Embedded Security (Summer Term 2019)
Dr. Stefan Nürnberger
CISPA Helmholtz Center for Information Security / Saarland University

Solution of Exercise Sheet #1

1 The Mysterious Chip (9 points)

A friend of yours works at the stock exchange. He tells you that they recently found a mysterious chip attached to their network switches that might manipulate stock if it received the right command from an insider. He wants you to inspect that chip – but he cannot give you the original because the police is already investigating it. He managed to copy the firmware of the chip and gave it to you.

Your task is to put that firmware on your ATtiny and retrieve the secret that the chip computes and stores in its internal EEPROM memory.

(a) To achieve that task, you have to use the Arduino as an in-circuit serial programmer (ICSP). ICSP is a common way to flash the content (code, data) of a microcontroller while it is already built into a circuit, e.g., already in a sold product. ICSP is an easy way to manufacture products because the bare chips are soldered onto PCBs and the software is flashed into the chip in a later step using ICSP[1].

i. To get started, hook up the Arduino as Master and the ATtiny as slave as shown in Figure 1.

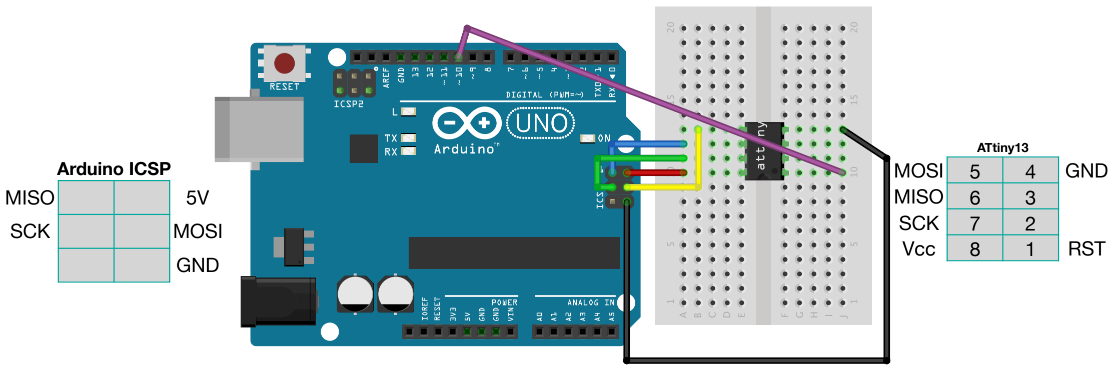

Figure 1: Arduino used as In-Circuit Programmer (ISP) to program the ATtiny

In this setup, the Arduino is able to flash (=overwrite!) the ATtiny's content. To do that, the Arduino needs the ISP software installed on it to work as a programmer for the ATtiny.

[1]The ICSP protocol uses MISO (Master Input Slave Output) and MOSI (Master Output Slave Input) to communicate while flashing. The data is synchronised using SCK (Serial Clock). The receiving chip will only accept to be flashed while its reset line (RST) is pulled low. You donŠt have to understand all this. The ArduinoŠs software will take care of all this.

ii. First, install the ISP firmware to the Arduino by issuing the following command:

```
avrdude -p atmega328p -c arduino -P /dev/ttyUSB0 -b 115200
-U flash:w:task1/ArduinoISP.hex
```

You have to replace `/dev/ttyUSB0` with your Arduino's serial port. The file `task1/ArduinoISP.hex` is the ISP software that is to be flashed into the Arduino.

If that step was successful, you can then write the ATtiny's firmware through the Arduino ISP using the following command:

```
avrdude -c avrisp -P /dev/ttyUSB0 -p t13 -b 19200
-U flash:w:task1/ex1_?????-?????.ino.hex
```

Replace the `?????` with the appropriate file name that matches your group's matriculation numbers. Please note that the secrets are personalised and you'll get 0 points of you use somebody else's secrets.

If that step worked, the ATtiny now runs the secret firmware discovered at the stock exchange.

(5 points) iii. Now it's your turn to get the EEPROM memory from the ATtiny13 to check what secret it stored. The EEPROM can be retrieved using

```
avrdude -c avrisp -P /dev/ttyUSB0 -p t13 -b 19200
-U eeprom:r:dump_eep.hex:r
```

The file `dump_eep.hex` now contains the entire 64 bytes of ATtiny EEPROM memory. Now its up to you to find a way to decode the `.hex` file format and find the secret. Please put the secret in your solution.

Solution
The solution can be reached by following the previous steps, the answer is unique for every team. The secret is an ASCII encoded string stored somewhere in the EEPROM (the start address is deterministic).

(2 points) (b) The discovered secret is surrounded by other data. Check if the address of the secret is deterministic and if the other data changes? How did you do that and what was the result?

Solution
If you try to run the last command multiple times, you will find that the secret is located at the same location every time, also the surrounding data is the same, so it is definitely deterministic.

(2 points) (c) Now that you know how .hex files are built, have a look at the original `ex1_?????-?????.ino.hex` file. Is the secret already visible in there? If yes, where? If no, give an example of how you would avoid a secret to show up.

Solution
No, it is not visible. One way to hide the secret is to encrypt it and decrypt it during runtime.

(+4 bonus) (d) You can convert the Intel HEX format firmware for the ATtiny to binary and then inspect the instructions using a disassembler, e.g. `https://onlinedisassembler.com/odaweb`. Can you see how the secret is handled and how the address in EEPROM is calculated?

Solution
Check out the code for details on how it works in C.

```
#include <EEPROM.h>

//Secret and key are defined at compilation
const char secret[] PROGMEM = SECRET;
const byte key = KEY;

//This function is heavily optimised by the compiler.
byte hex2int(char x){
    switch(x){
        case '0': return 0;
        case '1': return 1;
        case '2': return 2;
        case '3': return 3;
        case '4': return 4;
        case '5': return 5;
        case '6': return 6;
        case '7': return 7;
        case '8': return 8;
        case '9': return 9;
        case 'A': return 10;
        case 'B': return 11;
        case 'C': return 12;
        case 'D': return 13;
        case 'E': return 14;
        case 'F': return 15;
        default: return 0;
    }
}

void writeEEPROM(){
    //Randomise the memory - not really but should suffice
    byte seed = key;
    for(short i=0; i < 64; i++){
        seed = ((seed + 1) * (seed | 0x1C));
        EEPROM.write(i, seed);
    }
    //Randomly select initial address -- not really
    byte addr = (seed & 0x3C) % 0x0F;
    short j=0;

    //Decrypt the secret and store in EEPROM
    //pgm_read_byte is a MACRO to read PROGMEM memory.
    for(short i=0; i < (short) sizeof(secret)-1; i+=2){
        byte a = hex2int(pgm_read_byte(&(secret[i]))) << 4;
        byte b = hex2int(pgm_read_byte(&(secret[i+1])));
        EEPROM.write(addr + j++, (a | b) ^ key);
    }
    //Add a 0x00 byte just in case
    EEPROM.write(addr + j, 0x00);
}

void setup(){
  writeEEPROM();
}

void loop(){
}
```

One way to approach the disassebly is:

- Find what could likely be the encrypted secret – an HEX encoded string. It is easy to see that such string is double the size of the secret.
- Assuming the found string is the encoded secret, one could find the instructions that points to it. From the disassembly, you find two of such instructions (which respectively point to the i and $i + 1$, where i is a control variable for a loop. i is checked against the length of the string itself).
- From the sequence of instructions, one can see the two hex digits are joined to make a full byte.
- This full byte is then XORed with the R16(?) register, which is filled with a fixed value. This value is the key.

2 Power-Hungry (11 points)

Your friend has discovered another chip that intruders used to open a door. He sent you that chip's firmware as well (`/task2`).

He also told you that this time, the chip was receiving external data using two wires connected to it. These two wires transmit a binary signal using `CLK` and `DATA`. If the signal encodes a secret password, the chip would open the door to which it was connected.

(a) First install the new firmware (`/task2/attiny-sidechannel.hex`) using the ICSP method from before. When your ATtiny runs the new code, disconnect all wires but the `GND` and `5V`. You'll need those wires to send a digital signal from your Arduino in the next step:

Your task is to find the byte sequence that they used to open the door.

i. Connect the Arduino as depicted in Figure 2:

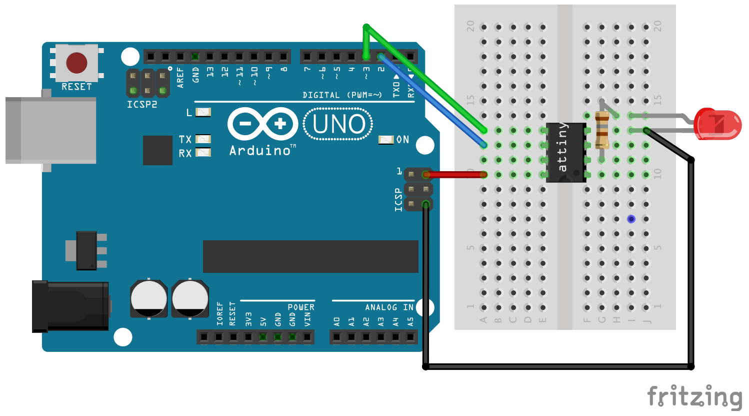

Figure 2: Arduino can send bits using two wires: CLK and DATA. The ATtiny has a feedback LED connected through a 180 Ohm resistor.

The CLK (ATtiny pin 6) and DATA (ATtiny pin 5) are both driven by the Arduino. When the ATtiny is waken up from standby mode (by a high CLK line), it waits for the clock line to go low again. When a falling edge is detected, it reads out the data line (interpreting a high level as 1 and a low level as 0); to receive another bit the clock line needs to go high (and then low) again.

(1 point) ii. The LED connected to pin 2 will light up every time it receives the byte `0xFE`. This is for you to test the successful setup of your communication. Use the provided file `/task2/task2-arduino-power-analysis.zip` in Platform.IO as a starting point. It can already send data. Add the appropriate code to send `0xFE` (turn on the LED) and check if it works.

Solution

Simply sending the byte 0xFE continuously using the "shitOutByte" function will make the LED light up. This is example code.

```
while(1){
    digitalWrite(CLK,HIGH);
    delay(15);
    shiftOutByte(0xFE);
    delay(250);
    digitalWrite(CLK, LOW);
    delay(15);
}
```

(2 points) iii. Apart from the special `0xFE` byte, each byte is processed as follows:

```
void processPassword()
{
  for(int i = 0; i < strlen(password); i++) {
    byte b = receive_byte();
    if (b == 0xFE) {
      digitalWrite(PIN_LED, HIGH);
    }
    if (b != password[i]) {
      goto_sleep();
    }
  }
}
```

Analyse the code and see which side channels you could exploit. Name them and explain why.

Solution
There is a power consumption side channel. We can send the key byte by byte while monitoring the power after every byte we send. Once we find out that the power drops, we know that the microcontroller has gone to sleep mode and that the last byte we sent is false. We then iterate over different values till it is correct. This side channel decreases the number of tests required from 2^{9*8} to $2^9 * 8$ which is a reasonable number of tests.
Another (unintended – but possibly true) side channel is to send a key composed by a test byte X followed by a 0xFE byte. If X is wrong, the ATTiny goes to sleep, thus it does't process the sequent byte 0xFE (the ATTiny is supposed to wake-up when CLK is high, however while going to sleep the interrupts are cleaned and disabled for some μsec, which implies it will fail to decode the 0xFE byte).

(b) To analyse power consumption, prepare the Arduino and the ATtiny as follows:
i. Arduino: Add a 120 Ohm (brown-red-brown) resistor to the GND line of the ATtiny to be able to measure the current on Analog input 0 (A0) of the Arduino (see Figure 3).
The purple wire[2] will feed the voltage after the resistor back the A0, thereby

[2]You can of course use any colour

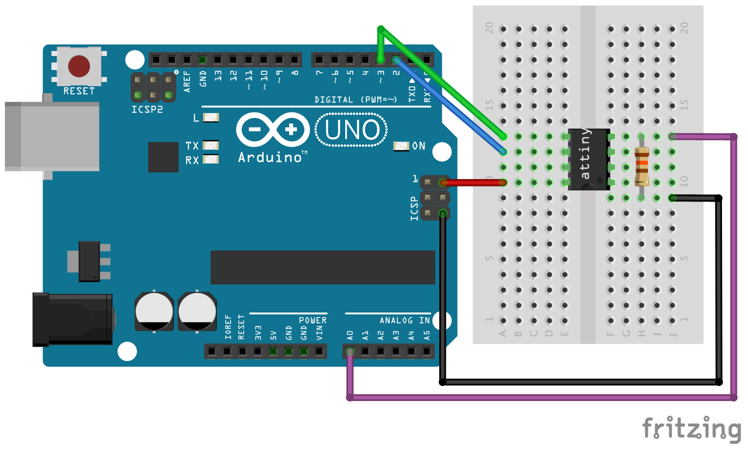

Figure 3: Current measurement using a resistor in series with the ATtiny's GND. The LED is removed in order not to disturb the current reading.

measuring the voltage drop across the resistor when the ATtiny consumes current. This voltage can then be read by the Arduino's ADC (analog to digital converter).

(1 point) ii. Find out how accurate the Arduino Uno's ADC is (e.g. how many bits) and which lowest number and highest number corresponds to which voltage.

Solution
The Arduino Uno's ADC has a resolution of 10 bits which means it can output values 0 to 1023 where 0 corresponds to 0V and 1023 corresponds to 5V.

(1 point) iii. Then check the provided Platform.IO source code for the Arduino and find out which maximum voltage the A0 input expects and to which values (integers) the different measured voltages will map.

Solution
As explicitly stated in the source code, the following line sets the reference voltage to 1.1V, which is the maximum voltage that can be measured.

```
analogReference(INTERNAL);
```

This means now 1023 will correspond to 1.1V.

iv. ATtiny: The current consumption of a processor increases with the clock speed. The clock speed of the ATtiny can be set to up to 9.6 MHz. This is done by setting internal fuses. Use the following command to do so:

```
avrdude -c avrisp -P /dev/YOURPORT -b 19200 -p t13
-U flash:w:task2/attiny-sidechannel.hex
-U lfuse:w:0x7a:m -U hfuse:w:0xff:m
```

(c) Everything is set up now.

(1 point) i. Explain how you think you can find the expected secret using the voltage measurement on A0.

Solution
If the microcontroller goes into sleep mode, the power consumption goes down, so the current it draws goes down, so the voltage across the resistor goes down, so the measurement becomes smaller. We can test for all values of the first byte and check which makes the microcontroller not go to sleep (i.e. measurement does not drop significantly), this will be the correct first byte, then we repeat for subsequent bytes.

(5 points) ii. Extend the provided Arduino code to smartly iterate over the 9-character key that the ATtiny expects. Can you find the key? How did you do it? What is the key?

Solution
The key is “A. TURING”. In the function “test_key” you need to return the number of successful bytes (in any reasonable way), note that the variable “cameUntilByte” is shadowed, so it’s value outside the loop will be always zero. You have to take care of that so it won’t be shadowed. Also you need to do other tricks to make it work as the microcontroller does not always wake up reliably. For instance, you need to do things like rejecting unreasonable results, increasing the delay, or test if the power after wake is adequate.