Timing Attacks

Zachary Driskill and Jacob Brown
Brigham Young University
450 Engineering Building
Provo, UT 84602
{zd227, jacobb00}@byu.edu

***Abstract*—We did a timing attack.**
***Index Terms*—timing attack, FPGA, constant-time**

I. Introduction

Timing attacks are a type of side-channel attack. TODO more intro.

To further learn about timing attacks, explored a timing attack that exploit a non-constant compare function to extract a secret key. Our source of inspiration is Joe Grand, who demonstrated this simple timing attack using a microcontroller, four buttons, and an oscilloscope [1]. The non-constant compare function is simply a compare function that exits as soon as a mismatch occurs (see the code snippet below).

```
def non_constant_compare(guess):
    for x, y in zip(key, guess):
        if x != y:
            return False
    return True
```

By measuring how long the compare function takes to execute, one can estimate how much of the input matched. For example, say we are guessing a 4 character string. We guess 26 strings, each simply a single letter repeated four times (”aaaa”, ”bbbb”, ”cccc” ... ”zzzz”), and measure the execution time of the compare function. The guessed string with the lowest compare time has the correct first digit. This process is repeated for the other three digits, making sure to put the discovered digits in front of the guessed digits.

We implement two simple timing attacks on the HaHav3 board that exploit a non-constant compare function:

1) **Human to FPGA** - the FPGA is the victim and a human is the attacker. The human uses an oscilloscope to measure compare time.
2) **FPGA to MCU** - the MCU is the victim and the FPGA is the attacker. The FPGA counts cycles to measure compare time.

II. Human to FPGA

For the human to FPGA timing attack, the non-constant compare function is implemented on the FPGA present on the HaHav3 board and a human carries out the attack. The FPGA compares an input guess to a secret key and indicates whether it is success or fail. A human uses the three buttons on the HaHav3 board to input a four pin guess and uses an oscilloscope to measure the time between the last button press and the compare function result. It is important to note that the FPGA waits until all four input values are received before beginning the compare function, so the final button press can act as a time reference for the start of the comparison function.

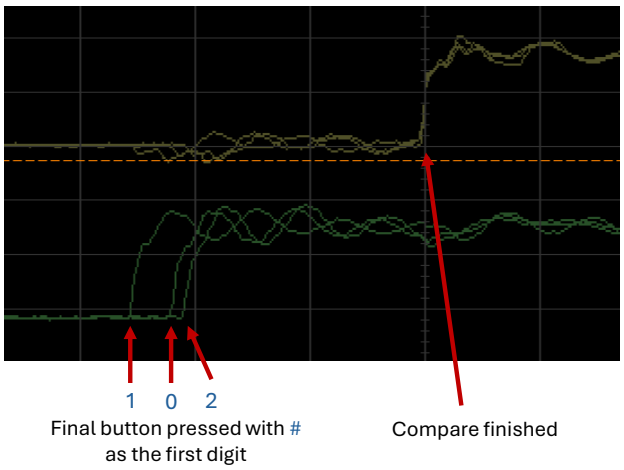

Fig. 1. Oscilloscope waveform capture showing when the final button was pressed for each input sequence and when the comparison finished. The ’compare finished’ point is centered at 0.

An oscilloscope waveform capturing the first step of the attack is shown in Figure 1. The green waveform is for the buttons (an OR between the three buttons) and the yellow waveform is for the compare complete signal (an OR between the success and fail signals). The three input guesses were 0000, 1111, and 2222 and the waveforms overlaid on each other in the figure. Looking at the delay from button press to compare finish, the delay is longest when 1 was the first digit, indicating that 1 is the correct value of the first digit.

The input guesses and compare delays for all steps in the attack are shown in Table I. The fourth digit is not susceptible to the time attack because, whether correct or incorrect, the compare function runs the full compare, so while the first three digits are discovered my measuring time, the fourth digit requires brute force.

III. FPGA to MCU

In the FPGA to MCU attack, MCU on the HaHav3 board is programmed with a secret key, and listens for guesses fr0m the FPGA on the HaHav3 board. When it receives a guess from the FPGA, it compares each byte of the guess with the key, and breaks as soon as it finds a byte that does not match.

TABLE I
INPUT VALUES AND COMPARISON DELAYS FOR EACH TESTED DIGIT.

Digit	Input Value	Delay (ns)
0	0000	109
	1111	128
	2222	104
1	1000	160
	1111	131
	1222	127
2	1000	153
	1011	153
	1022	173
3	1020	165
	1021	174
	1022	172

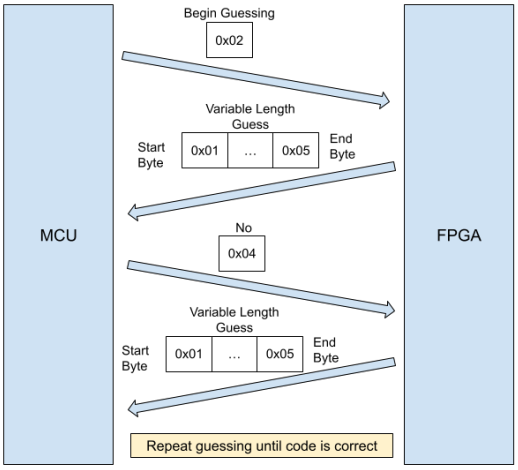

Fig. 2. Demonstration of a transaction between the MCU and the FPGA over the CM bus

The MCU and FPGA communicate over the CM bus on the board. This bus was setup up in previous labs to act as a one way communication channel for the MCY to send data to the FPGA. However, this timing attack requires two way communication, with the FPGA sending guesses to the MCU, and the MCU responding to say if the guess was correct. We created a simple protocol for the the FPGA and MCU to use. Values 0x00 through 0x05 are reserved as op codes, meaning they are not possible bytes in the secret key:

- 0x01 signal from FPGA that it is beginning a guess
- 0x02 signal from MCU for FPGA to begin sending guesses
- 0x03 signal from MCU that guess was correct
- 0x04 signal from MCU that guess was incorrect
- 0x05 signal from FPGA that it has finished sending its guess

A example of how the program would begin is shown In Figure 2. When the FPGA is listening for a message from the MCU, assigns the CM bus as high impedance so there is no conflicts from both entities driving the bus simltaneously.

Once this communication protocol was devised and implemented, we were able to apply a timing attack. For each byte in the secret key, the FPGA guesses all 250 possible values, and counts the delay between when it sends the end byte and when it receives a "NO" byte from the MCU. The byte value that had the longest delay is determined to be the correct byte, and this value is set as the first byte in all future guesses. This process is then repeated for all following bytes of the secret code.

This timimng attack is much more effective than a brute force attack would be. We successfully guessed a 32 byte key using this timimng attack. Table III shows how long the timing attack took to guess secret keys of different lengths.

TABLE II
AMOUNT OF TIME IT TOOK FOR OUR FPGA TIMING ATTACK TO GUESS KEYS OF INCREASING LENGTHS

Number of Bytes	Time (s)
1	0.3
2	0.6
4	0.9
8	1.8
16	3.6
32	7.25

This data shows that the amount of time our attack takes increases linearly with the length of the secret key. This makes it possible to guess much longer keys than you could with a brute force strategy. Based on the results in the Table III, we estimated that it takes about 225 milliseconds to guess all 250possible values for one byte of the key. That means it takes about about $250ms/250 = 900\mu$s for a single guessing transaction to take place between the MCU and the FPGA. If we extrapolate this information farther, we can estimate how long a brute force attack would take to guess keys of increasing length.

TABLE III
ESTIMATED TIME IT WOULD TAKE TO BRUTE FORCE KEYS OF INCREASING LENGTHS WITH OUR SETUP

Number of Bytes	Time
1	0.225 s
2	$900\,\mu\text{s} \times 250^2 = 56.25$ s
4	$900\,\mu\text{s} \times 250^4 = 40.7$ days
8	$900\,\mu\text{s} \times 250^8 = 4.41 \times 10^8$ years

IV. CONCLUSION

These simple timing attack implementations highlight the importance of constant time functions. In the real world, a non-constant implementation of encryption algorithms or security functions are at risk of timing attacks and pose a real threat to security. While making our four digit compare functions constant-time would be simple, making more complex algorithms constant-time requires careful thought and methodical techniques like bit-slicing.

ACKNOWLEDGMENT

Dr. Goeders taught us everything we know about hardware security. Joe Grand gave us the idea for the simple timing attack.

REFERENCES

[1] Joe Grand. Side channel timing attack demonstration, 2017.