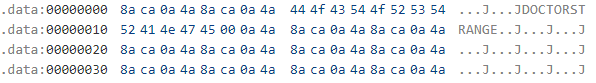
Solution to Exercise #1

Beshoy Saad 2572741 Vitalij Minor 2565427

# The Mysterious Chip

1. 1. Using onlinedisassembler.com to inspect dump\_eep.hex we get the following output:



We can see that decoding the hex data into ASCII gives a repeating pattern of three unprintable characters then the letter J, except for the address range 7 – 14 where the unique phrase “DOCTORSTRANGE” is present, which we assume to be the secret.

1. By flashing the ATtiny13 chip multiple times and repeating the steps to get the EEPROM content, we find that the address of the secret as well as the surrounding data does not change. We also ran the ATtiny13 firmware multiple times after flashing once (resetting the chip between runs) and we got the same result. So we conclude that the address of the secret is deterministic, and so is the surrounding data.
2. Using avr-objdump we can disassemble the hex file and inspect the assembly code. We can see that the secret is not directly visible in the firmware. One way to avoid having a secret show up in the firmware image is to calculate it during runtime as this obfuscates the final value of the secret.

# Power Hungry

1. 1. We added the following code at the beginning of loop():

digitalWrite(CLK, HIGH);

delay(20);

shiftOutByte(0xFE);

delay(500);

return;

The LED started blinking every 0.5 seconds.

1. We can exploit the Power Analysis side channel since the chip’s power consumption changes according to the value of the received byte.
2. 1. Consulting the online Arduino reference, we find that the Arduino ADC’s output is 10-bit wide. The digital value 0 corresponds to 0 volts while the digital value 1023 corresponds to the analog voltage reference which can be supplied externally, taken from VCC, or set to the internal ADC voltage reference.
   2. We see that the analog voltage reference is set to the internal 1.1 volts. This means that the 0 – 1023 range now corresponds to 0 – 1.1 volts, giving better resolution. Thus, A0 expects a maximum value of 1.1 volts.
4. According to the provided code snippet, the ATtiny13 firmware checks the received byte against the first character of the password. If they don’t match, the chip goes to sleep (which can be detected via power analysis). If the byte and the character match, the firmware attempts to receive another byte over the serial link and checks if it matches the second character of the password, and so on. Analyzing the power consumption of the ATtiny13 while it’s running the firmware and iterating over the possible byte values, we can deduce the password character by character. The voltage measurement on A0 gives us information on the power state of the ATtiny13, since high current consumption means high voltage measured by the ADC, indicating the chip is in active state, while low measured voltage means the chip is in sleep mode.
5. By iterating over the possible byte values (0x0 – 0xFF) and detecting when the chip stays awake (i.e. the sent byte matches a character of the key) we were able to extract the complete key character by character. The extracted key is “**A. TURING**”. The code for extracting the key works as follows:
   1. Initialize a buffer with the same length as the key to 0.
   2. Wake up the ATtiny13 chip. Measure its power consumption while awake.
   3. Send the first byte of the buffer to the chip. Measure the power consumption after the byte is received.
   4. If the chip goes into sleep mode, this means the sent byte was incorrect. Wait for some time, increment the byte by 1, and then repeat from step c.
   5. If the chip stays awake, this means that the sent byte was correct. Send the next byte.
   6. Repeat for each byte of the key until the full key has been recovered.