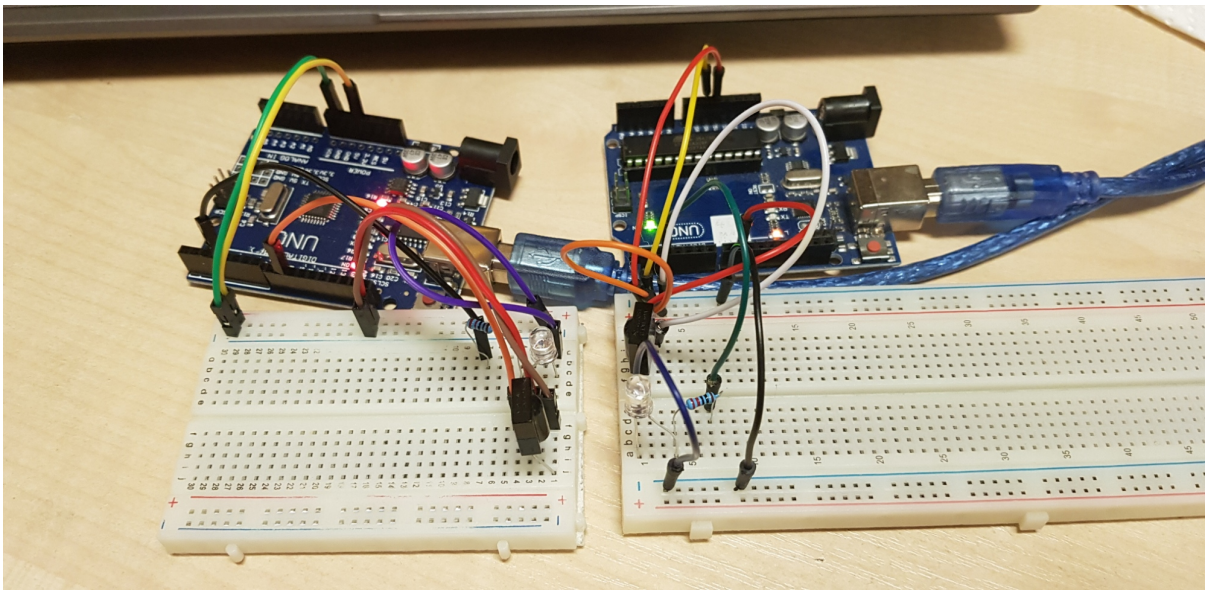

Figure 2.2: Circuit for list 2

The procedure implementation is the following:

- Two Arduinos are assigned roles: 1 for master, 2 for slave. This is defined by the ROLE constant in the main sketch.
- Master device creates the A51 cipher key, and is ready to distribute it.
- In the initial loop, master device transmits the key cipher via IR to the slave Arduino device, which saves it re-initializes the A51Cipher library with it.
- In the main loop, each device listens out for any Serial message from the connected computer encrypts and forwards it in IR via the NEC protocol by the help of the STIR library.
- Similarly, in the main loop, each device listens out for any incoming IR signals decodes and forwards it to the connected computer via the Serial port.

Visual Studio was utilized as the main development environment. In order for the build and C++ linking operations to be successful, certain library calls were mocked. This enabled us to write unit tests to detect flaws in our program logic quicker. The whole code repository can be viewed on `https://github.com/hakanyildizhan/pwr-embedded-sec-sys-labs`.

3 | Final remarks

3.1 List 1

Obtained messages' protocol was identified as NEC Infrared Transmission Protocol. Messages were decoded and successfully reproduced.

3.2 List 2

Separate Arduino libraries were written in order to implement the A5/1 Cipher encryption and message sending/forwarding. Arduino devices were programmed to take in a string message via serial port, encrypt it via A5/1 cipher algorithm and forward it via an IR LED via NEC protocol, which was detected by the IR receiver at the receiving end of the transmission, decrypted using the shared cipher key and forwarded back to the connected computer via serial port.

Bibliography

[1] *NEC Infrared Transmission Protocol*, https://techdocs.altium.com/display/FPGA/NEC+Infrared+Transmission+Protocol, Accessed: 19-10, 2022.