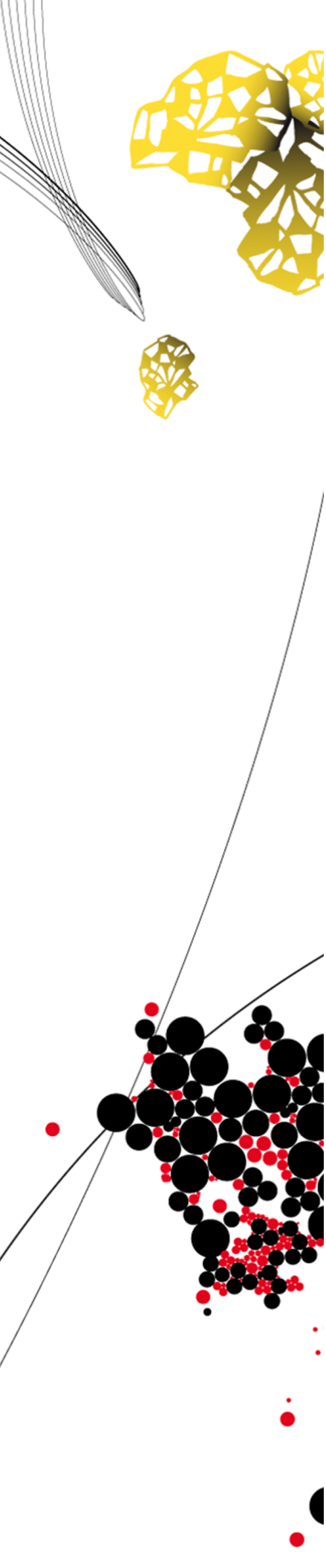


UNIVERSITY OF TWENTE.

Department of Computer Science

Dual Laser Fault Injection Attack

Yannis Koukoulis

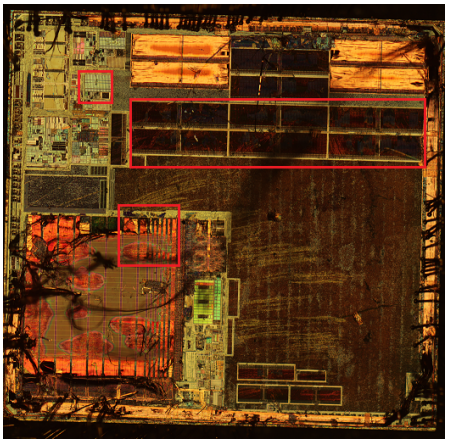
M.Sc. Thesis

in fulfillment of the requirements
of the EIT Digital Security & Privacy Master

September 2016

Supervisors:
Dr. A. Peter.
Dr. F. de Beer

Services, Cybersecurity and Safety Group
Department of Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands



Thesis advisor: Dr. Andreas Peter Yannis Koukoulis

Dual Laser Fault Injection Attack

Abstract

This thesis describes the development and demonstration of a Laser Fault Injection attack on a commercial, programmable 32-bit architecture 90nm technology target with a microcontroller and dedicated hardware peripherals. More precisely, we perform a dual, or second-order, laser fault injection. That is attacking the target at two different locations simultaneously, for the purpose of validating fault tolerant design and performance. The first laser aims to neutralize the security function, while the second precisely injects a fault into the AES encryption, resulting in a faulty ciphertext.

Hardware vendors must assume that the attacker is highly skilled, equipped with advanced tools and has abundant resources. We attack many different components both front- and back-side to illustrate that just one countermeasure is not sufficient, rather a combination is required for fault tolerant design.To the best of our knowledge such an attack is only performed once, concurrently with the present, however based on an FPGA target opened from the backside. In this thesis we aim at a fairly different scenario, on a commercial target, not only of hardware components of different type, but also of a large spatial distance between them.

We show how laser fine tuning has been used to characterize the vulnerable spots, and to subsequently inject faults. Moreover, we devise a reproducible and transferable approach of attacking commercial hardware with a detection countermeasure implemented. With the advent of multi-processor chips, embedded industry leans towards distribution of cores, peripherals and computation. We show that for security critical applications, relying on hardware distribution as a countermeasure is not sufficient.

Contents

List of Figures

Dedicated to my father.

Acknowledgments

I would like to express my deep gratitude to Dr. A. Peter, my research supervisor, for his patient guidance, enthusiastic support and precious critiques of this research work.
I would also like to offer my special thanks to Dr. F. de Beer from Riscure, for steadily facilitating my research and experimenting during the past seven months. My special thanks are also extended to Mr. Carpi, for his help setting-up and troubleshooting my experiments, Mr. Zhang for taking the time explaining the fundamental architectural concepts of Inspector software, and his useful feedback during building the Twinscan module, Mr. Timmers, and Mr. Spruyt for sharing their experience in fault injection concepts and techniques, as well as their warm support.
Finally, I would like to thank my father for his support and encouragement throughout my research.

1

Introduction

1.1 Motivation

Implementations of cryptographic algorithms continue to proliferate. The emergence of the ”Internet of Things” concurs to that. Hardware Security is a branch of IT security, leveraging models such as Fault Injection and Side-Channel analysis. Potential targets of those methods are, Smart Cards, Automated Teller Machines, Industrial Control Systems, video game consoles and Set-Top Boxes. This list is demonstrative but not exhaustive, for attackers are quite creative in identifying security shortcomings and potential targets. In the bank & finances sector - perhaps the most crucial from implications point of view domain - security has transmuted from armed forces to TLS and Public Key Cryptography. In case a smart card can be practicably

breached, or an ATM trivially tapped, our savings are in jeopardy and our established ”status quo” in peril. This is only one side of the coin. Imagine what can happen if state, or governmental secrets, that are secured with an encryption scheme, fall into the wrong hands.

Those are all valid reasons for us to want to secure our infrastructure. Cybercriminals are however poised to wreck these plans and wreak havoc. Defenders and IT security officers are trying to thwart the attacks and shelter the systems, in what is called a cat and mouse game [5]. In this game hardware attacks were devised and are successfully deployed. These attacks are becoming more practical due the abundance of highly-trained personnel, the advancement of dedicated tools and ever more sophisticated theoretical attacks. Consequently, there are numerous ways for attacking the hardware infrastructure.

In a holistic approach hardware attacks can be classified into non-invasive, semi-invasive, and invasive, regarding the level of physical modification they impinge upon the target (see [7] and [23]). The two predominant domains of hardware attacks are [6]:

Side-channel Attacks Timing analysis, Power analysis, Electromagnetic attacks. They are non-invasive.

Fault attacks Voltage, and Clock glitching, Laser Fault injection, Microprobing, Physical tampering. They range from semi-invasive to invasive attacks, hence include tampering.

Fault injection is being extensively used not only for evaluating the security of embedded systems and portable hardware, but also for breaching into systems. Two hyped examples of fault injection attacks are presented, in order to highlight the impact of a breach and the necessity for countermeasures.

The nicknamed ”Reset Glitch Hack” is attacking the XBOX 360. It was released by a French developer. ”`Memcmp` is often used to check the next bootloader's SHA hash against a stored one, allowing it to run if they are the same. Effectively we can put a bootloader that would fail hash check in NAND (i.e. memory), glitch the check and that bootloader will run, allowing almost any code to run.” A reset pulse is send to the processor on the right timing, neutralizing the hash-check, hence the injected bootloader will execute in a seamless fashion. Thereby the secure-boot chain is defeated[12].

Another attack against PS3 console was achieved, also know as the ”glitching attack”. This hardware attack involves sending a carefully-timed voltage pulse in order to compromise the Hashed Page Table and subsequently get read/write access to the main segment. The HPT performs the integrity check of the loaded memory, and maps all memory including the hypervisor. Hypervisor is the module that supervises the initial memory read and write procedures. Hence if we precisely glitch the hypervisor to desynchronize from the actual state of RAM, we enable arbitrary write access to the active HPT, and thus control access

to any memory region. The glitch is a 40 ns voltage pulse, roughly 100 clock cycles, carefully-timed. An FPGA was used to capture the correct timing and send the glitch pulse. Finally we can either inject and arbitrarily execute our exploit or dump the binary in order to examine for software vulnerabilities [14].

The aforementioned examples are not the only techniques that are deployed in hardware attacks. An overview of the known fault injection techniques is presented in table 1.1.

Techniques	Effects
Varying the supply voltage	Misinterpret or skip instruction
Glitching the external clock	Data misread, Instruction miss
Varying the temperature	Randomization of RAM cells, Glitching write or read operations
Shooting with white light	Fault injection
Shooting with Laser	Fault injection (higher precision)
Shooting with X-rays and ion beams	Fault injection (depackaging is not required)

Table 1.1: Notable techniques used in non-invasive and semi-invasive attacks

1.2 Background

This thesis is occupied with fault injections performed with lasers, in fact two of them. The necessity thereof will become apparent after the background retrospective, and especially the countermeasures introduced by the industry, and are described in the next section.

The fault injection legacy is instantiated by an accident. In the 60's it was observed that charged particles from the packaging of devices, usually compounded from Uranium isotops, were radiating, thus flipping logic bits. In the seminal paper [10] from 1965, Habing illustrates high levels of ionization can be created in semiconductor devices by irradiating the devices with short pulses of light. Thus, proving that controlled fault injection is feasible. It is shown furthermore that a pulsed-infrared laser can be used as a relatively simple, inexpensive, and effective means of simulating the effects caused by intense gamma ray sources on semiconductors. Thus, imitating the cosmic radiation. In-between new fault models for key extraction are devised. An overview of notable fault models is presented in section 1.5. Also, back-side laser fault-injections emerge as an alternative and more successful way to breach into an embedded systems, such as in [11]. Next, ever more precise and with higher success rates fault injection are illustrated, whereas technologies descend below the µm threshold, as in [21, 7, 18]. As of 2010, Trichina and Korkikyan introduce the

so called ”Multi-glitching” attack. This involves for the first time multiple glitches, albeit they use one laser, as the glitch is injected on the same spot. Finally, in [22] a simultaneous laser fault injection on different spots is achieved. As we will see in the next chapter this technique is used to bypass hardware duplication, and the authors manage to inject identical faults in two iterations of AES.
This research was in fact conducted at the same time with this thesis, nevertheless there are some differences compared to our research. First and foremost they attack an FPGA, whereby they configure the state registers in a-priori known positions. Moreover their target offers an easy back-side access, where this is not always feasible in commercial targets. Even more their attacked spots are very close to each other, whereas our setup attacks two spots significantly more distant. In fact, there is a trade-off between this distance and the spot size. Last but not least, while they bypass hardware duplication, we aim at hardware distribution.

Logical faults are getting scarce, whereas fault injection is becoming more relevant. There are a number of trends in the industry that increase the susceptibility of circuits to either external, or internal perturbations. These are the increasing circuit density, reduced operating and threshold voltage, increasing clock rates[20]. As inferred, in the hardware domain physical access to the target is the first premise for such an attack. Thereupon attackers with $1K voltage and clock glitching equipment, or $100K laser can compromise the devices by injecting a fault[26]. Laser fault injection gives extra a degree of spatial freedom. A successful breach may require a laser fault injection into the cryptoalgorithm, hence subverting the ciphertext, or an instruction skip, or a combination of both.

Furthermore, cryptographic algorithms are ubiquitous in mobile and embedded applications. Those cryptographic algorithms are executed either on a microcontroller or, as of late, in dedicated hardware (cryptoaccelerator). It is evident that the latter can be perceived as a countermeasure from the perspective of the attacker, or the security analyst, due to the spatial distance of the targeted spots.
The present thesis hereby performs a second-order attack in order to achieve a breach of the new sheltered systems. This is further delineated in the next section where the countermeasure's design is explained.

1.3 Countermeasures

Hardware manufacturers and software developers are consistently trying to thwart attacks, thereby implementing security functions. In hardware security, sensors were one of the first efforts of the industry to detect these. Glitching attacks, such as voltage and clock glitching, can be thereby countered, as an

erratic variation of these parameters can be easily detected. Laser fault injection whatsoever is not easy to counter not only because the chip is hard to be comprehensively covered with sensors, but also because a sufficiently small spot size can go through them undetected. Therefore, merely relying on sensors for security is not a sufficient countermeasure and for that reason redundancy is introduced [4, 15]. Next, a holistic presentation of redundancy countermeasures [22].

Detection-countermeasures compare the obtained results and retain the output if a discrepancy is detected.

Infection countermeasures try to transform the output in such a way, that it cannot be used anymore to conduct an attack.

Redundancy ranges from error detection codes, to repetition of encryption steps and hardware duplication. The countermeasure we designed is a combination of repetition of encryption steps and hardware distribution. The software part takes care of the double instantiation of the crypto-core and the comparison of the outputted ciphertexts. The output is either displayed, or destroyed depending on whether the two resulted ciphertexts are identical, or not respectively. The hardware part of the countermeasure, that substantiates the term **hardware distribution**, constitute the two cores that participate in the process and are located at different positions on the die. Hence, the term hardware distribution is self-explanatory, and the need for two effectively simultaneous faults is evident.
Two lasers can satisfy the need for two simultaneous fault injections on different areas, thereby circumventing the aforementioned security mechanisms. In fact one laser is used to inject the fault, while the second laser neutralizes the security function at the same time. The contribution that stems from the aforementioned is discussed in section 1.6

1.4 Use-cases

This section describes the use-cases we research, whereby two almost simultaneous glitches are required. Both involve except for the main core a second target (spot) on the die. The first scenario involves the AES crypto core, whereas the second one the RAM blocks1.2.

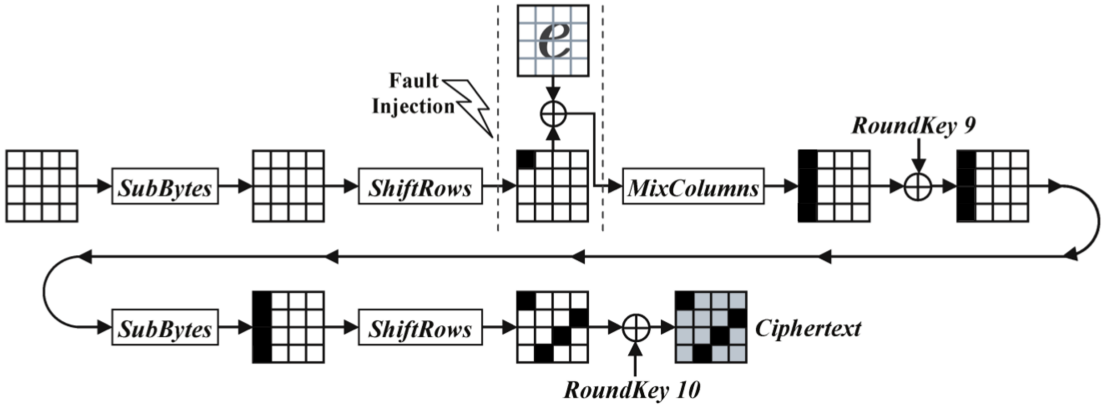


Figure 1.1: A fault injection and propagation in the last 2 AES rounds.

AES

In our experiments we are encrypting with AES-128, in Electronic Codebook, ECB mode. ECB indicates that the message is divided into blocks, and each block is encrypted separately. The key in that case has a length of 128 bits, the same as the block size, and the number of rounds is 10. According to the NIST AES standard [17], an AES round (apart form the 10^{th}), consists of the Subbytes(), ShiftRows(), MixColumns(), and AddRoundKey() transformations. By injecting a **single-byte fault** during the 9^{th} round, after propagating 4 bytes are finally subverted in the result. This in 4 byte modified result is called faulty ciphertext. Depending on the fault model implemented we need a minimum number of such faulty ciphertexts for optimum results, as well as the correct ciphertext (see next section for more details on fault models). Figure 1.1* illustrates a 9^{th} round fault injection and the propagation thereof, after each transformation. This expected pattern is also used in order to test whether our results are in accordance with it. Targets with a dedicated crypto core can be found easily and cheaply on the market. The CPU instructs the crypto core to compute encryption over the input two times. Subsequently it compares the two output and checks whether the result is the same, and retains the result in case they are not identical. The motivation is to bypass the countermeasure. Injecting the same precise fault in the AES hardware crypto core has been proven as of late by Selmke et al.. Our approach is to use two lasers, one for injecting the fault on top of the crypto core, and a second one on top of the CPU in order to neutralize the comparison.

Table 1.2: Targeted modules

*This scheme is an amended version, and the original can be found in [16]

Figure 1.2: The Bootloader

Components	Difficulty
HW AES	+++
CPU	+
RAM	+++

Secure-boot

Secure boot is a security standard developed by the pc industry to ensure that the pc only loads modules trusted by the pc manufacturer. The firmware checks the signature of the bootloader, and the firmware drivers. If these pass the test, the system either loads the operating system directly, or passes the authority to the next module in the secure-boot chain. Almost all recent pc come with UEFI that has the Microsoft certificate stored, where only Microsoft signed software is allowed to be loaded if the feature is enabled. Everyone buying a pc will end up being secured by the Microsoft-driven secure boot feature. Most Intel and ARM PCs are implementing it.
In that scenario we want to bypass the signature verification and at the same time inject a fault, for instance forge an opcode, in a loaded on RAM module.

1.5 Fault models

This section describes and summarizes notable faults models. A fault model is a proven method of key calculation based on faulty ciphertexts, as described earlier. Each fault model defines the flexibility of the fault bit, or byte, the precise timing, or round-stage of the encryption scheme the fault should be injected, and the amount of faulty ciphertexts it requires. Giraud has described two fault models to retrieve AES Key[8]. In another paper Giraud summarizes notable fault attacks against the most popular cryptographic

algorithms[9].

Dhiman Saha et.al provide a comprehensive survey of fault attacks against AES, and they introduce a new

	Fault Model	Fault location	Minimum number of required faulty results
DES	Byte (could be more)	Anywhere among the last 6 rounds	2
AES	Byte	Anywhere between the MixColumns of the 7^{th} and 8^{th} round	2
RSA-CRT	Size of the modulus	Anywhere during the computation of one of the CRT Components	1
DSA	Bit	Anywhere among 20 bytes	160
EC-DSA	Bit	Anywhere among 20 bytes	160

multi byte attack in [19].

1.6 Contribution

This thesis gives an overview of the current state-of-the-art in laser fault injection attacks, and introduces a new attack vector on embedded systems. This vector is the second laser that is leveraged to neutralize the countermeasure, while the first one injects the fault in the ciphertext. We have mentioned already that industry has devised detection countermeasures. Such a countermeasure we attack. Moreover, this countermeasure is combined with hardware distribution, a term that we introduce hereby, to indicate the two different cores that participate in the process. More precisely, the crypto hardware core performs the encryption, whereas the main core implements the comparison (countermeasure).

So far very few have attacked similar scenarios. Trichina and Korkikyan inject two faults in the same core at two different timings using one laser, albeit their technique and setup is incapable of attacking the system that this thesis does, however it can be perceived as a predecessor of the dual laser attacks. Recently, Selmke et al. (2016) attacked two different cores simultaneously, however, as we mentioned in section 1.2, their attack is performed on an FPGA where they design the logic of the board and position the state registers in ”a-priory” known vulnerable positions at a close vicinity. We, on the contrary, attack a commercial

target, hence, we do not know the vulnerable positions. We describe the approach we follow to nail them down. Moreover our targeted positions are more distant, as in most realistic scenarios, which imposes certain limitations. We describe the trade-off between the distance of the cores and the spot size.
Our results show that in security critical applications, hardware distribution in combination with a detection countermeasure do not constitute a sufficient countermeasure. Furthermore, we show that backside attacks have more chances of success. Even if the way from the back seems blocked, we show that it is worth it to sacrifice some functionality in order to achieve better success rate. Finally, we follow a holistic approach attacking all the peripherals registers that communicate with the core, and we draw interesting conclusions for the pattern of errors that are observed, depending on the module we target. Finally, we present a systematic and methodological approach on capturing the faults, characterizing the target and implementing a countermeasure, which is furthermore transferable and reproducible.
This thesis is structured in the following way:

- Chapter 2 describes the two laser setup, as well as the device under test (target) and every tool, or technique we used for setting up our experiments.
- Chapter 3 follows the procedure for characterizing and identifying a vulnerable spot on the CPU, and performing an instruction skip.
- Chapter 4 presents the approach on attacking hardware peripherals with the purpose of injecting a carefully-timed meaningful fault. The modules that are attacked are the cryptoengine, the Hash hardware peripheral and the SRAM blocks.
- Chapter 5 combines the previous findings in order to showcase a simultaneous fault injection on two spatially apart positions. We, furthermore, describe the differences of the Twinscan setup, as well as the added tools.
- Chapter 6 we summarize the results of our experiments, the feasibility of the attack in practice, as well as further research desiderata.

2

Setup

In this chapter we describe the setup, including the tools utilized, and the concepts involved in the subsequent experiments. Furthermore, we describe the preliminary work we performed on the target, and the physical background is summed up. The schematic overview of the setup is presented in figure 2.4.

2.1 Experiments' components

The main components of the setup are presented:

VC Glitcher is the heart of the Laser Fault Injection attack. It is an FPGA based glitch generation and control device which is able to create configurable glitch patterns. As its name betrays its fundamental purpose is to inject voltage and clock faults. Nevertheless, it has an attribute that is leveraged in our

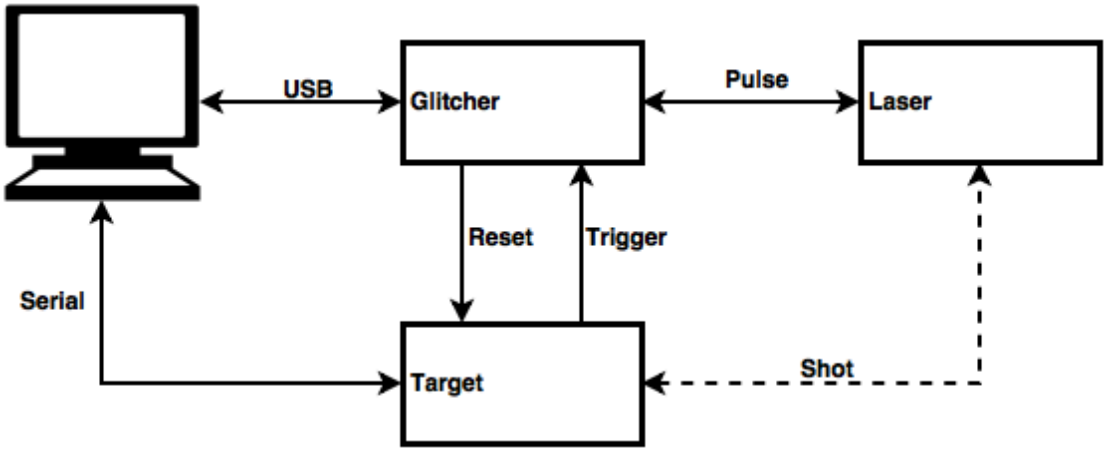


Figure 2.1: Schematic overview of the test setup

laser fault injection setup. This is outputting a pulse instead of a direct voltage, or clock glitch, at the same configurable time offset in their place. For the Voltage-Clock glitch case, a Smartcard is inserted into the designated slot, integral to the device, where its pads are in contact with the VC glitchers' pins, and thereby the glitch is communicated. If it is a SOC it is connected via USB to a smartcard replica, which is leveraged to accommodate the communication. In the latter more relevant case, the pulse is sent to the lasers, which in turn shoot, as instructed. The SOC, or the smartcard are fixed under the gun (see 2.3).

The time offset is considered towards a reference point in time. The reference point can be either the trigger that we sent to the VC glitcher's namesake dedicated input, or the reset of the target. The offset of the process we intend to attack can be identified in three possible ways.

- We have control over the binary running on the device under test. This case is popular in research, and when we attack a commercial target and we use a similar target of our control to make characterization of the device. In this case we pull-up the output of the pin whenever fits our purposes and we wire it to the aforementioned trigger input.
- We have no control over the binary. We can set the VC glitcher to interpret the reset of the device under test as trigger. Therefore all the offsets must be calculated from the targets' reset. The offsets in this case are large and non-intuitive.
- The above complexity is rounded with the aid of another dedicated tool, called icWaves. IcWaves

can extend the triggering functionality with a concept called pattern recognition triggering. This FPGA-based device generates a trigger pulse after real-time detection of a pattern. The pattern is usually obtained by Side Channel Analysis (SCA) techniques, applied on the target. These can be power consumption, or electromagnetic emission measurements (see section 4.1). Nevertheless, icWaves and pattern recognition will not be employed for this thesis, for we control the firmware of the target, thus the trigger.

Finally, the VC glitcher handles the reset of the target. There are cases where injected faults corrupt the target, and as a consequence they invalidate the subsequent iterations of the experiment. Thus, resetting is required. The VC glitcher has a dedicated input, named ”reset”, and after the process is finished it instructs the target to reset, before the next iteration begins. The target also has a pin, that when set, it reboots. To instruct for a reboot after every iteration, we set the appropriate checkbox in the software. The software is called inspector, and its purpose is explained in the next paragraph.

Inspector®* If the VC glitcher is the heart of the setup, Inspector is the brains. Inspector software is the interface between the user and the hardware. It collects the results and provides for the configuration of the glitch's (perturbation) parameters. These are in turn communicated to the hardware. The parameters are

- the number of measurements per spot,
- the pulse's length,
- the pulse's offset from the trigger,
- the pulse's power,
- areas', or spots' coordinates.

. Inspector also handles the application of implemented attack modules on traces taken. Such attacks are SCA attacks, or the relevant ”key retrieval from faulty ciphertexts” attack. It, moreover, drives the motorized device, (or the mirroring system in Twinscan's case). We define the area to be scanned by setting the three points, namely the northeast, northwest, and southwest points. A ”glitch test” function that triggers the laser, in combination with a camera help us intuitively define and visualize the targeted spots, based on beam's reflection on the die. The spots are represented by coordinates on an X-Y plane, with the NW point being the (0,0) spot. Having set the extreme spots we have defined a area. This are can be divided to a number of steps in each direction. The reader can

think of it as filling the area with a lattice of vertical and horizontal lines, where each intersection indicates an attacked spot. To move the target in order to target each of the aforementioned spots we instruct the motorized device to perform that (in the next section we describe the functionality of the motorized device). Finally, for every spot the results are collected and the target is moved to the next spot. The results taken from each spot are entries in a database and they include the coordinates. Thereby, we can navigate back to this spot if needed.

Motorized device and Mirroring System We clamp the device under test onto the motorized device. Our motorized device has a 3 axis translation table, with an X-Y positioning accuracy of 0.1µm. As mentioned earlier we can define the area to scan, by setting the three extreme points. The motorized device will take care of moving the target. In Twinscan's case, however, a double mirroring system, that drives the two laser beams, is used instead. The laser beams are driven via the same objective (refer to next paragraph) and these move independently to the motorized device. Each of the beams has its own table of coordinates which are not comprehensible to humans. Whatsoever, with the aid of a camera, we can "translate" these coordinates to actual positions over the die. The mirroring system consists of 4 mirrors, that constitute two X-Y grids, one for each laser. These grids have also precision of 0.1um. The Y coordinate which is associated with the focusing of the beams on the target can only be changed by moving the motorized board. The setup is calibrated is such a way that when the camera is focused on the dies, then the beams are also focused on the same plane. A dedicated controller and software are responsible for interpreting the move commands that we define in Inspector and communicate them to the mirrors. This interface was implemented as a part of this thesis. It is important for the reader to digest that whereas in the one-laser setup we move the board, thus the target relatively to the laser, in the Twinscan case the laser beams are moved while the target remains fixed.

Laser Station The laser station accommodates the lasers, and the rest of the devices, equipment as well as the cable-ware. This is necessary cause our lasers are classified, in accordance to the IEC 60825-1 standard, as class IV (above 500mW output power), which denotes that direct, as well as scattered radiation can cause severe or permanent damage to the skin, or eyes, without any magnification. In our experiments we leverage a blue laser (445nm, 3W), with 1*1.4um spot size, which offers precision in hunting down vulnerable areas, and a red laser (808nm, 14W), with a 6*1.5um spot size, that offers more power. These two are utilized in the front-side attacks scenario. For the backside attack we

have used the infrared laser (1034nm, 20W). The deliberation for this choice is given in 2.4. The delay between the trigger and the shot is for both lasers below 50ns. We take this into consideration for the subsequent calculations.

Objectives and Spot size The objectives utilized were a 5X, a 20X and a 50X. The spot sizes mentioned above refer to the 50X objective. With increasing magnification, the refractive index follows and subsequently the spot size decreases. Hence, the resolution is bigger for more powerful objectives. The 50X objective however is eminently hard to focus on the span of big surfaces, because the focusing depth is very small and the target is hardly perfectly horizontal. Moreover, it is extremely susceptible to the slightest disturbance, even the shutting of the door can set it out of focus. Hence, this objective is not used for preliminary scans, but rather for smaller areas, that are roughly on the same Z ground. In the subsequent experiments mainly 20x and 5x objectives where employed. Another limitation of the 50x objective is that it is hard to navigate on the die as the observed area is very small.

To calculate the laser spot for each objective the theoretical formula below is used. Given the *wavelength* of the laser and the *Numerical Aperture*, NA of the objective, the spot size is given by:

$$Laser\ Spot\ Diameter = \frac{1.22 \cdot wavelength}{\mathrm{NA}},$$

where $NA = n * sin(a)$ and "n" be the refractive index of the medium (in our case air, thus n=1), and 'a' be the half angle aperture of the objective[†].

Figure 2.2 and 2.3 present our laser fault injection setups.

2.2 Device under test

Having set up the tools, we needed to select a target, in order to prove our point. The selected target is the SoC STM32F417IG[‡]. The main reason behind this selection is the integrated hardware encryption core, which is necessary to implement the countermeasure and perform the attack described in this thesis. Furthermore, we opted for a commercial target in order to simulate a real case scenario, and this distinguishes this research from similar attacks. Finally this board has scored high in durability tests and we had already

[†]http://www.microscopyu.com/articles/formulas/formulasna.html
[‡]The STM32F4 family datasheet can be found here

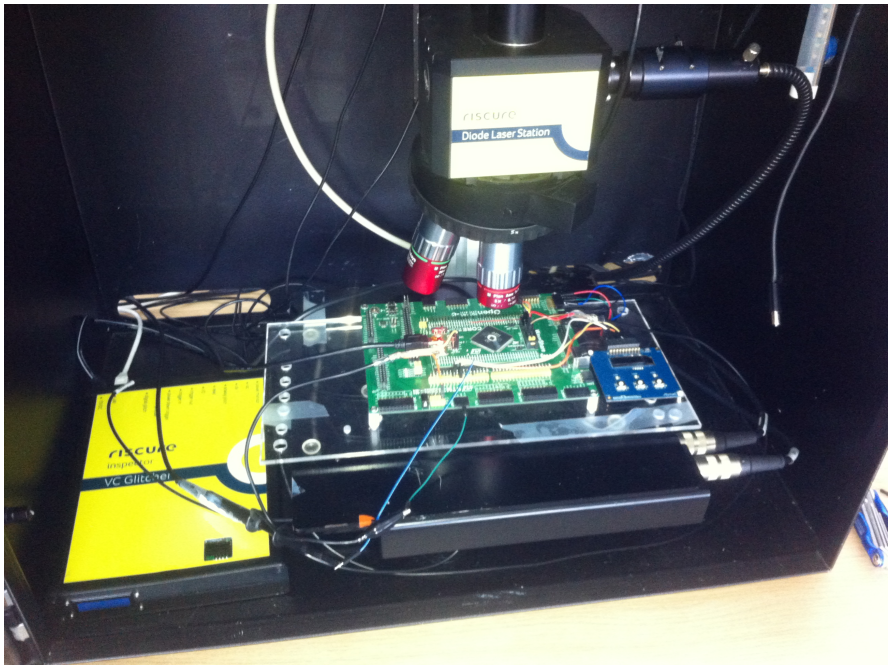


Figure 2.2: An overview of the Setup - Laser Station

many identical boards in stock. The last reason is important in a time-limited thesis, for ”killing” a board can cause a significant delay to the research due to ordering and decapping (see 2.3). The micro-controller is a Cortex-M4[§], based on the 32-bit ARM thumb-2® architecture[¶].

The system on chip is 90nm CMOS technology, hence the bit cell area is $1\mu m^2$. Our spot size is roughly 6 times larger, which corroborates that a single-byte fault injection is feasible. We will fondly call the board henceforward Pinata.

2.2.1 Programming the IC

The St-Link ISOL v2 was used for programming and debugging. An extra FTDI cable handles the Input-Output from the target to the computer. The built-in pins for input and output are occupied by the debugger. To round this problem, they are short circuited with two free pins on the board, thus enabling us to communicate with it, while debugging. The latter was required in order to debug the target, and delve into its assembly instructions. Finally the board is powered either via USB, or from a power generator. USB

[§]The Cortex M4 technical reference manual can be found here
[¶]The ARM v7-M architecture reference manual can be found here. For the latest version registration is required

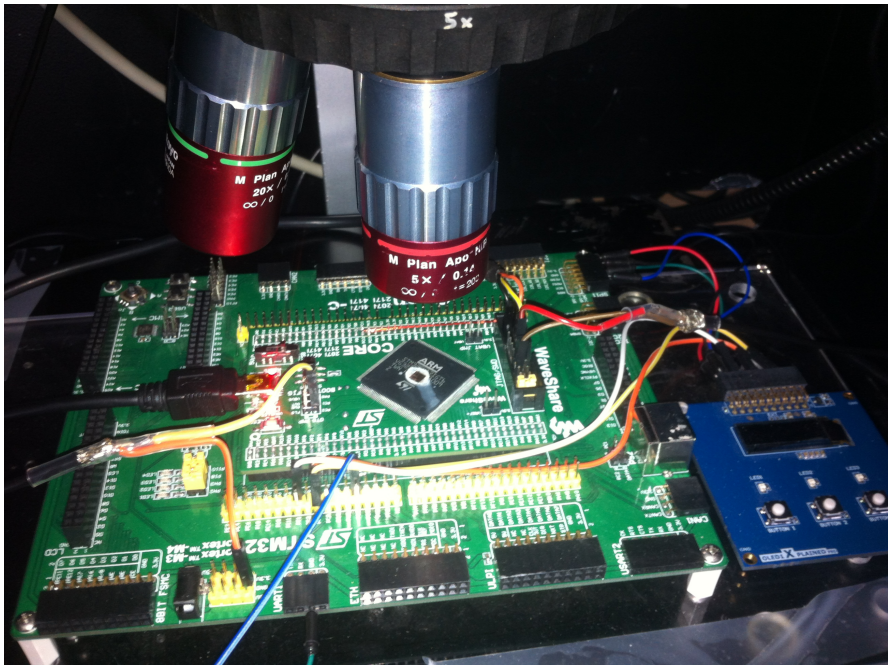


Figure 2.3: Close-up on the target

powers up with 5V while the power generator can be set to any value in the range 0-15V. We opt to set it to 3.3V, as is the recommended supply for a laser fault injection installation. After the installation of the drivers and the assembly of the parts, the programming as well as the debugging is a fairly straightforward process.

In order to expose certain vulnerabilities, and depending on the module that is targeted each time, the device under test is programmed accordingly. The code for each case is presented and explained in the respective section.

2.3 Decapping the Pinata

We decap the chips from the accessible to us side. This exposes the front-side of the die. The procedure is as follows. We initially mill a pocket of 4 mm diameter onto the epoxy layer. We apply nitric acid 4-5 times iteratively for roughly 4-5 seconds, we check whether the epoxy layer was adequately dissolved. We use acetone and isopropanol to wash off the residues. Finally, we check whether the die is properly exposed, otherwise we apply another iteration, as described above.

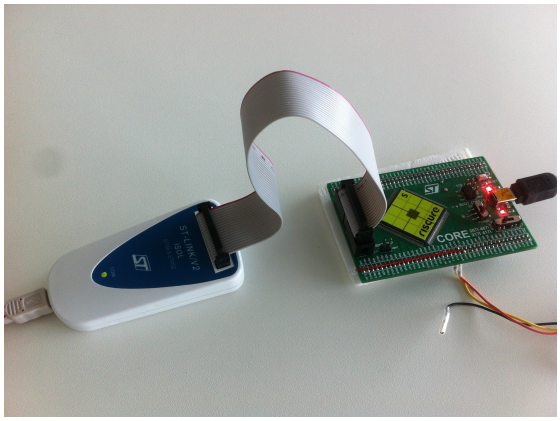


Figure 2.4: Pinata connected with ST-LINK/V2 ISOL debugger-programmer and serial I/O

As we will see in Chapter 4, before the end of this research we decapped the chip from the opposite side, subsequently exposing the back-side of the die. This task was significantly more challenging, because not only the painstaking soldering and de-soldering of the chip was required, but also milling a hole on the hard surface of the board in a cautious manner was required. The former was required for us to have access to the epoxy laser from the back-side and apply the aforementioned procedure, while the latter should ensure that the functionality of the board is not irreversibly damaged. Fortunately, the buses that had to be destroyed in our case, carried the USB Input - Output, which was not catastrophic; we rounded it by collecting the output from the JTAG pin.

2.4 Laser Energy

Before setting off to the attack, it is sensible to deliberate our choice of lasers with respect to the side we decide to ”enter”. From figure 2.5 we extrapolate that lower energies (higher frequencies) are absorbed less from silicon. Since back-side is covered with silicon, it is apparent that for entering therefrom, the 1034nm laser is suitable. Whereas the back-side is covered with the bulk of silicon, front-side is flooded by metallic layers forming the gates and logic of the chip. Hence, low absorption from silicon is no longer a requirement for front-side attacks. However another emerges. We need thinner laser beams that can penetrate easier due to less scattering by the metallic mesh, therefore, higher frequency lasers are selected (808nm, 404nm). Finally, when a back-side experiment is opted, the length of the silicon should be factored in for focusing correctly on the plane where the logical gates reside. This length is roughly 315µm, as described in [11].

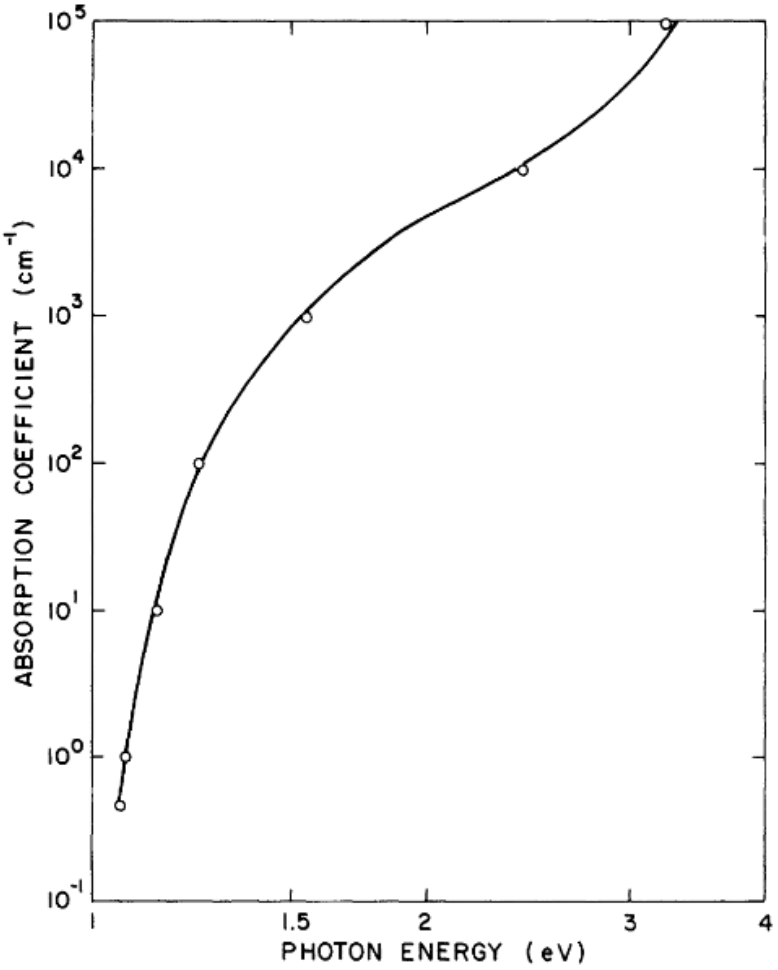


Figure 2.5: Absorption in intrinsic silicon

After this necessary theoretical and state-of-the-art background summary, we can now set off with describing the approach we followed in our research-attack.

3

Attacking the CPU

3.1 Identifying the vulnerable spot

CPU, or alternatively the main core is the first of the two spots we target, because it performs the comparison of the two ciphertexts that we want to skip. We bootstrapped the venture to find the CPU related logic by implementing the counter example. This is the name we have given to a loop that consists of two counters; one decreasing and one increasing. The implementation is presented in 3.1. The two counter design ensures that we will catch the glitch irregardless of whether the subverted value is bigger or smaller. Imagine that we have only one decreasing counter, a loop that returns when it reaches zero and we output the final value of the counter. Now, the glitch turns the counter to a bigger number. We will not catch

the glitch since the loop will terminate as expected when the counter reaches zero. The second counter whatsoever will reveal the glitch even in that case.

```
1            int up=0;
2            int down=1000
3              set_trigger();
4            {
5            while(--down){up++;}
6            }
7            clear_trigger();
```

Listing 3.1: The two-counter example

The above snippet describes our target, and the expected final values are $down = 0$ and $up = 999$. When the outputted values are in accordance with these the spot on the die is deemed green (see figure 3.1). Whereas is deemed red if any other pair of values, that do not agree with this, is outputted.

From the above preliminary scan we defined the correct energy range. While for low power the energy deposed on the die was not sufficient to inject a fault, for high energies the latch-up effect was observed. The latch-up effect would normally break the procedure, or give an erratic result. A broken iteration is deemed yellow. Figure 3.1 did not depict any yellows as this was a tuned scan. We clearly, whatsoever see a red pattern at the top-left quarter. That corroborates a strong indication that at this spot the CPU-related logic resides. To confirm this, we devised the code that is presented in 3.2. This code can validate this, and furthermore, we can thereby discover the vulnerable spot, in that vicinity, that enables the skipping of the instruction. We remind that this is the first objective of this thesis. Before we go into that, there are another two findings (see figure 3.1) whereupon we want to draw the attention. The red thin line from one side to the other, indicates a hard fault, or alternatively a persistent fault. The fact that it did not emerge in our subsequent experiments led us to discard it as an erratic event. Secondly, we can see that the whole region in the top left corner is red, which indicates that the glitch source power parameter was set correctly. In the opposite case, we should expect either to break the procedure, thus having yellow results, or not affect it at all, thus yielding green results.

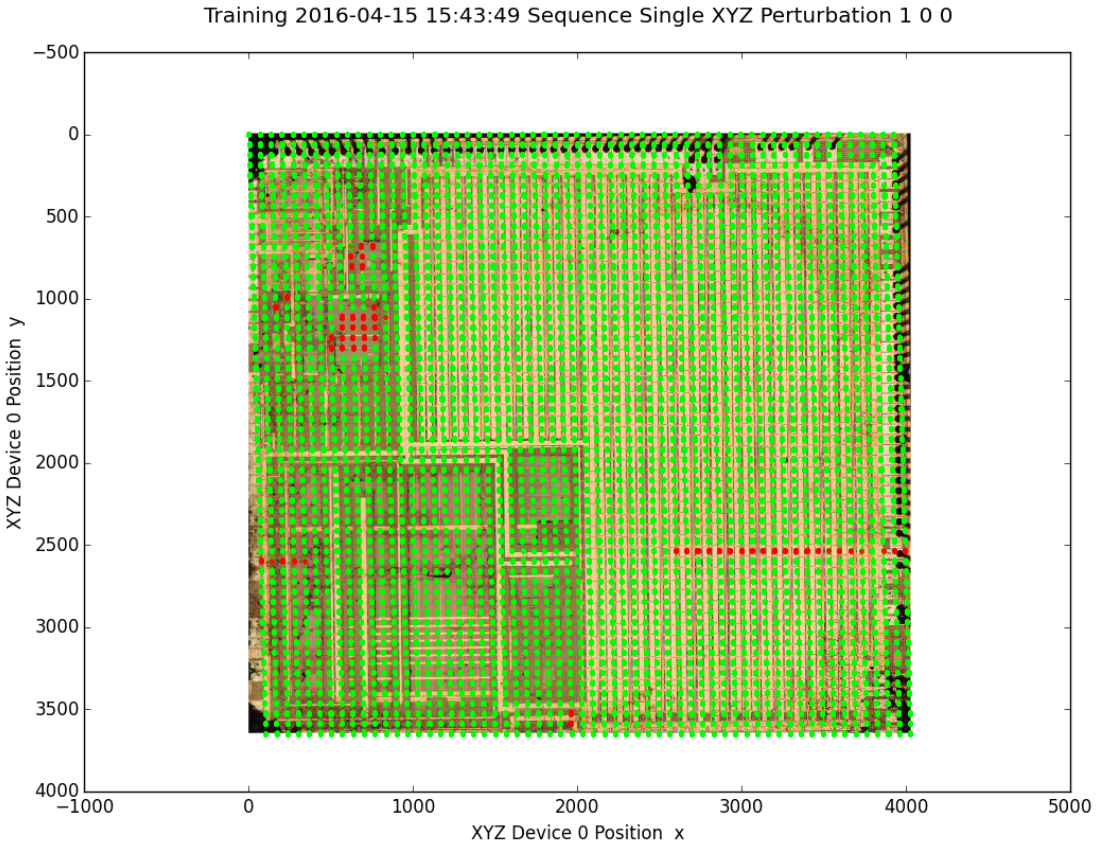


Figure 3.1: The CPU scan

3.2 Instruction skip

Let's return to the experiment that enabled us to track down the vulnerable position, that led to an instruction skip (without breaking the procedure). The code snippet 3.2 shows what was flashed onto the target. What we consider successful in this experiment is to skip the 'write' instruction in lines 24 and 25. Thus, not overwriting the initially placed value 0x66666666. If the output is 0x66666666 the glitch and consequently the instruction skip was successful, whereas if the output is 0x01010101, then we did not affect the target. As always we observed some erratic results, as well as mutes, and "breaks". For that purpose in all of our experiments we output the control value (A5), to ensure that the CPU functioned or terminate properly thus indicating that a sensitive spot was hit. If the control bytes are outputted as expected whereas the in-between values are erratic, then we can say that this spot is hit with the correct energy and it is a good

candidate to research furthermore. Highly likely a register, or a bus transfer was affected. The former case can lead to a memory dump.

```
        volatile unsigned int *play= (unsigned int *)0x2001bf00;
        volatile unsigned int *play2= (unsigned int *)0x1000a000;
          *play=0x66666666;
          *play2=0x66666666;


              set_trigger();
            {
              // loop glitch PEW PEW PEW
              asm (   "NOP\n"
                   "NOP\n"
                   "NOP\n"
                   "NOP\n"
                   "NOP\n"
                   "LDR r6, =0x2001bf00\n"
                   "LDR r5, =0x01010101\n"
                   "STR r5, [r6]  \n"
                   "NOP\n"
                   "NOP\n"
                   "NOP\n"
                   "NOP\n"
                   "NOP\n"
                   "LDR r6, =0x1000a000\n"
                   "LDR r5, =0x01010101\n"

                   "STR r5, [r6]  \n");
            }
            clear_trigger();

              send_bytes_uart(1,control_bytes);
        send_bytes_uart(sizeof(unsigned int),(uint8_t *)play);
        send_bytes_uart(1,control_bytes);
```

```
34          send_bytes_uart(sizeof(unsigned int),(uint8_t *)play2);
35
36                  volatile char buffer[112 * 1024]= {0};
```

Listing 3.2: The instruction skip snippet

We ran the above snippet on the target but we had to set the time offset correctly. In order to carefully time the pulse we manually set the trigger before the assembly instructions (see line 7 in 3.2). The reader can see the NOP's injected in the target. We opted for that in order to loosen the duration requirements for our glitch. This experiment was repeated without the NOP, with a high success rate. Subsequently, we went on debugging the target. Via JTAG we were stepping over each instruction, until we received the trigger rise. Therefrom we started counting. Our targeted 'write' was 9 instructions far from the trigger. This is not so intuitive since only six NOPs precede the targeted instruction. Nevertheless, the set_trigger() function call adds the extra 3 instructions (clock cycles) after the trigger is sent to the appropriate pin. Given that the clock is running at 168 MHZ, we can compute the suitable offset as $9/168M - 5ns = 9 * 5.9589 - 5(ns) = 40ns$. We factored in a 5 ns for the expected jitter of the laser. Intuitively enough, triggering earlier with a longer pulse duration has effectively, due to the NOPs, the same effect. Below is a close up of the presumptive arm cortex main core region.

Hereby we successfully glitched the main core, in fact we achieved an instruction skip. Many of the

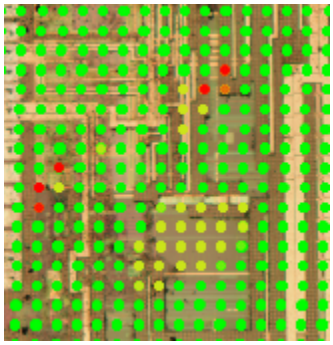


Figure 3.2: A close-up of the suggested region

seemingly successful glitches were filtered out, as some were associated with a CPU break-down. Some were outputting only partially and then muting. Given the targeted instructions, we expect the write not to happen, hence outputting the value 0x6666 6666 instead of 0x0101 0101.

In a real-case scenario we do not control the code executed, consequently neither do we have a trigger at

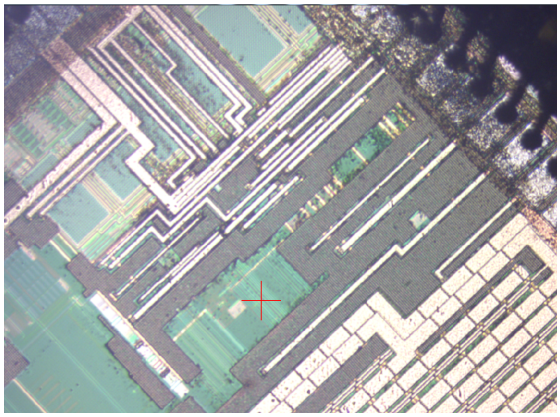


Figure 3.3: The vulnerable spot

16548	51.000000	74	30	1	713.703463	567.335880	false	A5 01 01 01 01 A5 01 01 01 01 A5 00 00 00 00 00 00 00 00
16549	60.000000	104	646	1	713.703463	567.335880	false	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
16550	84.000000	54	504	1	713.703463	567.335880	false	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
16551	35.000000	42	120	1	713.703463	567.335880	false	A5 01 01 01 01 A5 01 01 01 01 A5 00 00 00 00 00 00 00 00
16552	82.000000	64	612	1	713.703463	567.335880	false	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
16553	40.000000	46	400	1	713.703463	567.335880	false	A5 01 01 01 01 A5 66 66 66 66 A5 00 00 00 00 00 00 00 00
16554	69.000000	102	328	1	713.703463	567.335880	false	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
16555	84.000000	106	190	1	713.703463	567.335880	false	A5 01 01 01 01 A5 01 01 01 01 A5 00 00 00 00 00 00 00 00
16556	50.000000	48	366	1	713.703463	567.335880	false	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
16557	30.000000	88	350	1	713.703463	567.335880	false	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

our disposal. Finding, hence, the correct timing is challenging. However there are are certain methods to sort this. We can either set the offset from the reset of the target, or in a more high-end manner, we can apply pattern recognition. More precisely, the latter is carried out by costly equipment, but we can finally effectively trigger after an identified pattern adjacent to the vulnerable timing. For more details on how to track down the correct timing in a real case scenario please see subsection 4.1. There we explain how to track down the timing of the AES encryption machine. The comparison is highly likely to come right before or after the encryption. This approach on identifying the vulnerable timing of the comparison-check skip in an unknown target is addressed in the final section of this thesis, titled feasibility of this attack in reality (see 6.4).

3.3 Conclusions

In this section we followed a structured and methodological approach in order to find a vulnerable spot that can effectively lead to an instruction skip. Instruction skips can be powerful stepping stones to not only bypass security functions, but also to inject faults in software encryption processes. We managed to

neutralize the security feature in our scenario, that is the ciphertext check, by precisely tuning a laser shots' timing, duration, spot on the die and source power. We have furthermore devised transferable techniques and code snippets to identify and exploit CPU fault injection vulnerabilities that can be applied on virtually any commercial target. Finally, we have shown that embedded industry does not always take the necessary hardware security recommendations, as showcased in literature and in the present research, into serious consideration.

While shooting at the CPU, we mainly observe retained outputs or mutes. For that reason, we output a control byte to validate whether the procedure was broken, or muted. The erratic responds that, whatsoever, include the control byte corroborate an arbitrarily changed register, or a bus corruption. Also, in many cases, memory dumps were observed. We suggest to further investigate into arbitrary changes of registers, in order to dump selected memory segments. However, this is out of the scope of this dual laser attack, therefore we will not delve into this topic to a greater extent.

A countermeasure developed by the embedded industry is adding a random delay in computations. This could sabotage the efforts to track down the timing in a real case scenario. However, with the right amount of traces and high-end analysis this could even be circumvented. This example illustrates the ”cat and mouse” characteristic of the security domain.

4

Attacking the peripherals

For our proof of concept we have mentioned already that we need to attack a second core on the target. This, among others, corroborates the hardware distribution term selection. It furthermore necessitates a device under test with multiple cores-peripherals. We have chosen our target on that premise (we deliberated this process in section 2.2) as it integrates a cryptogrqaphic core that computes AES(various modes), DES and Triple DES. Furthermore it include other peripherals such as the hash core that is responsible for the computation of the hashes, namely MD5 end SHA-1. The latter also utilizes these hashes to compute the authentication algorithm HMAC. Another peripheral device is the Direct Memory Address arbiter, henceforward called DMA. DMA handles direct transfers from one of the peripherals to memory addresses and vice versa, albeit without the interruption of the CPU. Finally, it lodges the SRAM cells that accommo-

date the heap, the stack and the like. We attack these as well. Every of the above components reside at a certain distance from the main core, which makes this research the first to combine the attacks on two as distant spots on the die at the same time. Attacking various components also gives the opportunity to draw valuable insight on how to attack these components and can be leveraged in the future research as a stepping stone for more complex cases and scenarios. On the other hand we encourage industry to not rely in existent countermeasures and devise stronger immune ones.

4.1 AES

Hardware AES was chosen for obvious reasons. The main concern of this thesis is to achieve a laser fault injection into the AES core, between the 9th and 10th round as described in 1.4, thereby resulting to a faulty ciphertext. In order to inject a fault in the AES procedure, we started off with scans of the whole die, whereby we characterized the surface. The time offset was determined by measuring the duration of the procedure with the oscilloscope. We can set the oscilloscope to start measuring when it receives the trigger, and we can see the rise and fall thereof. This is roughly the duration that the crypto core is active, since our trigger is positioned tightly before and after its invocation.

For the discovery of vulnerable positions and timing we mainly have to set up the trigger and instantiate the crypto-core. For that purpose the code below (see snippet 4.1) was downloaded on the device. By sending the command byte 0xCA, the target waits for the 16 byte input, that is the plaintext, and it computes the ciphertext which it outputs. For hardware AES, the computation is carried out in hardware and the communication (I/O) is performed via the relevant 32-bit register, a word at a time. For more details on the technicalities of CRYP_AES_ECB() and nomenclature of the registers please refer to [1]. The implementation of the method belongs to STM and is integral to its standard peripherals' library.

```
1                 case (0xCA):
2      get_bytes(16, rxBuffer);
3      //Trigger pin handling moved to CRYP_AES_ECB function
4      cryptoCompletedOK = CRYP_AES_ECB(MODE_ENCRYPT, keyAES, 128, rxBuffer, (uint32_t)
     AES128LENGTHINBYTES, rxBuffer + AES128LENGTHINBYTES);
5      if (cryptoCompletedOK == SUCCESS) {
6        send_bytes(16, rxBuffer + AES128LENGTHINBYTES);
7      } else {
```

```
8         send_bytes(16, zeros);
9       }'
```

Listing 4.1: The hardware AES command

After the first scans we analyzed the results and we figured that the glitches were either not very precise, or affecting more than one byte. This was attributed to the fact that our clock cycle is very brief (5ns). Since our minimum laser shot duration was 20ns, in order to attack the AES we decided to down-clock the device. The SoC uses either an internal, or an external pll for clocking. No use of an external clock was made, hence we programmaticaly tuned the internal pll to clock the device at its minimum speed, namely 8 MHz. Hence, the clock cycle now has a duration of *125ns*.

In figure 4.2 a down-clocked iteration is shown. The time window is roughly 12 us. The disturbances depicted are not useful for analysis as they are noisy versions of the procedure's power consumption. In section 4.1 we describe the setup we used to take a better measurement.

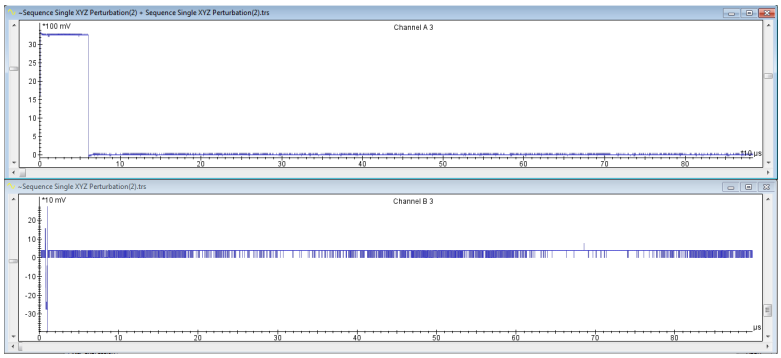


Figure 4.1: Trigger window and the pulse sent to the laser

The glitches happen at precise offsets as is shown in figure 4.4. We call these vulnerable timings. Since the computations are performed in words there are specific timings where each of the word can be subverted. This is furthermore confirmed in chapter 5.

A module was implemented, that collects all the traces (results) and resolves them to successful or not glitches. It moreover, resolves each result based on the color code we described earlier. A successful glitch is for instance when the result differs in exactly four bytes compared to the expected ciphertext. Then we call it faulty ciphertext and is depicted as a red dot. In this case we can be certain that we injected a single byte fault in the 9th round, as described in 1.4. These ciphertexts are suitable for our fault model. Figure 1.1 shows how a single-byte fault injection is propagated through the last two rounds and results

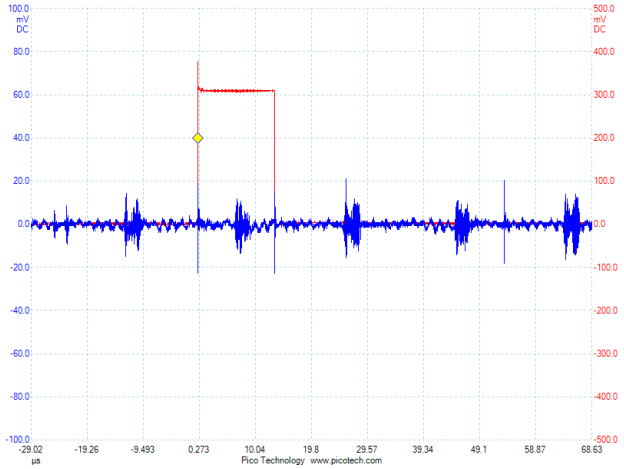


Figure 4.2: Down-clocked time window

in a subverted in four positions ciphertext. Having used all the combinations of lasers (blue and red) and objectives (5x and 20x), we did not yield any faulty ciphertexts. Hence, we deem that this boards' AES core is secure against such a fault injection. Nevertheless, we will pursue our research in proving the concept of two successful simultaneous glitches. Many of our results nearby the presumed crypto core vicinity were reproducible. They would happen at very specific point in time during the processors' activity and at certain spots. These are leveraged in order to prove our concept. Hence, the second glitch will be replaced by a controlled fault.

From the following figure 4.5 we can see a pattern of faults or breaks appearing on the die. Most of the faults are single byte changes, delays, outputs that miss a word, fact that leads us to the conclusion that we have found the main data bus that is used by the AES core to communicate internally and externally. The results were taken during hardware AES encryption.

Tracking down the precise timing in a real case scenario

In real case scenarios we do not have the luxury to have triggers pointing out the right timing. We setup an experiment to validate the correct timing, arrive to a better understanding of the target and confirm the attack as practical in real case scenarios. As mentioned previously there are two methods for arriving to this result. The first method, the one we followed in this thesis, was measuring the current consumption, by

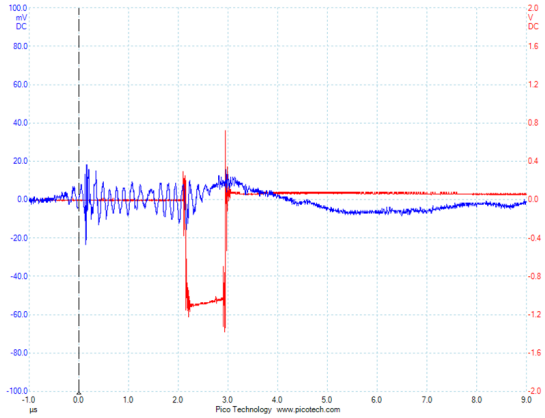


Figure 4.3: How the pulse influences the board

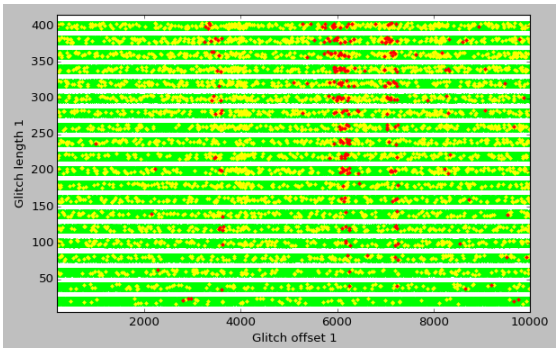


Figure 4.4: Vulnerable timing

interposing a current probe between the power supply, the board, and the oscilloscope. The current probe is a tool that enables measurements of power consumption of embedded technology and consequently side channel analysis. Our board was powered with the suggested 3.3 V power supply. The resulting power trace between the triggers' rise and fall is presented in figure 4.6. The measurement presented is cropped to the time window that the hardware core performs the AES, Electronic Codeblock cipher mode, encryption. The time window was defined with the help of the trigger, whatsoever after we have taken this measurement we can remove it and apply our findings in similar devices.

The second method used to track down the correct timing is Electromagnetic side channel analysis. In this case no trigger is needed therefore this method is universal and applies in real case scenarios in analysi of black boxes. The following figure 4.7 shows the input and output correlation computed over 1000 samples. We can see that the samples correlate at certain timings, these timings are demonstrated

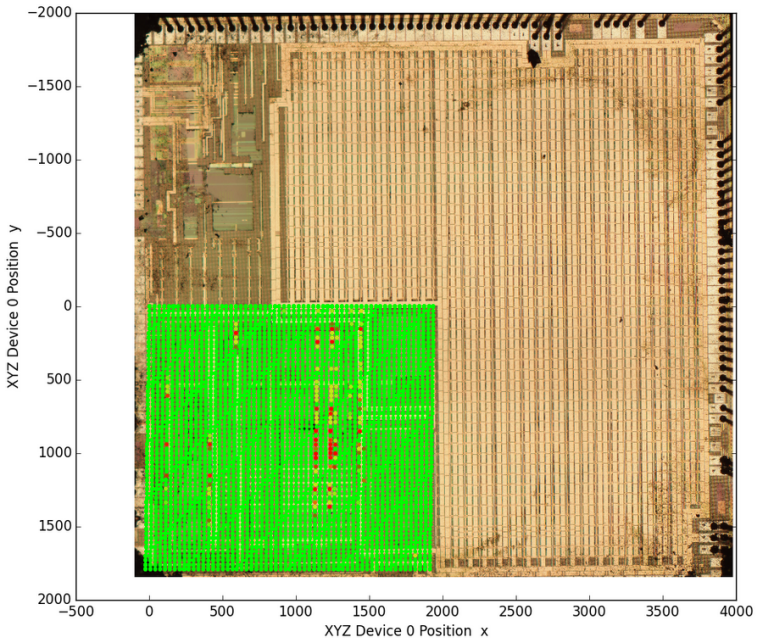


Figure 4.5: Decrypting the AES

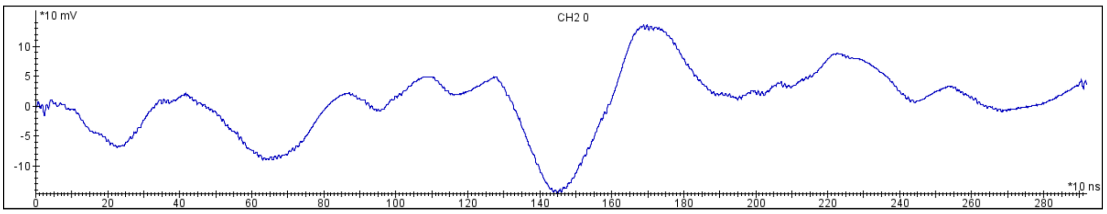


Figure 4.6: Power trace capture of the hardware AES procedure

by the input and output curves. Each of the four curves represent the processing of each of the words that the crypto-core receives as input. Similarly for the output. We derive that the encryption is happening inbetween, in fact, for this case between 1000 and 1600 ns. These numbers refer to the time elapse after crypto processing started. This processing start at a 500 ns offset from the reset.

4.2 SRAM

As we mentioned earlier we implement a suitable target in order to attack the SRAM cells. Hereby, the attack aims for a bit flip, which can be combined with a skip of the hash check of this SRAM block. This

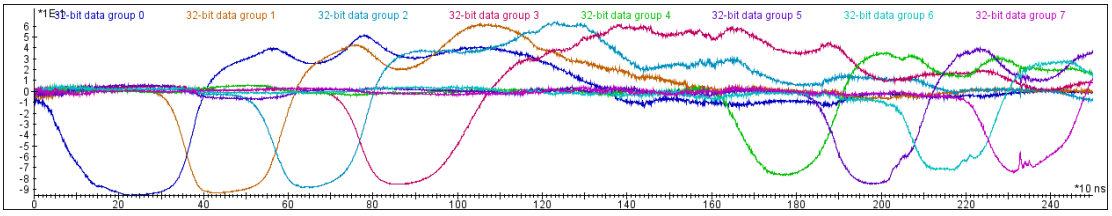


Figure 4.7: Input and output correlation of the Hardware AES process

is another dual laser attack scenario, where we want to cause a bit flip for instance to an executable part of the SRAM, hence forging the code executed, but at the same time neutralizing a countermeasure, such as the hash check of that block. At first we created a methodology for attacking the ram. The code we developed and runs during the described attacks is shown in the snippet of code 4.2

```
1                volatile char buffer[112 * 1024]= {0};
2                volatile int *startram =buffer;
3          volatile int *endram=buffer + NUM_ELEM(buffer);
4                int hits1 = 0;
5
6                memset(buffer,0x01,114688);
7
8                set_trigger();
9                for (volatile int *i = startram;i<endram;i++){
10             if(*i!=0x01010101){
11              hits1++;
12             }
13         }
14         clear_trigger();
15
16
17               case (0xCA):
18    get_bytes(16, rxBuffer);
19    //Trigger pin handling moved to CRYP_AES_ECB function
20    cryptoCompletedOK = CRYP_AES_ECB(MODE_ENCRYPT, keyAES, 128, rxBuffer, (uint32_t)
    AES128LENGTHINBYTES, rxBuffer + AES128LENGTHINBYTES);
```

```
21      if (cryptoCompletedOK == SUCCESS) {
```

Listing 4.2: The code behind RAM attacks

In order to attack the RAM and capture a bit-flip the following procedure was adopted. We constructed an uninitialized array and filled it with 0x10100101 words. Moreover we maximized the size of the array, for covering as much of the SRAM as possible. The stack consumes space during allocation that amounts to the stack limit, and is by default allocated in the attacked SRAM cells. We configured the linker to allocate the stack instead of the SRAM cells to the Core Coupled Memory, CCM. Hence, we managed to free some valuable space and store an array of 112 Kbytes. Our array is filled with the value 0x1010 0101 and is stored in the .bss segment that accommodates the uninitialized data. The value was chosen because we can catch both sets and resets of bits, whereas the word 0x1111 1111 could only capture resets. An overview of the addressable memory segments that this board provides is shown in table 4.1. The SRAM cells are discernible in the delayered picture of the die at the upper-right quarter (see figure 4.8)

Next, we set the trigger and instructing the laser to shoot, at a seemingly random offset, albeit within the

Table 4.1: Segment allocation

Segment	Memory	Size (bytes)	Memory address range
text	rom	95372	0x08000000 - 0x08100000
data	ram	3772	0x20000000 - 0x20020000
bss	ram	136440	»
heap	ram	-	»
stack	ccm	-	0x10000000 - 0x10010000

time window that the device is counting for subverted words (lines 8-14). The reason we simultaneously shoot and count is that we aim to record a temporary bit flip, that would otherwise have gone unrecorded. Nevertheless we do not stop there, we perform another similar iteration, counting for subverted word after the completion of the laser shooting. Thus, validating that the captured glitch was not a product of bus or CPU corruption, but rather a persistent bit-flip.

Figure 4.8 shows a delayered die such as the one under test. We can identify the memory block on the top right quarter. Despite the fact that we tried both lasers and all possible power ranges, to the extreme values that permanently destroyed one board, no permanent bit-flip, fulfilling the requirements described above, was recorded. Therefore the SRAM was deemed secure against a bit flip breach. From figure 4.9 furthermore, judging from the facet of the right half of the die, we presume that the SRAM blocks reside

underneath an impenetrable metallic shield.

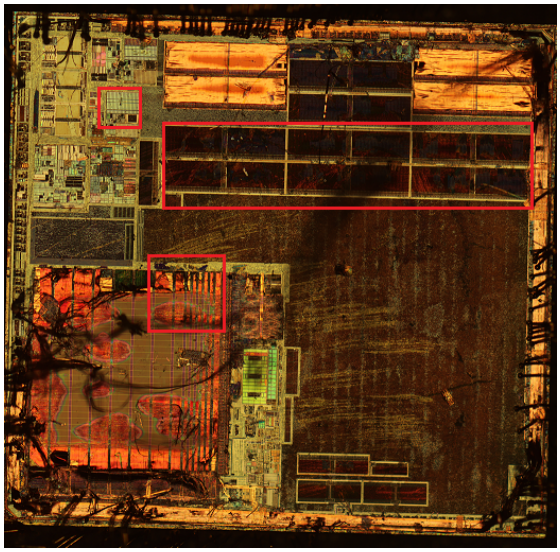


Figure 4.8: A decapped delayered picture of the die

As shown in figure 4.10 there is only one area where red, or yellow results are recorded. A red result signifies that the `hits` counter (line 11 from snippet 4.2) is not zero, hence a subverted word was found. Green designates an expected result, whereas the yellow cases are time-outs, incomplete sequences, or corrupted control bytes. It appeared that there were some glitches, but not on the presumed SRAM area. Moreover a second array enumeration didn't provide any solid results that would indicate that an SRAM cell was permanently changed.

4.3 HASH with DMA

Direct Memory Access module, DMA enables the transfer between a peripheral register and the memory, and vice verca, without the intervention of the CPU. It is sensible to mention that apart from the memory, all the available hardware registers, including the peripherals', are mapped to an address. The DMA transfer is carried out via a 16-byte FIFO buffer. Two registers, the stream x peripheral address register, `DMA_SxPAR` and the stream x memory 0 address register, `DMA_SxM0AR` control the input and output address (for more details on the registers please refer to [2]). Our efforts were focused on subverting in a controlled manner these addresses. There was no indication that those registers can be forged. Furthermore the output

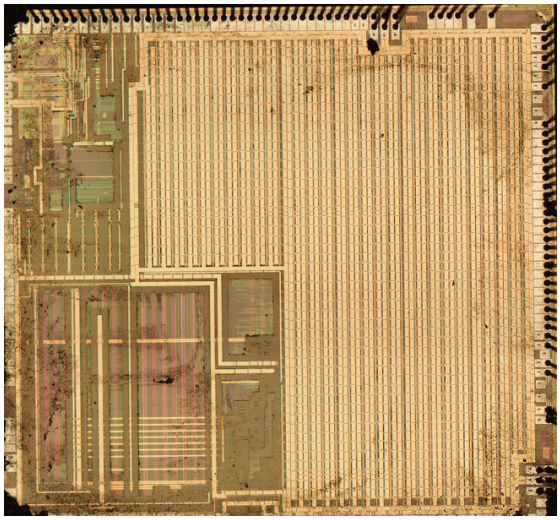


Figure 4.9: A decapped picture of the die

remained. Whatsoever, other registers were changed, therefrom the pattern in figure 4.11. This pattern presents all the spots that other registers participating in the hash computation, or DMA-related have been changed. After careful observation, we concluded that these registers were not directly subverted, rather, a glitch during the cryptoprocess forced them to lawfully change.

4.4 Back-side

After a series of unsuccessful front-side attacks, we finally opened the back-side at the cost of broken USB data transmission bus (see figure 4.12). The process for back-side decapping is described in section 2.3

Using the knowledge of the target that has been acquired, and described in the previous sections, we launched a primitive first attack on the die, in search for a vulnerable AES position. Shooting from the backside would heat the die to such extent, that it would constantly crash in the middle of an experiment. This was rounded with the setup shown in figure 4.13.

Finally, we yielded faulty ciphertexts with this procedure. The rate is not useful in that case as we were shooting in the blind regarding position, energy and distance of the target from the objective. The latter necessitates setting the plane of focus and depends on the thickness of the silicon substrate on the back-side. In [22] and [3] two approaches are described for figuring out the correct distance of the target. Unfortunately, the time was not sufficient to further characterize the target and try to raise the rate and

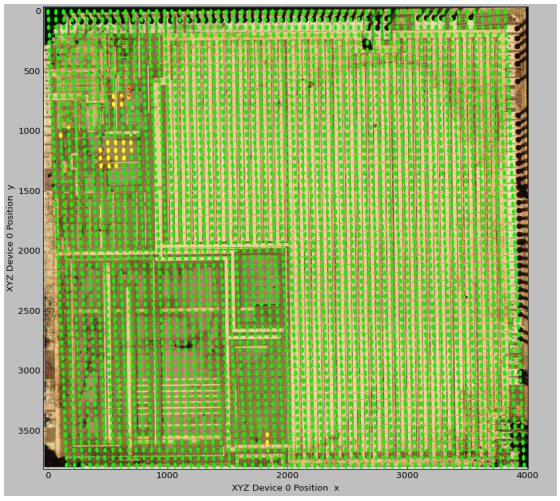


Figure 4.10: The ram experiment

the number of faulty ciphertexts. Even more to try the dual laser attack, thus having to characterize the first vulnerable spot. This is proposed as a further research, in order to fully breach this target.

4.5 Conclusions

In this chapter we attacked the various peripherals provided from our device under test according to the manufacturers' specifications. The reasoning behind this approach is to find vulnerable spots that allow us to subvert the output of the processes. The crypto core, and the SRAM cells are peripherals that we confront in the real case scenarios breaches. Successfully injecting a fault in one of these, even more combined with a simultaneous glitch (as we will show in the next chapter), can lead to a critical and meaningful breach of a state-of-the-art system. The implications of such attacks were thoroughly presented in section 1.1. We have shown that many peripherals can be under attack, broadening the attack surface of our targets. We have also followed a methodological procedure in order to capture injected faults in a variety of scenarios. Industry trends favor multi-core and multi-peripheral SOCs as well as cooperation thereof in computations[*]. Thereby the present attack will become more relevant, as new opportunities emerge and knowledge is garnered. We show that the system under attack is not be adequately secured against hardware attacks and more specifically laser fault injections.

[*] A term we hereby suggest is hardware distribution. This term captures the countermeasure aspect of the multi-core computation

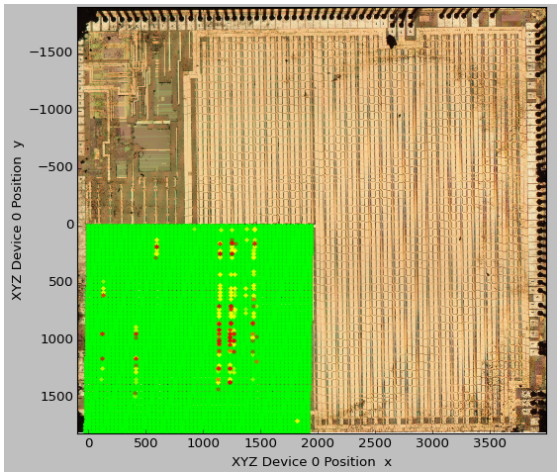


Figure 4.11: The AES with DMA scan

When targeting the cryptoaccelerator area, the outputs were mainly byte changes, word skips, or delayed output. However the final control byte would be sent, indicating that the cryptoprocessor returned ”success”, or terminated gracefully. Irregardless of the laser or the objective the results would not show any significant differences. We drew the conclusion that the red laser beam is perceived as ”slim” as the blue beam for this target. The objective however, and subsequently the spot size affects the potential for fault injection. Longer pulses suppress the output of the crypto-core but they do not break the procedure. This means that although the CPU will not receive any word of the ciphertext, SUCCESS is returned from the AES hardware, and the control bytes are printed.

Moreover, we drew the conclusion and confirmed previous research, see [11] and [22], that backside attacks can be significantly more successful, in that is easier to achieve fault injections. Whereas front side is covered with metallic mesh that acts as a security countermeasure making attacks unfeasible, back-side is free of countermeasures. We hereby suggest that industry should direly research into potential defenses thereof.

Finally, we observed that controllable and reproducible faults could be injected. When targeting at precisely the same coordinates with the same power and timing, an identical fault could be forced to the output. We leverage these spots in the next chapter, where we combine two of the previously explained attacks, in order to show that it is feasible to breach a system with distributed hardware participating in the security countermeasure. Our results show that such a countermeasure can be compromised.

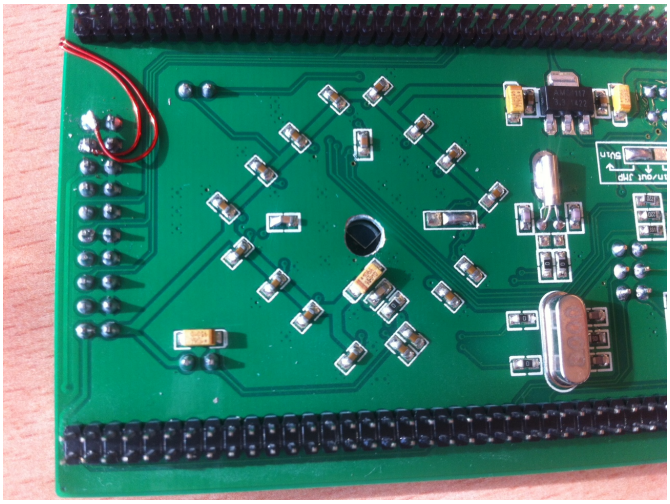


Figure 4.12: Entering from the backside

Figure 4.13: A setup with a fan

5

Combining the Attack

5.1 Preliminary work

Twinscan

Twinscan is the commercial name that stands in for dual laser station, tool set and infrastructure. The two lasers have been fixed to the frame as seen in figure 5.1, while their beams are chirourgically driven by a mirroring system via the same lens. In fact, a dedicated for each laser mirroring system is responsible for steering the beam over a X-Y plane. The mirror can steer the beams with an 1um precision over an area of 5x3mm. The latter makes the tool capable of achieving an attack against hardware, that resides in distant ”districts” on the die. The beams can also move relatively to each other. To the best of our knowledge no

Figure 5.1: Twinscan prototype used from programming

such equipment was used in research so far. A dedicated controller translates a serial input (the command) to signals that are in turn communicated to the four mirrors. These mirrors steer the beams over the two X and two Y axises, one for each laser respectively. Since the dual laser injection station is a brand-new setup, for the purposes of this thesis we implemented an interface. This interface interconnects the Inspector® functionality and graphical interface with the new commands that can be interpreted from the controller. The language of coding was Java and the difficulty in this task lies in redesign the moving-the-beams process, as it was in principal different from its predecessors. For more insight on the design choices, implemented functionality, and use-case scenarios please refer to chapter 2 of [13].

Setup

The setup, as described in chapter 2 and depicted in figure 2.4 is modified to provide for the two lasers.

In order to split the pulses and send them to the suitable laser, each aims at a different spot and has its own timing, a new component was positioned between the vc glitcher and the lasers. This integrated circuit, takes the vc glitcher pulses that come at the configured offsets and outputs to each laser based on pattern that is set by the user. The following example illustrates how the pattern works. Let's assume that we want to send the pulses alternatively to each laser (as in this attack). Let also assume that the first pulse (odd pulses) must trigger Laser A and the second pulse (even pulses) must trigger the Laser B. Then binary representation of the sequence for the first laser is 1010 1010 and so on. Similarly, for Laser B

the sequence is 0101 0101. Hence, in the splitter's interface we must type in for laser A, 5555 5555 5555 5555, and for laser B, AAAA AAAA AAAA AAAA. The extra bits allow for more complex patterns than the aforementioned.

Furthermore our setup needs a beam prism. This prism is suitable only for a range of wavelengths. The subsequent experiments were carried out with two 808nm laser, thus the prism was selected for the range 700-1100nm. Its purpose is to facilitate the channeling of the two beams, that are coming from different angles, into the same lens, as well as drive the reflected on the die light to the camera.

5.2 Implementing the target

The first step was to implement the countermeasure and the duplication of the encryption process. The specifications we set from the outset required the comparison of the successive encrypted outputs, as well as retaining it in case the ciphertexts were not identical. Hence, in order to bypass the concealment of the subverted ciphertext, not only we need to inject the fault at the right timing during the encryption process, but it is also required to skip the comparison that hides the output. The logical flow of the software is shown here.

```
1 ciphertext1 [] = CRYP_AES_ECB(MODE_ENCRYPT, keyAES, 128,  rxBuffer, (uint32_t)
      AES128LENGTHINBYTES, rxBuffer + AES128LENGTHINBYTES);
2
3 ciphertext2 [] = CRYP_AES_ECB(MODE_ENCRYPT, keyAES, 128,  rxBuffer, (uint32_t)
      AES128LENGTHINBYTES, rxBuffer + AES128LENGTHINBYTES);
4
5 if (ciphertext1.equals(ciphertext2)) then print ciphertext1
6 else print(rubbish)
```

From the above snippet we extrapolate that one glitch, will not suffice to break the system. for instance, injecting a fault in the encryption but not skipping the 'if' instruction will mute the output. Similarly, glitching the CPU, thus skipping the "if" command will not be more successful, as the correct ciphertex will be outputted

The implementation of the countermeasure in C is shown in listing 5.1.

```
1                 case (0xCA):
```

```
2       get_bytes(16, rxBuffer);
3       uint32_t * enc_1 = rxBuffer + AES128LENGTHINBYTES;
4       uint32_t * enc_2 = rxBuffer + AES128LENGTHINBYTES + AES128LENGTHINBYTES;
5       //Trigger pin handling moved to CRYP_AES_ECB function
6       cryptoCompletedOK = CRYP_AES_ECB_no_trigger(MODE_ENCRYPT, keyAES, 128,  rxBuffer
      , (uint32_t) AES128LENGTHINBYTES, rxBuffer + AES128LENGTHINBYTES);
7       cryptoCompletedOK = CRYP_AES_ECB(MODE_ENCRYPT, keyAES, 128, rxBuffer, (uint32_t)
       AES128LENGTHINBYTES, rxBuffer + AES128LENGTHINBYTES + AES128LENGTHINBYTES);
8       if (memcmp(enc_1,enc_2,16)==0) {
9         send_bytes(16, rxBuffer + AES128LENGTHINBYTES + AES128LENGTHINBYTES);
10      } else {
11        send_bytes(16, zeros);
12      }
13    break;
```

Listing 5.1: A software implementation of duplication

As inferred from the snippet above we instantiate the crypto-core twice (line 6 and 7). We store the outputs in adjacent memory spaces by adding an offset of AES128LENGTHBYTES (= 16 bytes) for every iteration. The offset is referring to the position of `rxBuffer` that contains our input, namely the plaintext. The storing in adjacent memory spaces does not have any implication in the realistic nature of the target, it is done so only for the purpose of easy retrieval. Furthermore, since we only need to glitch one of these processes we slightly modified the second iteration by setting a trigger only for the second encryption process. TDuring the first iteration no trigger is raised. We also want to skip the if instruction in line 8, thus forcing the target to output the ciphertext, even in the case of not matching results. Otherwise, the output is ascii zeros, or the expected ciphertext. In the following section we are setting the attack into motion.

5.3 Attack

The first thing to do, after having set up the Dual laser station and put our board under the gun, was to confirm the first spot. Based on our knowledge of the target (described in chapter 3), we have arranged a scan of a small area, with the recommended source power and duration. We identified the exact timing from the dissassembly of the if. As show in figure 5.3 the ”if” is 4 assembly instructions. We aim the branching instruction underlined with red. The 3 instructions previously to the if instructions, are the han-

dling the triggering, while the trigger pulse rises at the last instruction thereof, namely at `strh r2, [r3, #24]`. This was figured by connecting the board to an oscilloscope while stepping each instruction (debugging). Finally, we found the exact spot and timings, once again for the new setup. The experiment was slightly modified. In order to implement it as fast as possible, we substituted the line 8 from 5.1 with `if(memcmp(enc_1,enc_2,16) != 0 ) .. .`

Since we were shooting only with one laser the ciphertexts would be identical, hence, counterintuitevely a successful skip would output the correct ciphertext, in any other case the output is 16 00s. The result is shown in figure 5.2.

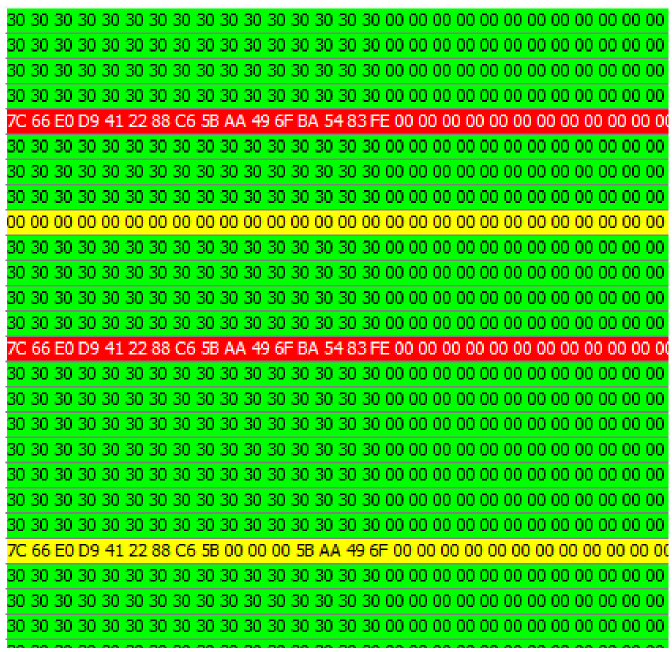


Figure 5.2: The assembly of our code adjacent to the if instruction

We injected NOP's before and after the if in order to make sure that we do not shoot at any other instruction with adverse results. After meticulous finetuning of the parameters we managed to raise the rate of successful glitches at 6%.

Now we needed a second spot where we can create a controlled fault. Looking into the results from the experiments carried out in chapter 4, we could find some candidates. We agreed on a spot that can produce the ciphertext

7C 66 E0 D9 41 22 88 C6 00 00 00 00 BA 54 83 FE

After fine tuning the rate can grow up to 60%-80%. The timing that led to the highest ratings is between 1140-1160ns after the trigger.

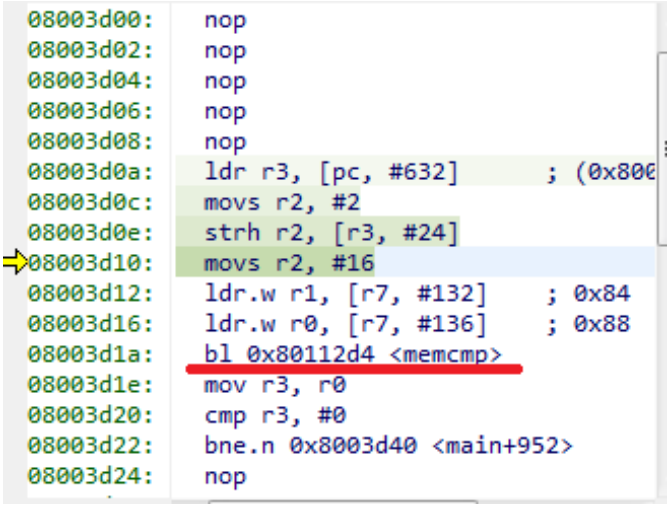


Figure 5.3: The assembly of our code adjacent to the if instruction

The last thing we needed to figure out was the timing for the second laser shot. In the final experiment due to the software design we have one trigger in our disposal. Therefrom we need to figure the two offsets. Since we are using the same trigger injected in the encryption process the first offset does not change. Hence, the offset for the glitch was configured as a random value between 1140ns and 1160ns. In order to find the offset for the second shot that skips the comparison, we injected a trigger into our target immediately before the "if" we aimed to bypass. Then with the aid of the oscilloscope we measure the delay of the second trigger in comparison to the first. As described earlier the first trigger is raised right before the encryption procedure. The second trigger would be discarded from the final target, hence we subtracted the 3 clock cycles consumed by it (see fig 5.3), but also added as many as needed to find the assembly line responsible for jumping. We need to be very precise in time. The second trigger as we see rises 2300ns after the first trigger, therefore we expect the branch instruction to come at an offset of 2400ns. Figure 5.4 shows the two triggers, as well as when the encryption and if routines are happening.

Having collected all this data we configured every software detail and we let the experiment run overnight, over a small area for the CPU region (see 5.5). Next day we had the objective. A controlled fault was injected in a part of the die, while the muting was bypassed by shooting the CPU, thus outputting the faulty

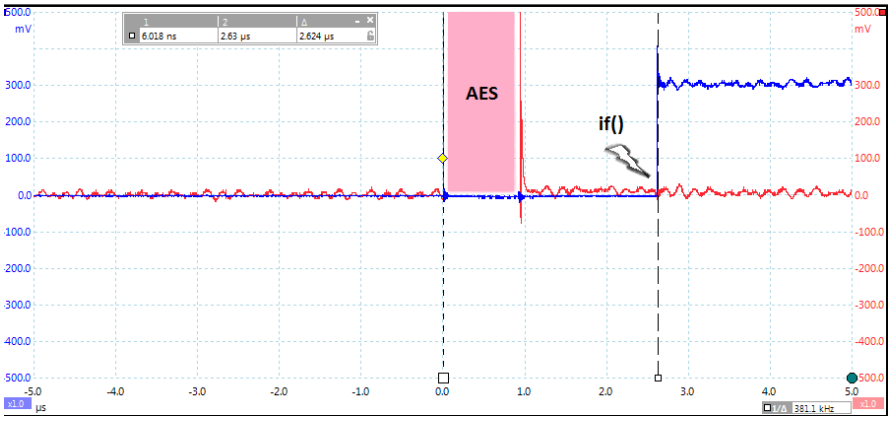


Figure 5.4: The time distance between the 2 triggers

ciphertext (see figure 5.6.

5.4 Conclusions

In this section we combined the previously described single attacks into the dual laser attack. We leveraged the potential that the dual laser station offers, to aim two vulnerable positions simultaneously. We described two ways on how to find the correct timing for an attack. Subsequently, we used the previously acquired knowledge after the characterization of the target, to position the laser on top of the correct spots. Inevitably, due to switching to the new laser station, re-installing software with the new module and refixing the target, we had to calibrate the coordinates from scratch and perform smaller area scans. A quick ”re-characterization” of the target for the correct spots was thus required. The fact that we have successfully injected the same faults in the new setup corroborates that this approach is reproducible. Moreover, the procedure of characterization described in the previous chapters, and subsequently setting up the dual laser attack combining the components, can be applied in all commercial targets that implement similar countermeasures. We have shown how we can target these peripherals and test for vulnerabilities that lead to a bit-flip, or a faulty ciphertext. In the next chapter our findings are summed up, and we draw the conclusions that emerged during this research.

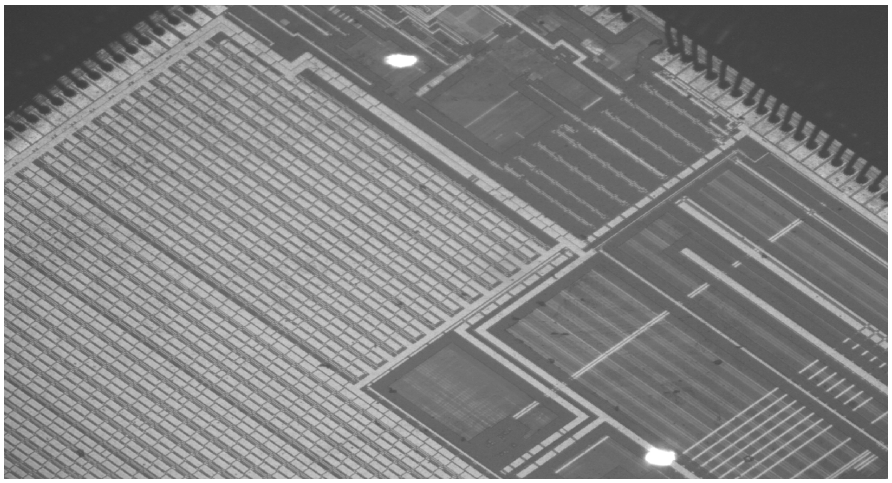


Figure 5.5: The exact time when both lasers shoot

50.000000	1222	74	1	-119437.40...	30890.666843	634734.004...	333317.683...	false	7C 66 E0 D9 41 22 88 C6 5B AA 49 6F BA 54 83 FE
60.000000	1232	76	1	-119437.40...	30890.666843	634734.004...	333317.683...	false	7C 66 E0 D9 41 22 88 C6 5B AA 49 6F BA 54 83 FE
57.000000	1232	78	1	-119437.40...	30890.666843	634734.004...	333317.683...	false	7C 66 E0 D9 41 22 88 C6 5B AA 49 6F BA 54 83 FE
56.000000	1226	72	1	-119437.40...	30890.666843	634734.004...	333317.683...	false	7C 66 E0 D9 41 22 88 C6 5B AA 49 6F BA 54 83 FE
56.000000	1206	70	1	-119437.40...	30890.666843	634734.004...	333317.683...	false	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
61.000000	1230	68	1	-119437.40...	30890.666843	634734.004...	333317.683...	false	7C 66 E0 D9 41 22 88 C6 5B AA 49 6F BA 54 83 FE
62.000000	1244	78	1	-119437.40...	30890.666843	634734.004...	333317.683...	false	7C 66 E0 D9 41 22 88 C6 00 00 00 00 BA 54 83 FE
58.000000	1200	72	1	-119437.40...	30890.666843	634734.004...	333317.683...	false	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
56.000000	1204	80	1	-119437.40...	30890.666843	634734.004...	333317.683...	false	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
51.000000	1222	74	1	-119437.40...	30890.666843	634734.004...	333317.683...	false	7C 66 E0 D9 41 22 88 C6 5B AA 49 6F BA 54 83 FE
64.000000	1234	70	1	-119437.40...	30890.666843	634734.004...	333317.683...	false	7C 66 E0 D9 41 22 88 C6 5B AA 49 6F BA 54 83 FE

Figure 5.6: The faulty ciphertext is outputted by bypassing the countermeasure

6

Conclusion

6.1 Summary

In this chapter we discuss the results of the experiments, the feasibility of the attack in practice,and suggested research. Our results indicated that a simultaneous fault injection in two different processes of two distinct cores almost at the the same time is clearly feasible. Therefore, in security critical applications, hardware distribution in combination with a detection countermeasure do not constitute a sufficient countermeasure.

By targeting the crypto-core vicinity we managed to reproduce controlled faults, resulting in subverted ciphertexts in very high rates. Irregardless of the laser, or the objective the results would not show any

significant discrepancy. During the CPU attack the glitch was not feasible when targeting with the 5X objective. We later on observed that it was not because the intended glitch would not happen, but rather due to the large spot size, adjacent CPU areas would be affected. These spots would lead in a 'break' process. This can be seen as successful glitches being covered by the inconclusive, erroneous terminations of the CPU.

When we targeted the CPU we observed either mutes, or breaks. The latter was attributed to the arbitrary changes registers have undergone, overheating of the die, or over the top source power. Sometimes crypto would not return success, thus outputting 3030303030. This can happen if we hit with very high power, or if the output, or input routine of the crypto core does not conclude successfully.

During the CPU attacks we observed arbitrary dumps of information. This information could be static, or constant variables, as well as initialized arrays, that reside in the data segment of the physical memory. Our suspicion is that the register that points to the memory that is outputted is subverted. No further investigation was performed on this topic, as it is out of the scope of research. Nonetheless this can lead to severe disclosure of sensitive information, and can be leveraged as a stepping stone, or standalone to breaches of state-of-the-art systems.

The reproduction of our results was confirmed by achieving similar results both with the simple laser station and the Twinscan. To switch stations we had to assemble all the tools and set up the connections from scratch. We moreover, changed devices under test during the experiments and the results were consistent.

Finally, We have shown two ways of finding the correct timing of an AES cryptoprocedure. We described the tools that can make it possible. Also, we followed a methodological approach on nailing down vulnerable spots, that is furthermore reproducible. We devised and presented code snippets that are appropriate to capture faults in each attack scenario. Moreover, we implemented a countermeasure that can thwart known one laser attacks, and we have proven that it can be compromised with a two laser setup. We showcased that the overall security of a target cannot rely on hardware distribution, but rather on thwarting fault injections over the die. As we have described to comprehensively secure a die from similar set-ups is impractical, therefore new countermeasures should be designed. We outline a research direction regarding countermeasures in the next section. Furthermore, we grasp the industry trend of accumulating cores onto the die, thus manufacturing multi-core SOCs. This attack will be more relevant in the future, as even a single encryption process might be computed by more than one cores. Finally, we have followed an

approach that can be applied irregardless of target, as long as we have optical access to the die; hence this is transferable.

6.2 Further research

In this section we outline directions for further research.

The back-side attack revealed a presumably vulnerable position regarding the AES. This corroborate the potential these attacks have. It is worth it to open the back-side, even if this looks unfeasible at first due to the underlying wiring of the board. Further research into the back-side attack is recommended for characterizing the device from the back-side, confirming the vulnerable state register of the crypto-core and finally breaching the target by retrieving the key. Also, the back-side is free of countermeasures, the industry should urgently contemplate on securing it in order to extinguish this risk. Naively we suggest adding a plain metallic layer to thwart back-side attacks.

Finding countermeasures for this kind of attack is another domain of exuberant research. Further research is also suggested for improving infection countermeasures, such as designs that introduce random delays, and dummy routines, as described in [25]. Thereby, finding the correct timing will become impractical. Furthermore, as embedded industry includes ever more cores, the possibility to distribute even more the processing, can make the attack described here unfeasible.

In chapter 3 we have observed memory dumps happening because of an arbitrary but random register corruption. Further research on these dumps on commercial targets is recommended. The risk of this attack is high, as sensitive information, compromising the underlying system, can be disclosed. The ability to compromise CPU registers in a controlled manner has a dire impact on the security of any system. Numerous exploits can be devised from this outset. Securing at least the CPU registers must become top priority for embedded industry.

6.3 Limitations

In this novel approach of the dual laser attacks, there is a limitation that this thesis was confronted with. We mentioned in the introduction about the trade-off between the objective that is used and the spot size, and we hereby explain it. We know so far that the objective and the spot size are conversely relative, in fact an increasing magnitude of magnification entails a smaller spot size. In modern embedded systems with a lot

of metallic layers smaller spot sizes are necessary, as they can penetrate where larger ones cannot. On the other hand if the targeted spots are too far apart that the 20x objective cannot involve both, then only a 5x objective can satisfy that spatial span. Which limits how small our spot size can be, thereby significantly limiting the precision of attacks.

6.4 Feasibility of the attack in reality

In this section an approach is suggested on how we can apply characterization of an unknown, non-controlled target used as a state-of-the-art equipment. As our target is a commercial device, this attack is fairly similar to what we should expect to confront in real use cases. Finding the vulnerable spots is the main difficulty as it was in our research. This can be only carried out by thorough observation and analysis of the results. However in reality we do not control the binary that is running on the target. Thereby, imposing another difficulty, that is finding the correct timing for both shots. In that case we wouldn't have the convenience to use the trigger as we had in this research. A suggested approach to round that problem is to procure an identical target, in order to set up our own code on the target, and using the methodology described in this thesis to figure the exact spots on the die where each core is vulnerable. After this is sorted, we can derive the timing by side channel analysis of the target. In section 4.1 we presented a methodology for identifying the timing an AES process is carried out based on electromagnetic emissions (leakage). Having figured the vulnerable spots and the timing of the AES, we can leverage our intuition - suspicion on which countermeasure is implemented, and fine tune our experiments. By running lengthy experiments over large areas and perturbation parameter ranges, we have high chances of finding the second timing required, that of the countermeasure. Finally, we can force the faulty ciphertexts and retrieve the key, thus invalidating the security function of the device, and breaching into the system.

Bibliography

[1] Description of stm32f2xx standard peripheral library. http://www.st.com/content/ccc/resource/technical/document/user_manual/59/2d/ab/ad/f8/29/49/d6/DM00023896.pdf/files/DM00023896.pdf/jcr:content/translations/en.DM00023896.pdf, Dec 2011.

[2] Rm0090 - reference manual. http://www.st.com/content/ccc/resource/technical/document/reference_manual/3d/6d/5a/66/b4/99/40/d4/DM00031020.pdf/files/DM00031020.pdf/jcr:content/translations/en.DM00031020.pdf, May 2016.

[3] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. How to flip a bit? In *On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International*, 2010.

[4] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006.

[5] Colin Barker. Hackers and defenders continue cybersecurity game of cat and mouse. http://www.zdnet.com/article/hackers-and-defenders-continue-cyber-security-game-of-cat-and-mouse/, 2016.

[6] Jem Berkes. Hardware attacks on cryptographic devices. *Prepared for ECE 628, Winter 2006*.

[7] Franck Courbon, Philippe Loubet-Moundi, Jacques JA Fournier, and Assia Tria. Adjusting laser injections for fully controlled faults. In *Constructive Side-Channel Analysis and Secure Design*, pages 229–242. Springer, 2014.

[8] Christophe Giraud. Dfa on aes. In *Advanced Encryption Standard–AES*, pages 27–41. Springer, 2004.

[9] Christophe Giraud and Hugues Thiebeauld. A survey on fault attacks. In *Smart Card Research and Advanced Applications VI*, pages 159–176. Springer, 2004.

[10] Donald H Habing. The use of lasers to simulate radiation-induced transients in semiconductor devices and circuits. *Nuclear Science, IEEE Transactions on*, 12(5):91–100, 1965.

[11] Clemens Helfmeier, Dmitry Nedospasov, Christopher Tarnovsky, Jan Starbug Krissler, Christian Boit, and Jean-Pierre Seifert. Breaking and entering through the silicon. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 733–744. ACM, 2013.

[12] Donald Knuth. The reset hack: a new exploit on xbox 360. http://www.logic-sunrise.com/news-341321-the-reset-glitch-hack-a-new-exploit-on-xbox-360-en.html.

[13] Yannis Koukoulis. We are all lean, from start-ups to larger organizations. *University of Twente (publication pending by the time of writing)*, 2016.

[14] Nate Lawson. How the ps3 hypervisor was hacked. https://rdist.root.org/2010/01/27/how-the-ps3-hypervisor-was-hacked/.

[15] Paolo Maistri. Countermeasures against fault attacks: The good, the bad, and the ugly. In *2011 IEEE 17th International On-Line Testing Symposium*, pages 134–137. IEEE, 2011.

[16] Amir Moradi, Mohammad T Manzuri Shalmani, and Mahmoud Salmasizadeh. A generalized method of differential fault attack against aes cryptosystem. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 91–100. Springer, 2006.

[17] NIST FIPS Pub. 197: Advanced encryption standard (aes). *Federal Information Processing Standards Publication*, 197:441–0311, 2001.

[18] Cyril Roscian, J-M Dutertre, and Assia Tria. Frontside laser fault injection on cryptosystems-application to the aes'last round. In *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, pages 119–124. IEEE, 2013.

[19] Dhiman Saha and Debdeep Mukhopadhyay. A diagonal fault attack on the advanced encryption standard.

[20] John R Samson Jr, Wilfrido Moreno, and Fernando Falquez. Validating fault tolerant designs using laser fault injection (lfi). In *Defect and Fault Tolerance in VLSI Systems, 1997. Proceedings., 1997 IEEE International Symposium on*, pages 175–183. IEEE, 1997.

[21] Alexandre Sarafianos, Cyril Roscian, J-M Dutertre, Mathieu Lisart, and Assia Tria. Electrical modeling of the photoelectric effect induced by a pulsed laser applied to an sram cell. *Microelectronics Reliability*, 53(9):1300–1305, 2013.

[22] Bodo Selmke, Johann Heyszl, and Georg Sigl. Attack on a dfa protected aes by simultaneous laser fault injections. *FTDC.2016.16*, 2016.

[23] Sergei Skorobatov. Fault attacks on secure chips: from glitch to flash. In *Design and Security of Cryptographic Algorithms and Devices (ECRYPT II)*. Albena, Bulgaria, 2011.

[24] Elena Trichina and Roman Korkikyan. Multi fault laser attacks on protected crt-rsa. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2010 Workshop on*, pages 75–86. IEEE, 2010.

[25] Harshal Tupsamudre, Shikha Bisht, and Debdeep Mukhopadhyay. Destroying fault invariant with randomization. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 93–111. Springer, 2014.

[26] Jasper van Woundenberg. Embedded systems under fire fault injection on secure boot. https://www.riscure.com/documents/rsa_presentation_2013_jasper_van_woudenberg_riscure.pdf?1403039987.

This thesis was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 11 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface. The above illustration, "Science Experiment 02", was created by Ben Schlitter and released under cc by-nc-nd 3.0. A template that can be used to format a PhD thesis with this look and feel has been released under the permissive mit (x11) license, and can be found online at github.com/suchow/Dissertate or from its author, Jordan Suchow, at suchow@post.harvard.edu.