Embedded Security - Report 1

Bartosz Drzazga, Mateusz Jachniak

20.10.2020

Contents

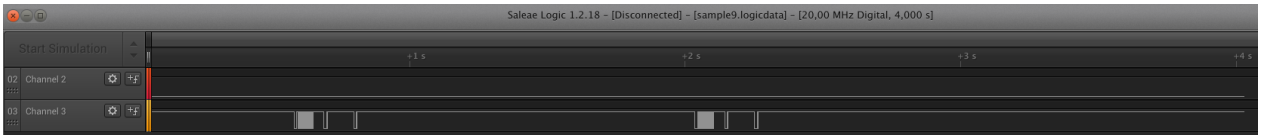
1 List 1

1.1 Exercise

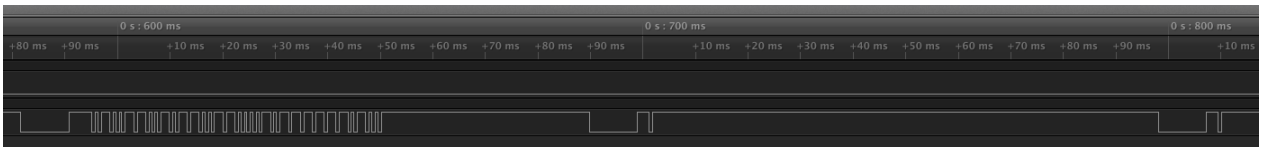
IrDA and oscilloscope showcase. Saleae download page for logic analyzer software. We used infrared receiver TSOP31236 datasheet and some code for Arduino. Traces of IrDA communication from the remote as picked up by the receiver will be send to respective students via email for decoding.

1.2 Knowledge gathering

To open our sample, we begin with installation of `Saleae Logic` software. After installation, we take our first look at IrDA traces in analyzer:



We can see two signals, the first starting at about 0.5s in the sample, and the second starting at about 2s mark. After quick inspection, we conclude that both signals are the same. From now on, we will only focus on one of those. Let's take a closer look at the first signal:



Even without any knowledge about the signal (protocol, type of data transmitted), it is easy to identify a few distinctive parts in the sample. The traces start with a spike lasting for about 10ms, followed by 5ms long break. Then, for about 50ms, some data is transmitted. After about 100ms since the start, there is another spike of approximately 10ms, 2ms break and 1ms spike. Exactly the same signal is recorder after another 100ms break.
In our research, we find out that the sample recording shows transmission in `NEC Infrared Transmission Protocol`. To learn more about this protocol, we use online resources like:

- https://techdocs.altium.com/display/FPGA/NEC+Infrared+Transmission+Protocol
- https://www.sbprojects.net/knowledge/ir/nec.php
- https://exploreembedded.com/wiki/NEC_IR_Remote_Control_Interface_with_8051

Here, we give a short summary of the protocol. Key features are:

- pulse distance encoding of the message bits
- carrier frequency of 38kHz
- 8 bit address and 8 bit command length

- logical '0' – a 562.5µs pulse burst followed by a 562.5µs space, with a total transmit time of 1.125ms
- logical '1' – a 562.5µs pulse burst followed by a 1.6875ms space, with a total transmit time of 2.25ms
- least significant bit first

When a key is pressed on the remote controller, the message transmitted consists of the following, in order:

- 9ms leading pulse burst
- 4.5ms space
- 8-bit address for the receiving device
- 8-bit complement of address
- 8-bit command
- 8-bit complement of command
- final 562.5µs pulse burst in order to be able to determine the value of the last bit

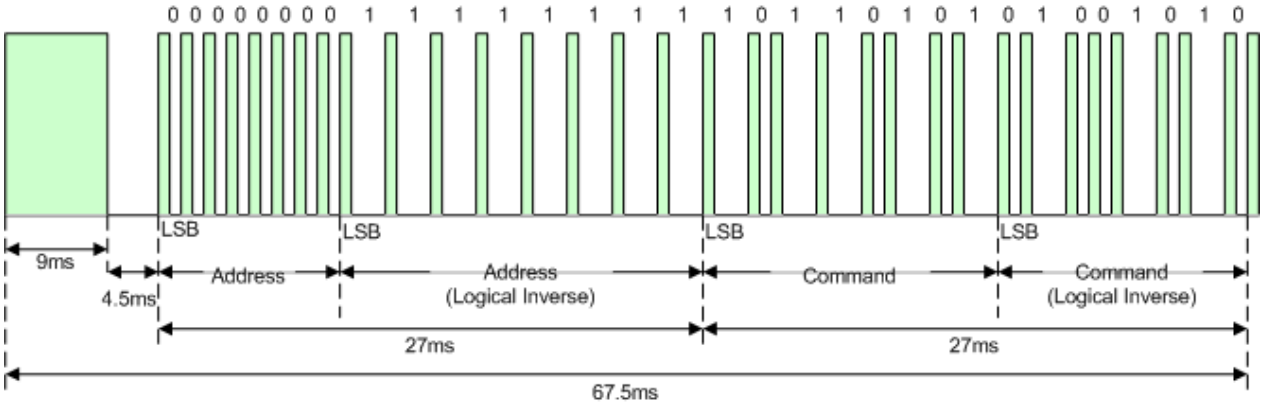


Figure 1: Example frame using the NEC IR transmission protocol. Source: techdocs.altium.com

Note that the whole frame always lasts 67.5ms (without the final burst) because each frame always contains the same number of 1 and 0. This is the result of transmitting complements of address and command.
If the key on the remote controller is kept pressed, a repeat code will be issued. A repeat code will continue to be sent out at 108ms intervals, until the key is finally released. The repeat code consists of the following, in order:

- 9ms leading pulse burst
- 2.25ms space
- final 562.5µs pulse burst

The protocol description indeed matches the traces in our sample. That confirms we are dealing with `NEC Infrared Transmission Protocol`.

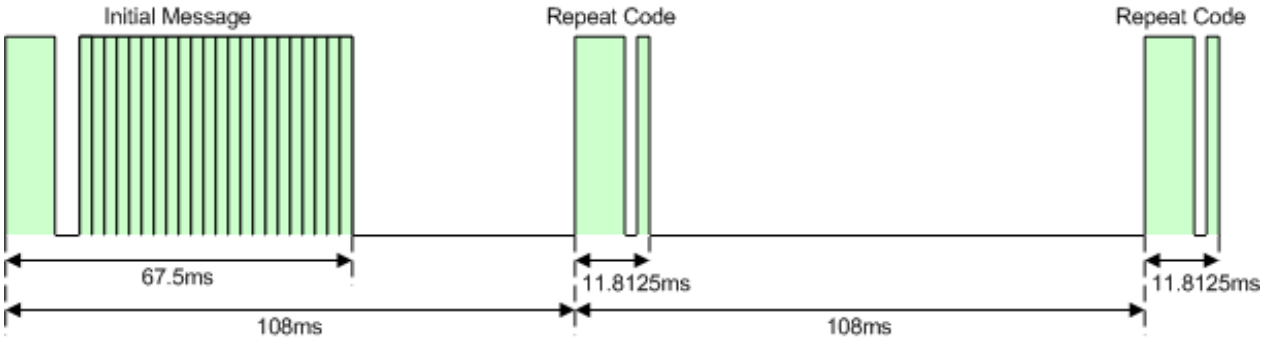


Figure 2: Example repeat codes sent. Source: techdocs.altium.com

1.3 Solution

As we stated before, in our sample we observe two similar signals. This means the same button on a remote was pressed. For each signal, we notice two repeat codes. This means that for each time the button was pressed for about 200ms. Let's decode the message:

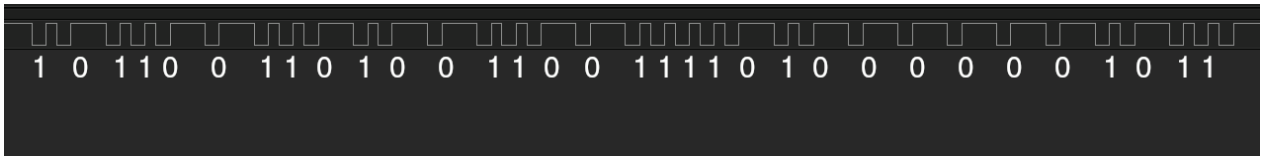


Figure 3: Decoded signal

The address of the device is 11001101b (CDh) while the command is 00101111b (2Fh).

1.4 Outcomes

IrDA communication is a low-priced and generally used short-range wireless technology. It is usable, because secure data transfer, line-of-sight and a very low bit error rate that performs it very popular. It is widely used in e.g. at automobiles, computers, microelectronics, medical devices, household apparatuses. It is also easy in use, as we can see in example above.

2 List 2

2.1 Exercise

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).** (choosen)
- Adruino 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 Knowledge gathering

We start with research on A5/1. For this we use online resources, mostly https://en.wikipedia.org/wiki/A5/1. A5/1 is one of several algorithms specified for use in GSM for securing transmission. A5/1 is based on a combination of 3 linear feedback shift registers (LFSRs) with irregular clocking (depending on clocking bit).

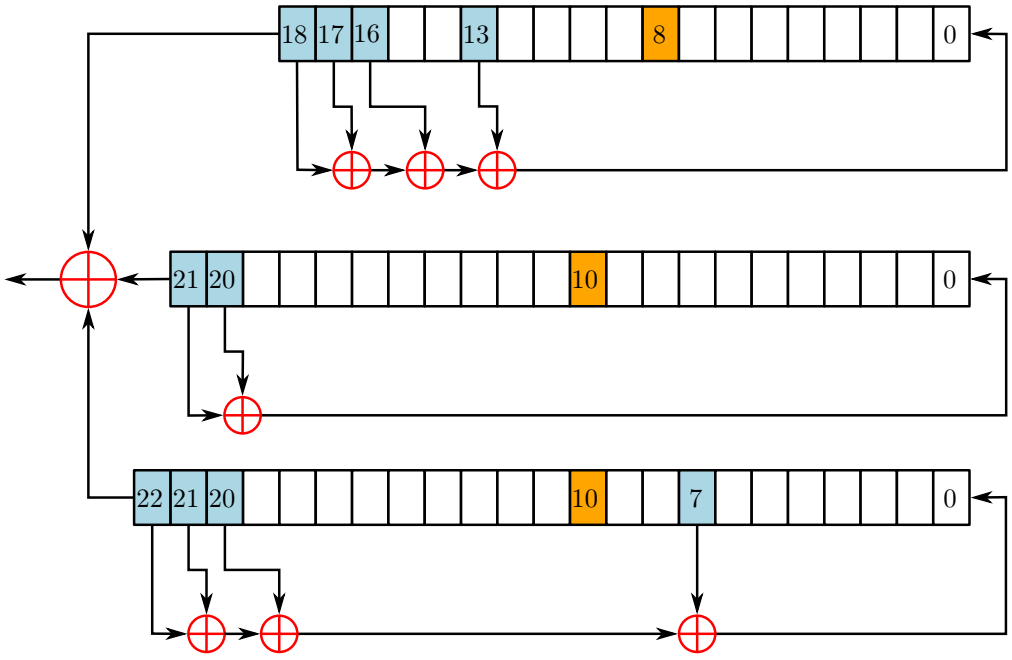


Figure 4: Registers of A5/1 Source: https://en.wikipedia.org/wiki/A5/1

The 3 LFSRs are specified as follows:

LFSR number	bits	clocking bit	tapped bits
1	19	8	13, 16, 17, 18
2	22	10	20, 21
2	23	10	7, 20, 21, 22

The bits are indexed with the least significant bit (LSB) as 0. Each LFSR has an associated clocking bit. For every cycle, the majority bit is determined, i.e. each clocking bit is examined and the value present in two or three bits is the majority bit. Every register with clocking bit matching the majority bit is clocked. As the result, at each step at least two registers are clocked.

The keystream is initialized by setting the registers to zero. Then, a 64 bit key is incorporated. For ith bit of the key, the bit is XORed with LSB of each register and registers are clocked. In the same way, a 22 bit frame number is added. Then all LFSRs are clocked in accordance to majority rule for 100 cycles. After initialization sequence, the system is ready to produce keystream.

The next step of our research focused on Arduino platform. We installed Arduino IDE and set up development environment, e.g. UART connection via USB. We looked up Arduino libraries for IR communication and decided to use `IRremote Arduino Library` https://github.com/z3t0/Arduino-IRremote. This resources include connection examples and sample code.

2.3 Solution

In order to construct a working, encrypted communication link, we start with implemenation of A5/1 cipher. We decide to use a `Python` implementation by Aayush Dixit found on https://github.com/dixitaayush8/A5-1/blob/master/a51.py. This is a minimal working implementation, sample program execution:

```
$ python test.py
Enter a 64-bit key:
    0101001000011010110001110001100100101001000000110111111010110111
[0]: Quit
[1]: Encrypt
[2]: Decrypt
Press 0, 1, or 2: 1
Enter the plaintext: test
test
11101100111011111001001011100000
$ python test.py
Enter a 64-bit key:
    0101001000011010110001110001100100101001000000110111111010110111
[0]: Quit
[1]: Encrypt
[2]: Decrypt
Press 0, 1, or 2: 2
Enter a ciphertext: 11101100111011111001001011100000
test
```

The next step is to transmit the ciphertext to Arduino using UART. Arduino runs a program that reads the input and emits the encrypted message. The sending software reads binary output of A5/1 and transmits the stream in blocks of 32 bits.

```
#include <IRremote.h>

IRsend IrSender;

void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
    Serial.begin(9600);
}

unsigned long tData = 0;
int ctr = 1;
void loop() {
    if (Serial.available()) {
      unsigned long tmp = Serial.read() - 48; //ASCI char to int
      if (tmp == 0 or tmp == 1) {
          tData += tmp << (32-ctr);
          ctr++;
      }
      if (ctr == 32) {
        IrSender.sendNEC(tData, 32);
        Serial.print(F("sendNEC(0b"));
        Serial.print(tData,BIN);
        Serial.println(F(", 32)"));
        delay(50);
        ctr = 1;
        tData = 0;
    }
  }
}
```

After the ciphertext is transmitted, we use another Arduino to receive the message. This board runs the following code:

```
#include <IRremote.h>
int input_pin = 4;

IRrecv IrReceiver(input_pin);
decode_results signals;

void setup()
```

```
{
  IrReceiver.enableIRIn();  // Start the receiver
  IrReceiver.blink13(true); // Enable feedback LED
  Serial.begin(9600);
}

void loop() {
  if(IrReceiver.decode(&signals)) {
    long c = signals.value;
    Serial.println(c, BIN);
    IrReceiver.resume();
  }
}
```

The program outputs on a serial port the same binary ciphertext. The last step is to read the binary stream from serial on a PC and decrypt the message using the same key.

2.4 Outcomes

As we mentioned in previous secttion 1.4 IrDA-enabled devices can easily interact inexpensively and securely. As a curiosity we can mention, that when we plug in Arduino to computers, we have to change mode on USB port using the command "chmod 777 /dev/ttyUSB0". That knowledge we got from the Make me Hack movie - https://www.youtube.com/watch?v=6_Q663YkyXE.
Next interesting thing could be fact, that max length on one streamed block is 32 bits. This limitation comes from the types of variables used in Arduino Uno (unsigned long).

2.5 Pictures

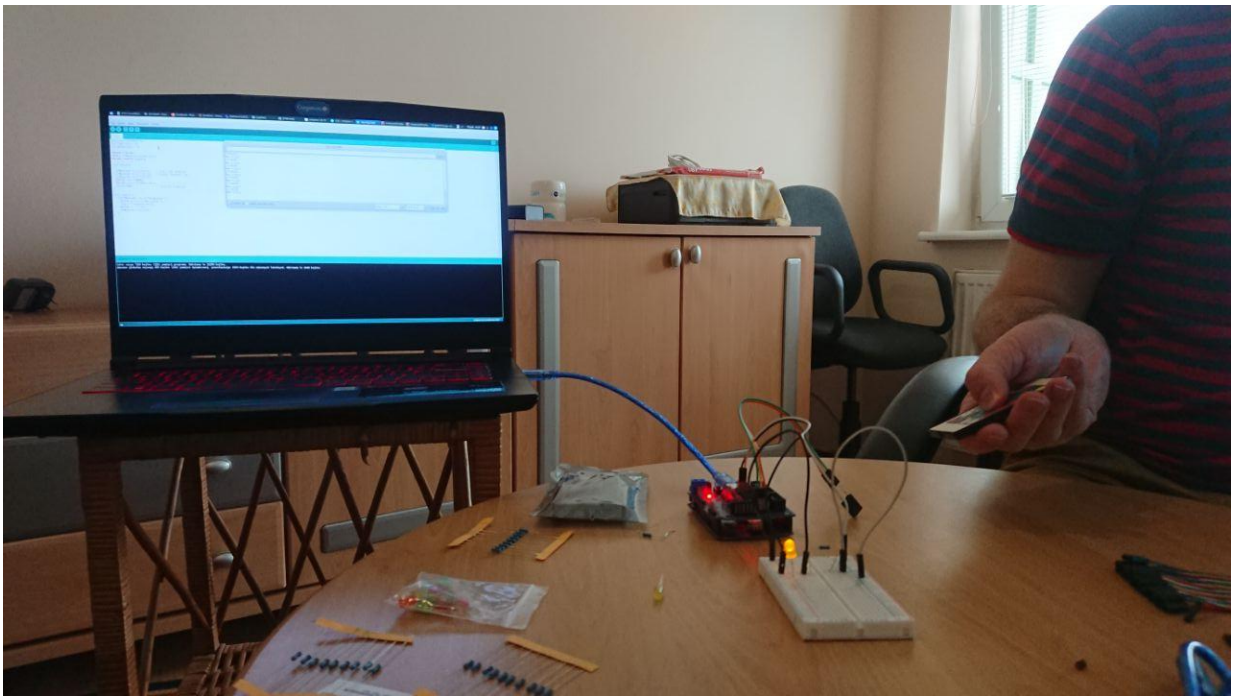


Figure 5: Our test of reciving a signal from other device

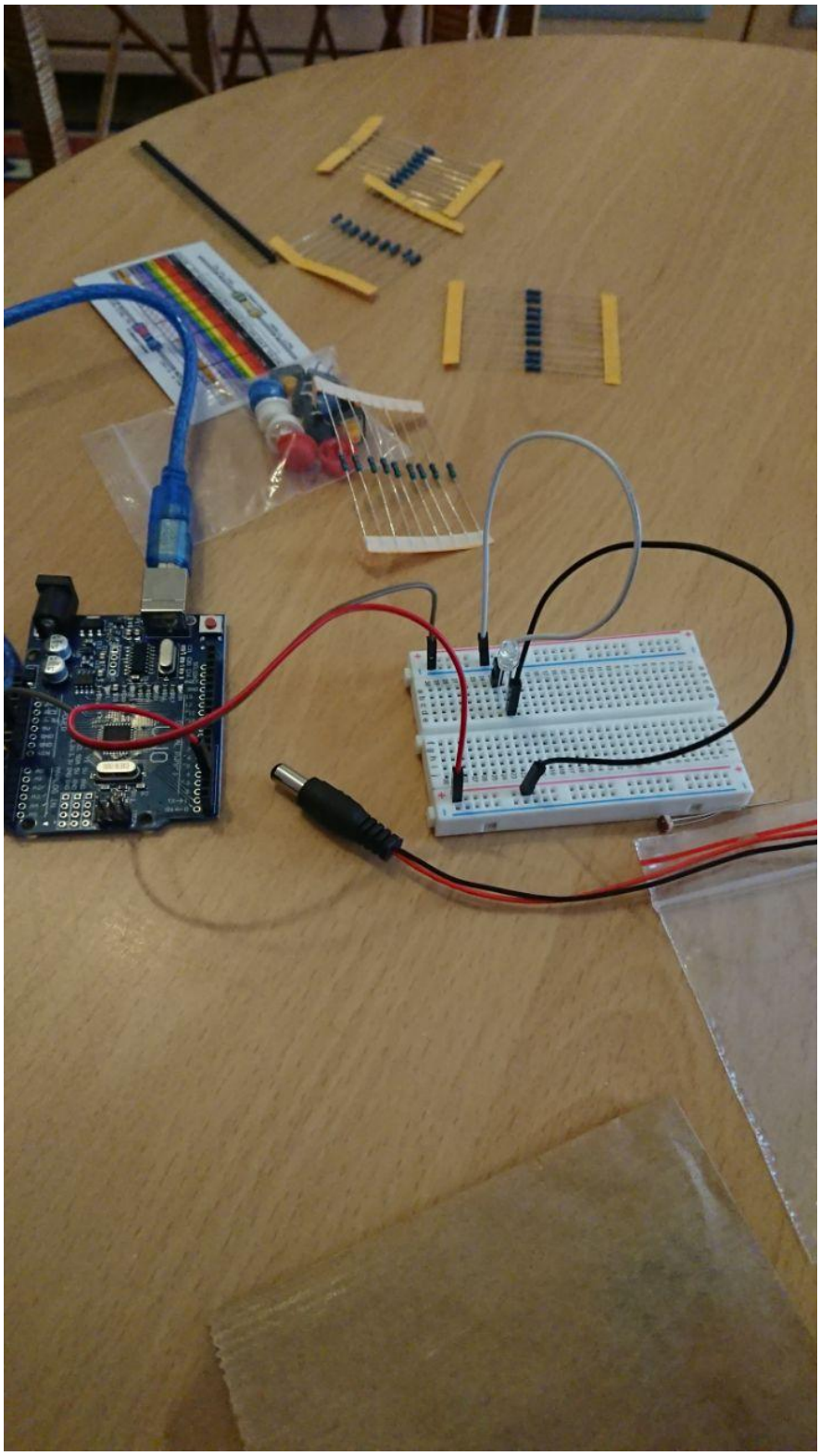


Figure 6: Our test of reciving a signal from other device