An In-depth and Black-Box Characterization of the Effects of Laser Pulses on ATmega328P

Dilip S V Kumar[1], Arthur Beckers[2], Josep Balasch[2], Benedikt Gierlichs[2], and Ingrid Verbauwhede[2]

[1] SEAL, Department of Computer Science and Engineering, IIT Kharagpur, India
[2] imec-COSIC, KU Leuven
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{firstname.lastname}@esat.kuleuven.be

Abstract. Laser fault injection is a complex, physical process with many parameters that influence the success of the injection. Some parameters are difficult to control. While many works have established *that* focused laser light can inject seemingly random faults in electronic devices such as microcontrollers, FPGAs and ASICs, only few works explain precisely *why* a specific fault can be observed. We narrow this gap and characterize in detail the effects of laser pulses on an 8-bit microcontroller in a black-box fashion, with access to only public documentation. With our setup and settings we can inject faults only in the read-out circuitry for the on-chip flash memory. As result of our analysis we are able to inject bit-reset faults in individual bits of opcodes and data words stored in flash with *100%* accuracy and repeatability. This allows us to easily demonstrate well known attacks on cryptographic software, e.g. manipulation of a block cipher implementation's number of rounds. At the same time our study informs the targeted development of countermeasures.

Keywords: laser fault injection · microcontroller · black-box characterization.

1 Introduction

After the publication of the first fault attack by Boneh et al. [7] a vast body of work has been accumulated regarding fault attacks, countermeasures and fault injection methods. A fault attack aims to inject a fault during a device's nominal operation, and to analyze the faulty output, in order to retrieve secret information stored on the device. The most common fault analysis methods are Differential Fault Analysis (DFA) [5], Collision Fault Analysis (CFA) [16] and Ineffective Fault Analysis (IFA) [6,12]. Each of these attacks requires the fault introduced into the device to have certain properties, the introduced faults need to behave according to a desired fault model. To counter these fault attacks countermeasures usually rely on some form of redundancy [4] or on sensors [18].

To achieve the required fault model different fault injection methods can be employed. These fault injection methods are often classified according to their invasiveness. Common fault injection methods include voltage glitching [17], clock

The final authenticated version is available online at
https://doi.org/10.1007/978-3-030-15462-2_11.

glitching [3], electromagnetic pulses [21] and laser fault injection [24]. Of these injection methods, laser fault injection is the most invasive since it requires the exposure of the bare die. However, laser fault injection is also very precise and can target a specific area of an integrated circuit (IC) and introduce faults controlled both in time and space.

Laser fault injection is a powerful method to disrupt an IC's operation. Laser light, with the correct wavelength, can penetrate an IC's substrate and reach the transistor layer. The coherent light produced by the laser source is directed at the IC's circuitry and causes transistors to toggle. Due to the small spot size, the area illuminated by the laser light, very precise faults can be introduced into the circuit. The spot size can be in the μm range and depends on the used wavelength, the quality of the laser source and the optics. For this reason laser fault injection was used first to simulate the effect of single event upsets [15] on ICs and later to perform fault attacks on cryptographic implementations [24].

With shrinking technologies it is no longer possible to target a single transistor. Instead clusters of transistors are illuminated by the laser light, making the outcome of a laser fault harder to predict. A multitude of parameters can be varied all of which can influence the outcome of a laser fault injection.

Laser fault injection can be applied to every platform ranging from ASICs or FPGAs to microcontrollers (μCs). In this paper we will focus on laser fault attacks on μCs. Previous works have already demonstrated the vulnerability of μCs to laser fault attacks. The first laser fault attack on a μC was published by Skorobogatov et al. [24]. They managed to set bits in SRAM using a laser pointer and some optics. Later a more in-depth study into faulting the SRAM of μCs was done by Agoyan et al. [2]. Courbon et al. demonstrated that besides SRAM the registers of a μC are also vulnerable to laser fault injection [13]. Note that these results were not obtained on the μC that we use in our study.

We chose the ATmega328P as our target μC. The susceptibility of the ATmega328P to laser fault injection was already demonstrated by Breier et al. [9,10]. In their work the authors demonstrate that meaningful faults, i.e. program flow or data is corrupted but the chip does not reset, can be introduced into the ATmega328P. The region on chip where they achieved these faults corresponds to one of the regions also found sensitive to laser pulses in our experiments. For our experiments we use a setup similar to the one used by Breier et al.

In their experiments they achieved three types of faults: instruction skip faults, register disturbance faults and address change faults. Under register disturbance faults they understand faults that influence the values stored in registers. Address change faults are faults where data from a different register than intended is written to memory. Breier et al. describe the type of faults they observe but give no explanation for the underlying mechanism.

Contribution We characterize in detail the effects of laser fault injection on the 8-bit μC ATmega328P while it reads from program memory. This includes the pre-loading of operations in the fetch stage and the loading of data in the execute stage. We not only locate the regions sensitive to laser fault injection,

The final authenticated version is available online at
https://doi.org/10.1007/978-3-030-15462-2_11.

but also describe the physical effects causing the fault model. The different fault types observed by Breier et al. can be unified under the fault model resulting from our characterization. As result of our analysis we are able to inject bit-reset faults in individual bits of opcodes and data words stored in flash with *100%* accuracy and repeatabiltiy.

2 Experimental design

In this section we describe our experimental setup for fault injection as well as our methodology to characterize and understand the injected fault.

2.1 Target device

In this work we target an 8-bit AVR microcontroller, the Atmel ATmega328P[19]. It has 32kB of flash memory, 1kB of EEPROM and 2kB of RAM. It operates on a two stage pipeline with fetch and execution stages: the next instruction is fetched during execution of the current instruction. In our experiments the target device (DUT) runs at 4 MHz. The ATmega328P was chosen for several reasons. The fault sensitivity of the μC has already been demonstrated, but not described in depth, in previous works. The ATmega328P is also used in the Arduino UNO platform, making it easy to program and interface with the target device. Another important advantage of the ATmega328P is its availability in DIP packages. This allows to easily decapsulate the target device and should the device be damaged by the laser fault injection it can easily be replaced.

Laser fault injection is a semi-invasive attack where the adversary needs line-of-sight to the bare die. Therefore the backside of the chip's package together with the copper paddle was removed using a cheap hobbyist mill. After removing the thermal paste with acetone the bare die is exposed. Our characterization is performed from the backside. Thus the laser first has to penetrate the chips substrate before it reaches the transistor layer. The backside of the IC is chosen as a target since the topside holds the metal routing wires. These are not transparent to light and would therefore block or hinder the laser light from reaching the transistor layer.

The DUT is mounted on a modified Arduino UNO board. Since the laser fault injection is performed through the backside of the DUT a DIP socket was soldered to the Arduino UNO PCB, such that the exposed die is pointing "upwards" towards the microscopic lens of the laser fault injection setup.

In order to better understand which part of the μC was faulted, we took a through substrate image of an ATmega328P using a near infrared (NIR) camera (Figure 1). The original substrate thickness of the chip is around 500 μm making it impossible to make a clear image due to the high absorption coefficient of Si for NIR light. Therefore the substrate of the μC was thinned down mechanically to 50 μm. After thinning a complete image of the 3 by 3 mm die was taken. No thinning was done on the chip used for our characterization experiments.

The final authenticated version is available online at
https://doi.org/10.1007/978-3-030-15462-2_11.

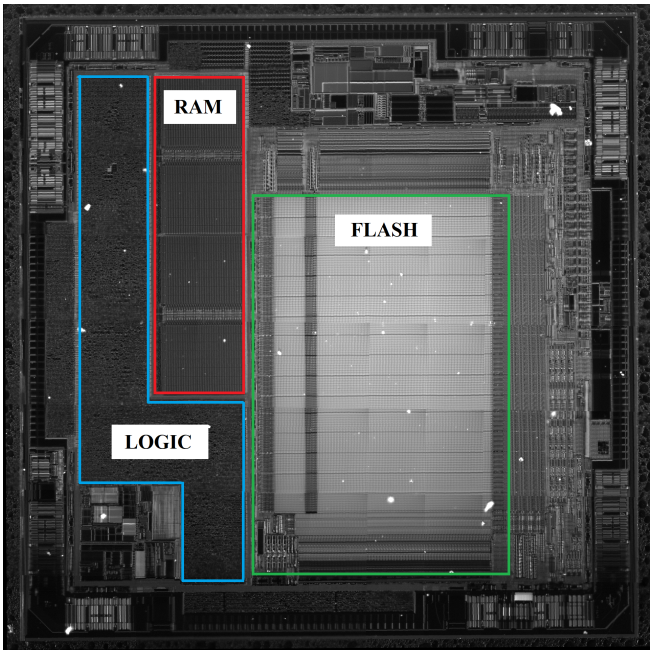


Fig. 1: Through substrate image of an ATmega328P.

2.2 Laser fault injection setup

The laser fault injection setup used to perform the characterization employs a diode pumped laser source with a 1064 nm wavelength. The laser source has a maximal output power of 2 W and a minimal pulse width of 1 ns. The laser beam is focused using a 50X microscope objective. The microscope objective is mounted on a XYZ stepper table that has a minimum step size of 0.1 μm. A NIR camera in combination with NIR LEDs is used to focus the laser source on the target μC.

With the stepper table we control the position of the laser relative to the DUT. The trigger for the laser fault injection setup is generated by the DUT on one of its output pins. Communication with the DUT is done through a serial interface. All the components of the setup (steppers, communication, etc.) are controlled with a PC.

2.3 Methdology

Our black-box analysis follows the methodology of Balasch et al. [3]. Before injecting a laser pulse, the device is reset and initialized in order to bring it to a fully known state. Under a known state we understand that all registers and memories contain the same values before every pulse injection. After resetting the device, the target code is executed, a fault is injected and the content of all

The final authenticated version is available online at
https://doi.org/10.1007/978-3-030-15462-2_11.

working registers is read back over the serial port. This process is repeated for every target location on the μC.

After collecting the data for every target location we examine the difference between the expected register values and the ones received after laser pulse injection. The goal is to reverse-engineer what instruction was executed instead of the target instruction, which in turn reveals which bits of the opcode have changed, and how. *This manual process is extremely time consuming and at the heart of our study.*

By carefully selecting the target instructions and analyzing the different faults we obtain for each of them we can get an understanding of the underlying cause for the obtained faults.

To ensure that the fault injection affects only a single instruction and hence to simplify our task, we surround the target instruction with several *nop* instructions. Our test code is shown in Table 1.

Table 1: Test code.

Cycle	Instruction
1	sbi 0X0B, 7 // trigger
2	nop
3	nop
4	target
5	nop
6	nop

2.4 Locating sensitive regions

In a first step we scanned the entire chip area for regions that are sensitive to laser fault injection. We applied laser pulses with large pulse width, twice the clock period, while executing different target instructions such as *brne* and *ldd*. We used a relatively large step size of 20 μm for these scans. During these initial scans we only checked if the program was correctly executed or not, by comparing the content of the registers after normal and faulted execution.

The initial scans revealed that only laser pulses directed at flash memory caused meaningful faults in the program's execution. This region will therefore be the focus of the characterization.

3 Faulting the flash memory

The initial scanning of the chip to find sensitive regions showed that flash memory was the only region that demonstrated meaningful faults during the program execution. Note that *no permanent faults* were introduced. The content of the flash memory cells remained the same before and after the laser fault injection.

The final authenticated version is available online at
https://doi.org/10.1007/978-3-030-15462-2_11.

This observation suggests that we inject faults in the read-out circuitry. Flash memory has different components that can be faulted. The flash found in the ATmega328P most likely has a NOR-type structure where flash memory cells can be accessed individually. Each memory cell can be addressed separately by enabling its word-line and bit-line as can be seen in Figure 2. Some control logic is required to enable the correct word-line drivers and select the correct bit-line. The content of the addressed memory cell is read out by one or several sense amplifier(s). There are also analog components to generate the high voltages needed to program the flash cells, to generate reference voltages for the sense amplifiers, etc. Each of these components might be sensitive to laser pulses. The fact that flash memory can be faulted by laser fault injection has been demonstrated by Skorobogatov [23].

The characterization is done in a black-box setting thus we can never be completely sure about the effects the laser pulses have on the targeted circuit. Our initial scan of the chip showed that the entire flash region was sensitive to laser pulses. Besides the flash memory cells themselves another region in the bottom left corner proved to be sensitive to laser pulses. Based on the image taken through the chips substrate we can observe that this is most likely an analog component.

In the next sections the effects of laser pulses on the ATmega328P's flash memory will be discussed in more detail.

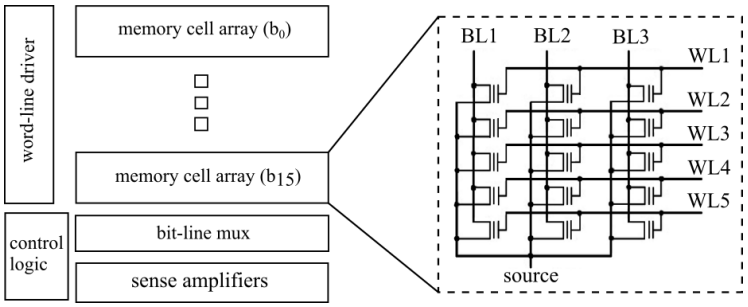


Fig. 2: Flash memory structure.

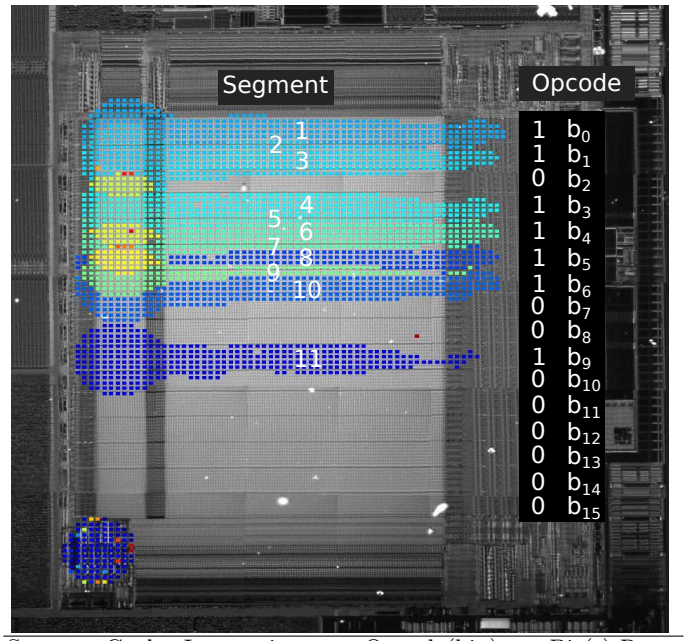
3.1 Effect of Laser Pulses on Program Flow

In order to analyze the effects of laser pulses on program flow, we set our target instruction to be *muls r23, r27* (Multiply Signed). Figure 3a illustrates the sensitive regions in flash memory that are vulnerable to laser fault injection for our target instruction. The heat map consists of 11 different fault sensitive regions, enumerated in Figure 3a, and in each of these regions, our target instruction is replaced with the execution of a different instruction. The instructions executed

The final authenticated version is available online at
https://doi.org/10.1007/978-3-030-15462-2_11.

in place of our target instruction are shown in Table 3b. The sensitive regions are colour-coded in the heat map in order to distinguish them from each other.

(a) Fault Sensitive Regions on flash memory



Segment	Cycle	Instruction	Opcode(bin)	Bit(s) Reset
	i	nop	0000 0000 0000 0000	
	i+1	muls r23, r27	0000 0010 0111 1011	
1	i+1	muls r23, r26	0000 0010 0111 1010	b_0
2	i+1	muls r23, r24	0000 0010 0111 1000	b_0 & b_1
3	i+1	muls r23, r25	0000 0010 0111 1001	b_1
4	i+1	muls r23, r19	0000 0010 0111 0011	b_3
5	i+1	muls r22, r19	0000 0010 0110 0011	b_3 & b_4
6	i+1	muls r22, r27	0000 0010 0110 1011	b_4
7	i+1	muls r20, r27	0000 0010 0100 1011	b_4 & b_5
8	i+1	muls r21, r27	0000 0010 0101 1011	b_5
9	i+1	muls r17, r27	0000 0010 0001 1011	b_5 & b_6
10	i+1	muls r19, r27	0000 0010 0011 1011	b_6
11	i+1	nop*(Invalid)*	0000 0000 0111 1011	b_9

(b) Instructions replacing the target instruction.

Fig. 3: *muls r23, r27.*

Based on the instructions that replace the target instruction in Table 3b, we could observe that the fault segments are correlated with the *opcode* of *muls r23, r27* and the *opcode* of all the erroneous instructions differ from our target

instruction by only one or two consecutive bit resets. Let us represent the bits of the *opcode* as $b_{15}b_{14}\cdots b_0$. As we traverse the memory from top to bottom in Figure 3a, we can notice a gradual advancement in the bit positions that are reset. Fault injection in Segment 1 resets the bit b_0 and in Segment 11 resets the bit b_9. And the segments that reset two bits of the *opcode* are formed from the overlap of the segments immediately above and below them. It is also interesting to note that the relative positions of the fault segments in Fig 3a resemble the bit positions in the target instruction's opcode. For instance, Segments 3 and 4 are separated from each other by a fault free-region as they represent non-adjacent bits b_1 and b_3. Note that these two bits are equal to 1 in our target instruction whereas the bit between them, b_2 is equal to zero in our target instruction. From this, we can be certain that the fault model is *bit-reset* and not *bit-flip*.

We repeated our experiments on more instructions such as *ldd*(Load Indirect from data space), *breq*(Branch if Equal),*mov*(Copy Register) etc. and observed very similar results. Figure 4 illustrates the results we obtained on targeting the instruction *ldd r11,z +0x1c*.

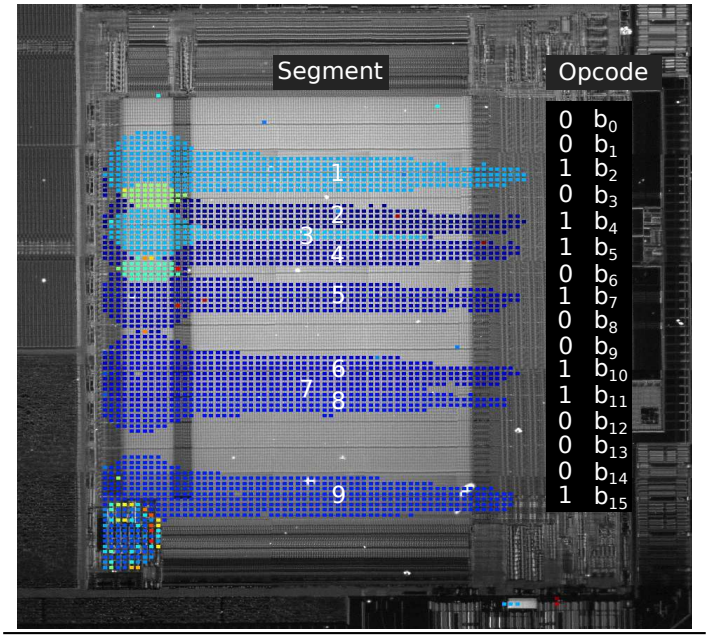
Flash memory in ATmega328P is organized as 16-bit words. Every instruction in AVR assembly language has a unique 16-bit opcode (Operation code) representing it. On programming the chip, all the instructions of our program are stored in the flash memory as 16-bit operation codes. In the pre-fetching phase, the opcode of the instruction pointed to by the Program Counter (PC) is loaded through the 16-bit Program Bus. Hence injecting a laser pulse during fetch disturbs the opcode, thereby replacing the target instruction with a different instruction.

Overlapping of fault segments does indeed create specific regions near the left margin of the memory that can even reset more than 2 bits in the instruction *opcode*. But these regions are hard to generalize for all instructions as the level of overlap would depend on many parameters such as the pulse width, laser intensity and the instruction's opcode. The size of the regions influenced by the laser pulses depends on the settings of the laser source. For our experiments we used a pulse width of 150ns and a laser power of 0.6W. Relatively high laser power is required since there was no thinning of the IC's 500 μm thick substrate. An increase in laser power or pulse width will up to a point increase the size of the sensitive regions as can be seen in Figure 5. This suggests that a certain amount of energy is required to inject a fault into flash memory. Furthermore, the regions are obviously much larger than any single flash memory cell. This supports our assumption that we are injecting faults into the read-out circuitry. More precisely, we seem to affect a part that is shared for many bits.

At the bottom-left corner of flash memory, there exist some fault sensitive regions as depicted in Figure 6. These regions form concentric rings, and in each of these regions the target instruction is always replaced with a fixed instruction irrespective of the target instruction. Based on the through substrate image we can observe that this region most likely contains analog circuitery. One possible explanation for this behaviour would be that injecting faults in these locations

The final authenticated version is available online at
https://doi.org/10.1007/978-3-030-15462-2_11.

(a) Fault Sensitive Regions on flash memory.



Segment	Cycle	Instruction	Opcode(bin)	Bit(s) Reset
	i	nop	0000 0000 0000 0000	
	i+1	ldd r11, z+0x1c	1000 1100 1011 0100	
1	i+1	ldd r11, z+0x18	1000 1100 1011 0000	b_2
2	i+1	ldd r10,z+0x1c	1000 1100 1010 0100	b_4
3	i+1	ldd r8, z+0x1c	1000 1100 1000 0100	b_4 & b_5
4	i+1	ldd r9, z+0x1c	1000 1100 1001 0100	b_5
5	i+1	ldd r3, z+0x1c	1000 1100 0011 0100	b_7
6	i+1	ldd r11, z+0x14	1000 1000 1011 0100	b_{10}
7	i+1	ldd r11, z+0x04	1000 0000 1011 0100	b_{10} & b_{11}
8	i+1	ldd r11, z+0x0c	1000 0100 1011 0100	b_{11}
9	i+1	add r11, r4	0000 1100 1011 0100	b_{15}

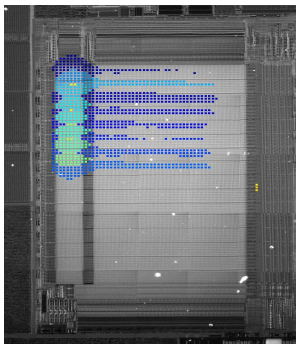
(b) Instructions replacing the target instruction.

Fig. 4: *ldd r11, z + 0x1c.*

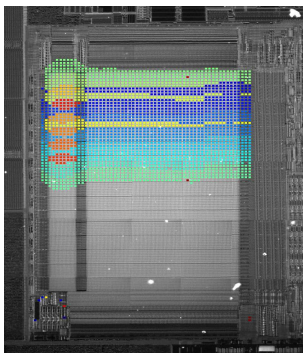
produce *stuck-at* faults during opcode fetch. The instructions executed instead of our target instruction in some of these regions are:

- nop (Opcode : 0000 0000 0000 0000)
- muls r16, r16 (Opcode : 0000 0010 0000 0000)
- and r0, r16 (Opcode : 0010 0010 0000 0000)
- or r0, r18 (Opcode : 0010 1010 0000 0010)
- or r0, r16 (Opcode : 0010 1010 0000 0000)

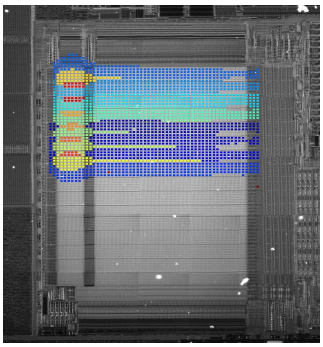
The final authenticated version is available online at
https://doi.org/10.1007/978-3-030-15462-2_11.



(a) Width = 70ns & Power = 0.6W



(b) Width = 150ns & Power = 0.6W



(c) Width = 70ns & Power = 1.2W

Fig. 5: Influence of the laser parameters on the size of the sensitive regions.

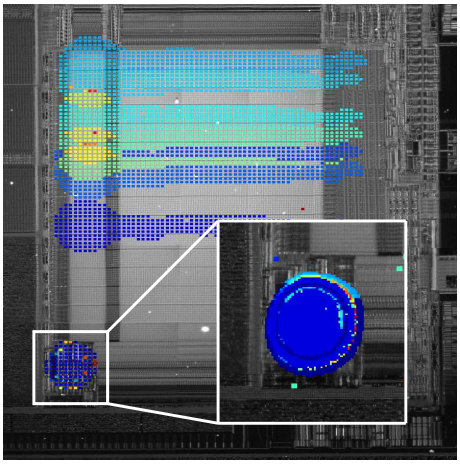
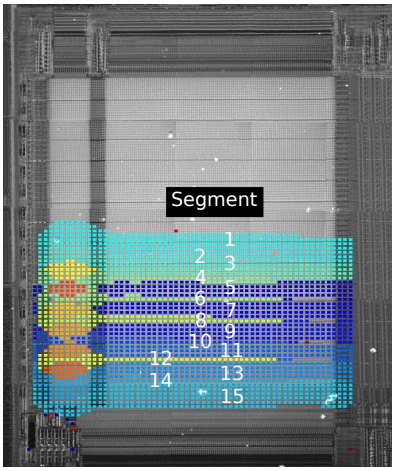


Fig. 6: Concentric Fault Sensitive Rings.

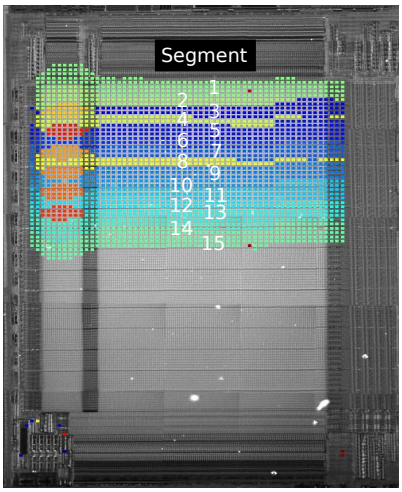
The final authenticated version is available online at
https://doi.org/10.1007/978-3-030-15462-2_11.

4 Effects of Laser pulses on data flow

From the observations we have made on the effects of laser fault injection on program flow, we can be fairly certain that fault injection does indeed disturb the 16 bits of opcode fetched from flash memory. In this section, we demonstrate how fault injection can also affect the execution stage of target instructions that read values from program memory, thereby inducing data faults.



(a) lsb(Z) = 1



(b) lsb(Z) = 0

Segment	Value *r9*	Bits Reset
	255	None
1	254	v_0
2	252	v_1 & v_0
3	253	v_1
4	249	v_2 & v_1
5	251	v_2
6	243	v_3 & v_2
7	247	v_3
8	231	v_4 & v_3
9	239	v_4
10	207	v_5 & v_4
11	223	v_5
12	159	v_6 & v_5
13	191	v_6
14	63	v_7 & v_6
15	127	v_7

(c) Data Faults in Register *r9*

Fig. 7: Faults in the data flow

As discussed, Program Memory in the DUT is organized and accessed as 16-bit words. However, the `lpm` (Load from Program Memory) instruction uses a

byte address pointer (Z) to load a single byte to the destination register. Selecting between low or high byte of the 16-bit word is most likely determined by the least significant bit of the Z pointer. This characteristic enables the following experiment. We store a table of 8-bit constants with the value 0xff in program memory. We use the command `lpm r9,z` as our test instruction, to load an element from the table. The constants being 8-bit wide, two adjacent elements of the table are stored together as a 16-bit word in Program Memory. To verify this assumption, we load two constants stored at consecutive memory locations using `lpm` instructions. If our assumption is right, one constant should be a low byte and the other a high byte. We perform a step-wise scan throughout flash memory and monitor if the value getting stored in Register *r9* matches the expected value (0xff). Figure 7b and 7a show the heat maps for the cases when the least significant bit of Z is 0 and 1 respectively. The fault pattern in both figures 7b and 7a are identical, as depicted in Table 7c, indicating every bit of the 16-bit word can be reset. In other words, there exists a dedicated region on flash memory that can be targeted to reset a particular bit of the 16-bit word.

Additionally one can notice that fault sensitive regions for both cases are present in only one half of flash memory. This observation implies that the low byte of the 16-bit word from which the constant 0xff was fetched is present in the top half of the flash memory and the high byte in the bottom half. We can now conclude that the bits of flash memory word are arranged in flash memory cells in a vertical arrangement, with the least significant bit placed near the top margin of flash memory and the most significant bit near the bottom margin. The fault sensitive region for a certain bit of a target instruction or data, e.g. the least significant bit, comprises that bit of all words in flash memory, e.g. all LSBs. This suggests that (part of) the read-out logic is shared among all bits that have the same position in a word.

5 Application

In this section, with the knowledge we have acquired so far, we demonstrate a couple of threats laser fault injection could pose to an implementation of AES on our platform. Figure 8a illustrates a snippet of AES-128 encryption, where `_rcon` indicates the round counter. Now suppose we target the Branch If Not Equal (brne) instruction in this snippet and inject a laser fault in the location shown in the Figure 8, highlighted in red. The opcode bit b_{10} of `brne RoundFunction` would be reset changing the instruction type of the target instruction from Branch if Not Equal (`brne`)(Opcode : 1111 01kk kkkk k001) to Branch if Equal (`breq`)(Opcode : 1111 00kk kkkk k001). If `brne` were to be replaced with `breq` in the encryption algorithm, it would prevent the control from branching and would drastically reduce the number of rounds executed during encryption, thereby enabling key-recovery through simple cryptanalytic attacks [11].

Additionally, look-up tables such as the S-box would likely be stored as constants in flash memory and frequently accessed during encryption (resp. decryp-

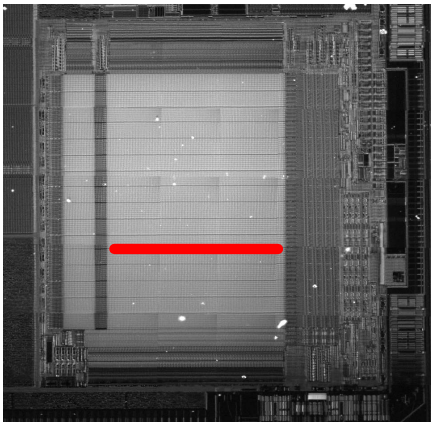
The final authenticated version is available online at
https://doi.org/10.1007/978-3-030-15462-2_11.

tion). As this would require the usage of the *lpm* (Load from Program Memory) instruction, an adversary could target this instruction in order to inject faults in the data flow. This would allow to mount classical DFA attacks such as Piret and Quisquater [20].

When countermeasures or tools [8] are developed to defend against fault attacks, particularly DFA, the assumption is often that only the operands are vulnerable to fault injection. Or if an attacker manages to fault an instruction it will only lead to an instruction skip. These assumptions might hold for most devices, but as demonstrated in this paper, they do not hold for all of them. Therefore, when developing countermeasures, it is safer to assume a fault model where an attacker is able to manipulate the opcode bits up to a certain degree.

```
...
RoundFunction:
...
inc _rcon
cmp _rcon, 0x0a
brne RoundFunction
...
```

(a) AES encryption snippet



(b) Fault Injection Location

Fig. 8: Faulting AES round counter.

6 Conclusion

In this work we described the effects of laser pulses on the flash memory of an 8-bit microcontroller, the ATmega328P. We target the read out of data from flash memory and are able to reset one or multiple bits of the fetched data or opcode to zero. We have full control over which of the 16 bits is reset. Being able to reset arbitrary bits in the opcode is a very powerful fault model which allows us to perform any of the many well known published attacks. For instance, the opcode of instructions can be faulted such that the program flow is altered, or data read from memory is faulted in order to enable a classical DFA attack. The acquired fault model is supported by the physical layout of the flash memory.

Understanding the underlying cause for a fault model does not only allow us to mount powerful attacks, it also helps us in understanding which countermeasures need to be taken in order prevent these attacks. In order to detect faults

in the flash memory, error detecting codes that detect at least two bit resets can for instance be added. However, to detect the faults that we observed in our experiments, it may not be necessary to protect the memory itself, but only the readout circuitry.

Acknowledgment

This work was supported in part by the Research Council KU Leuven: C16/15/058 and through the Horizon 2020 research and innovation programme under Cathedral ERC Advanced Grant 695305.

References

1. M. Agoyan, J. Dutertre, A. Mirbaha, D. Naccache, A. Ribotta, and A. Tria. How to flip a bit? In *IOLTS 2010*, pages 235–239. IEEE Computer Society, 2010.
2. M. Agoyan, J. Dutertre, D. Naccache, B. Robisson, and A. Tria. When clocks fail: On critical paths and clock faults. In D. Gollmann, J. Lanet, and J. Iguchi-Cartigny, editors, *CARDIS 2010*, volume 6035 of *LNCS*, pages 182–193. Springer, 2010.
3. J. Balasch, B. Gierlichs, and I. Verbauwhede. An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs. In L. Breveglieri, S. Guilley, I. Koren, D. Naccache, and J. Takahashi, editors, *FDTC 2011*, pages 105–114. IEEE Computer Society, 2011.
4. H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, Feb 2006.
5. E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In K. S. Burton, editor, *Advances in Cryptology — CRYPTO '97*, pages 513–525, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.
6. J. Blömer and J.-P. Seifert. Fault based cryptanalysis of the advanced encryption standard (aes). In R. N. Wright, editor, *Financial Cryptography*, pages 162–181, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
7. D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In W. Fumy, editor, *EUROCRYPT '97*, volume 1233 of *LNCS*, pages 37–51. Springer, 1997.
8. J. Breier, X. Hou, and Y. Liu. Fault attacks made easy: Differential fault analysis automation on assembly code. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(2):96–122, 2018.
9. J. Breier and D. Jap. Testing feasibility of back-side laser fault injection on a microcontroller. In *Proceedings of the WESS'15: Workshop on Embedded Systems Security*, WESS'15, pages 5:1–5:6, New York, NY, USA, 2015. ACM.
10. J. Breier, D. Jap, and C.-N. Chen. Laser profiling for the back-side fault attacks: With a practical laser skip instruction attack on aes. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, CPSS '15, pages 99–103, New York, NY, USA, 2015. ACM.
11. H. Choukri and M. Tunstall. Round reduction using faults. In *2005 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 13–24, 2005.

The final authenticated version is available online at
https://doi.org/10.1007/978-3-030-15462-2_11.

12. C. Clavier. Secret external encodings do not prevent transient fault analysis. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 181–194, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
13. F. Courbon, P. Loubet-Moundi, J. Fournier, and A. Tria. Adjusting laser injections for fully controlled faults. In E. Prouff, editor, *Constructive Side-Channel Analysis and Secure Design*, pages 229–242, Cham, 2014. Springer International Publishing.
14. J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.
15. D. H. Habing. The use of lasers to simulate radiation-induced transients in semiconductor devices and circuits. *IEEE Transactions on Nuclear Science*, 12(5):91–100, Oct 1965.
16. L. Hemme. A differential fault attack against early rounds of (triple-)des. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, pages 254–267, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
17. C. H. Kim and J. Quisquater. Fault attacks for CRT based RSA: new attacks, new results, and new countermeasures. In D. Sauveron, C. Markantonakis, A. Bilas, and J. Quisquater, editors, *Information Security Theory and Practices - WISTP 2007*, volume 4462 of *Lecture Notes in Computer Science*, pages 215–228. Springer, 2007.
18. K. Matsuda, T. Fujii, N. Shoji, T. Sugawara, K. Sakiyama, Y. Hayashi, M. Nagata, and N. Miura. A 286f^2/cell distributed bulk-current sensor and secure flush code eraser against laser fault injection attack. In *International Solid-State Circuits Conference - ISSCC 2018*, pages 352–354. IEEE, 2018.
19. Microchip. Atmega328p. `https://www.microchip.com/wwwproducts/en/ATmega328P`, July 2018.
20. G. Piret and J. Quisquater. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2003.
21. J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In I. Attali and T. P. Jensen, editors, *Research in Smart Cards - E-smart 2001*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.
22. C. Roscian, A. Sarafianos, J. M. Dutertre, and A. Tria. Fault model analysis of laser-induced faults in sram memory cells. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 89–98, Aug 2013.
23. S. Skorobogatov. Flash memory 'bumping' attacks. In S. Mangard and F. Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 158–172, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
24. S. P. Skorobogatov and R. J. Anderson. Optical fault induction attacks. In B. S. K. Jr., Ç. K. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 2–12. Springer, 2002.
25. J. G. J. van Woudenberg, M. F. Witteman, and F. Menarini. Practical optical fault injection on secure microcontrollers. In L. Breveglieri, S. Guilley, I. Koren, D. Naccache, and J. Takahashi, editors, *FDTC 2011*, pages 91–99. IEEE Computer Society, 2011.