Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot - Extended Version

Aurélien Vasselle, Hugues Thiebeauld
Eshard - Bordeaux, France
aurelien.vasselle@eshard.com

Quentin Maouhoub, Adèle Morisset, Sébastien Ermeneux
ALPhANOV - Bordeaux, France
adele.morisset@alphanov.com

(Invited Paper)

Abstract—This paper describes the outcome of a laser attack study on an Android smartphone targeting specifically the secure boot sequence. Laser fault injection has become a classical attack path in the secure chip industry to investigate potential security mitigation. The implementation of such attacks on a recent mobile phone remains relatively unexplored and represents different challenges, both at hardware and software levels. In this paper, we show how the device is crafted to get a direct access to the silicon and explain the corresponding experimental setup. By inserting our own software into the boot sequence, it was possible to achieve a fine characterization of the die sensitivity to laser emissions. With the knowledge of potential perturbations, several attack scenarios were built, allowing to malevolently get the highest level of privilege within the mobile phone.

Index Terms—Laser Fault Injection; Smartphone; SoC; Secure Boot; Hardware security; Reverse engineering; TrustZone; TEE

1 Introduction

1.1 Laser Fault Injection

Hardware fault injection has become a common practice to stress the security of embedded devices like secure controllers, RAM, processors, etc. An incorrect behavior can be obtained with out of spec usage, generally through unexpected variations of physical properties, such as voltage underfeeding [1], clock or voltage glitches [2], silicon substrate biasing [3] or even electromagnetic [4] and light beam interactions [5].

Optical fault injection is a technique allowing to induce a physical disruption inside a chip aiming to create a faulty behavior during code execution. Practical results were first published by Skorobogatov and Anderson [6], using a focused camera flash to obtain flipped bits by locally interfering with an integrated circuit. Relatively shortly after this publication, laser systems turned out to be a very powerful mean to inject faults on semiconductor devices. Indeed, laser systems combine a high power with fine control over the surface and time to achieve finer disruptions. The laser beam generates electron-hole pairs when passing through the silicon with an appropriate photon energy of at least 1.12eV (silicon band-gap), corresponding to a wavelength of at most 1100nm [7]. The electrical field generated by the transistors can make these charges drift and create a transient photocurrent. This spatially and temporally localized photocurrent may have an incidence on the circuit logic. It can cause multiple bit errors as well as controlled and reproducible single bit-flip by choosing the appropriate laser parameters [8].

Cryptographic attacks assisted by fault injection into secure devices have proven the ability to obtain bits of a secret key since the Differential Fault Analysis (DFA) [9] [14]. Most algorithms are theoretically vulnerable to laser fault attacks: lots of studies were performed on Data Encryption Standard (DES) [10], Advanced Encryption Standard (AES) [11] [12] [13] and RSA [14]. As a result of the cat and mouse game, dedicated countermeasures have been designed to tackle the threats behind faults. A first principle is to detect the physical disruption using sensors. In particular light absorption can be prevented with photosensitive semiconductor structures implemented on the sensitive areas of the die [15]. A more generic approach makes use of algorithmic protections at hardware or software level to ensure redundancy and/or error correction of the program and data flow [16]. State-of-the-art laser attacks now require to get very fine beams to find a way between sensors and other hardware protections. Moreover, applying different laser pulses on the silicon in time and space became a way to circumvent logical protections in secure software. Exploring laser attack techniques to validate the security of a product is important, particularly when the silicon can be directly accessed without big efforts [17].

A lot of parameters must be considered when implementing a laser fault injection attack: wavelength, spot size, location on the chip (X, Y), focus (Z), timing, pulse width and intensity. Only a proper combination of these physical parameters can lead to valuable faults and be exploited through a practical attack scenario.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2860010, IEEE Transactions on Computers

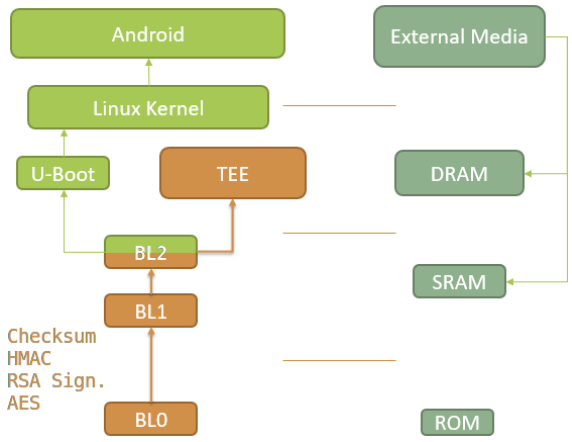

Fig. 1: Secure boot program flow

1.2 Secure Boot

Secure Boot refers to a sequence of cryptographic verifications ensuring the authenticity and conformity of bootloader software in order to sustain a Chain of Trust. By doing so, the device can securely initialize a Trusted Execution Environment (TEE), starting from the System on Chip (SoC) bootrom code, inherently secure. It allows software, peripheral and I/O protection, code isolation and security management. ARM leadership on the smartphone market has placed TrustZone, their hardware-based security solution, inside almost all the devices offering secure boot.

To ensure the origin of any software and make sure it can be considered secure, the device relies on software verifications involving asymmetric cryptography, in which trusted authorities can sign a code with a secret private key. Doing so, everyone has the opportunity to authenticate the author by using the corresponding public key. This is particularly critical when dealing with several entities and companies.

Even if software attacks have exploited design flaws in the TEE to obtain secure code execution [20] [21], only few attacks targeted directly the secure boot sequence. Recent research has developed BootStomp, an automated vulnerability analysis tool [22] for bootloaders available as cleartext binaries. In most cases, secure boot is an elusive attack target since a standard user has no way to either interfere with the executed software or even understand it, provided that it is properly ciphered and/or obfuscated. Moreover, thanks to its limited attack surface, this primary code is by nature more likely to be flawless. This explains why it is particularly valuable to consider physical attack scenarios during the early stages of boot. Previous work has demonstrated a successful voltage glitch attack [23] exploiting ARMv7 program counter specificities.

1.3 Motivations

As seen in the smartcard industry over the past 15 years, hardware attacks can take security considerations one step further by stressing software and cryptographic implementations considered secure. Yet, their applicability to mobile phones is not obvious. Among these examples the laser, if well mastered, constitutes the tool with the greatest potential to perform fault injection on a complex SoC, thanks to its high precision and efficiency.

However, the meandering route towards a working proof of concept requires to overcome several technical challenges. Indeed, exposing the SoC die within a mobile phone represents the first obstacle since the Package-on-Package technology became usual in mobile phones or smart devices industry: the SoC is confined between the PCB and stacked DRAM dies, and therefore cannot be easily exposed to laser beams. When mechanical issues are overcome, the physical dimensions involved in the technology still represents a challenge by the huge parametric exploration space, which cannot be entirely exhausted. Furthermore, mobile phones' software architecture is complex and built upon several levels of privileges. The whole platform security heavily relies on the secure boot sequence, which makes a manifest target for fault injection as explored in [23]. To the best of our knowledge, no published work showing a laser fault injection targeting an off-the-shelf SoC has been released.

Exploring this critical boot sequence allowed us to get a better understanding of the security protocols and how much they could be stressed. Although the hardware and software manipulations can be tricky, the outcome of breaking the secure boot sequence is worth the effort when considering the stakes of a successful attack: secure privilege escalation at bootloader level or signature forgery. It may turn out to be worthwhile for critical assets.

On one hand, the main purpose of this study is to form a better opinion about the risk that laser attacks may present and the corresponding impacts for embedded devices security. On the other hand, we wanted to assess how practical and difficult it would be to set up such an attack.

2 Gaining Access and Knowledge

The purpose of this section is to explain how the device under test (DUT) was altered for our experimental setup in order to be prepared for optical fault injection. After finding entry points, the reverse engineering phase led to valuable understanding of the architecture and discovery of a smart physical access method.

2.1 Low level Software and Debug Access

The first step towards our goal was to take control of the device at an early stage of the boot sequence. For this we had to exploit a software weakness which allowed us to flash part of the firmware without being detected by the TEE security. We were able to execute a custom firmware up to BL2, but BL1 was still heavily protected. It still gave access to debug functionalities, allowed for reverse engineering, and useful modifications of the boot sequence down the road.

2.2 Architecture Overview - Reverse Engineering

One indisputable factor of success during the sample device preparation was the software reverse engineering. We mostly used Hex-Rays IDA pro as static analysis tool for

firmware disassembly and decompilation, in cohesion with Zynamics' Bindiff, to easily compare functions reversed from other firmware versions or similar devices. The study proceeded in background by accumulation of small mechanisms understanding, lecture of the available documentation and substantive thinking.

We were able to dump every piece of code composing the boot sequence at runtime, bypassing encryption and memory protections. The firmware is composed of few hundred kilobytes of compact code and the overall program flow is simple - apart from the security perspective. This confirmed the initial statement: achieving software attacks on the boot sequence is very uncertain, considering the very low number of entry points and the architectural strength.

After rearranging the puzzle pieces, the objective was to understand the big picture (see Figure 1):

On reset (or power-on), the device sets up a basic environment to allow a safe execution of its boot sequence. For example, it sets up the different clock ratios, and the main processor clock at $1.4\,\mathrm{GHz}$.

Once the environment is stable, the SoC chooses its boot media and retrieves the firmware (FW) that is then loaded in its internal SRAM memory for verification.

The secure memory in which the FW is copied cannot be modified by the user at runtime. Thus, the CPU can safely validate the firmware integrity via a checksum and then performs the cryptographic verification. First the public key is authenticated with an Hash-based Message Authentication Code (HMAC) to make sure that it has not been modified. Then it can check the FW signature with this public key to confirm the code origin and author. Lastly, the now-trusted software can be AES deciphered with a hardware-backed master key in order to be, ultimately, executed. The overall design of the cryptographic system is robust and mastered.

These steps should be executed between each program block handover in order to build a chain of trust. Exploring different devices, we could see that the first stage is very well protected, but factually the further you progress in the chain of trust, the more these protections decline.

Noticeably, we never encountered any countermeasure against physical attacks, and worse, several weaknesses were found. From a functional point of view, the secure boot security can be bypassed by conditional "ifs" already hard-coded in the Read-Only Memory (ROM). These bypasses cannot be reached by software, as hardware fuses permanently disabled the program path. But without resilience, every physically-induced misbehavior during this boot sequence will result in totally different program flows, as further detailed in section 5 on attack scenarios. This is in direct contrast with the security architecture complexity, but makes us enthusiastic about the purpose of our study.

2.3 Physical Access

Generally, gaining a non-destructive physical access to a chip implies opening packages with mechanical or chemical preparation to remove the different layers of plastic, metal, or epoxy. Once uncovered, the chip may require additional preparation in case of sophisticated attacks. For instance bulk thinning with dedicated machinery enable to reduce laser dispersion when attacking from the backside.

For a mobile phone, the accessibility is different. When targeting the SoC, a simple hot air gun allows clearing direct access to the backside surface, taking apart the 3D package stack. But the main issue remains: The chip is located underneath its DRAM in a Package-on-Package (PoP) [24] assembly. In other words we need to somehow keep the device working while obtaining optical access for a laser pulse illumination.

The most generic solution is to relocate the DRAM with a dedicated PCB aside from the main die but it requires highly accurate routing and State-of-the-Art industrial technologies, as mobile devices are at the cutting edge of electronic integration expertise. In 2017, the development of such solutions was considered and quantified to roughly a month of development and production, several target boards, and a total cost around $\$5000$ per prototype, without any success guarantee. Such PCB might be mandatory on other devices, but for our target, we took a simpler exploration path that kept the attack cost as low as possible:

At this stage we had reverse-engineered enough information concerning the target smartphone boot sequence to know that it was possible to modify the hardware configuration at the bootloader level. As a result, we can take control of the executed code before most of the device initialization, in order to bypass the DRAM setup. Our code is thus located inside the SoC internal SRAM memory.

For our analysis, the device does not have to boot all the way to the Linux operating system or Android. The fault injection setup was focused on the secure boot sequence, and these modifications allow developing a complete software environment to test the security of the lowest cryptographic layers.

3 Fault Injection Setup

3.1 The Target

The most relevant specifications of our targeted System on Chip architecture are:

- Quad-core Cortex A9 @ 1.4 GHz - 0.71ns clock period
- 8-Stage pipeline with variable length, NEON extension
- 32 KB Instruction and Data L1 cache with 1 MB shared L2 cache
- Out-Of-Order execution and Dynamic Branch Prediction
- 64 KB ROM, internal 256 KB RAM, and eFuse key storage
- Hardware Crypto-engine, TrustZone Hardware
- Multiple clock and power domains over the SoC

Speculative execution, related to caches and dynamic code optimization make software fluctuating at runtime, and are very complex to analyze without precise knowledge of the architecture. But these observations do not apply during the boot, which is inherently predictable. However, the challenge lies in uncharted obstacles remaining: the

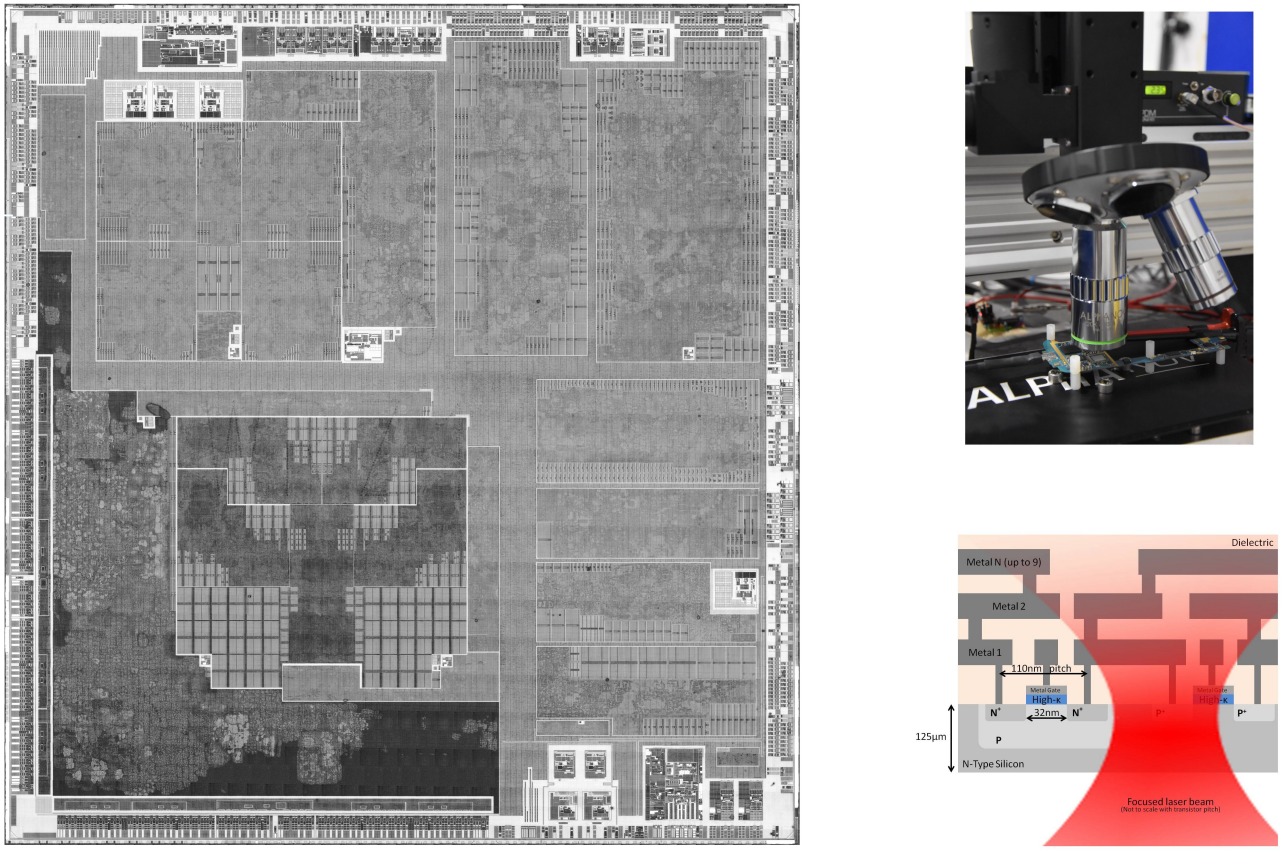

Fig. 2: (a) Backside infrared imagery of the SoC (150 MPixels) - (b) Fault injection setup - (c) Cross-section view of the DUT

clock speed, the pipeline architecture and the transistor density. Our concerns are more focused on the semiconductor processes at the CMOS 32nm High-K Metal Gate (HKMG) technological node.

Using the microscope station described below, we measured a silicon thickness of $125\,\mu\mathrm{m}$, which is thin enough to allow infrared imaging and provide a suitable laser penetration throughout the substrate. This avoids thinning a large die over $1\,\mathrm{cm}^2$ of surface. Backside IR imaging provided a good way to observe the semiconductor properties and the overall system architecture as depicted in Figure 2. Moreover, due to the high number of metal layers, at least 9, the backside approach turned out to be a mandatory choice for the infrared laser.

The manufacturer gives a gate pitch of $110\,\mathrm{nm}$, with an average density of $3\,000\,000\,\mathrm{gates/mm}^2$. A rough approximation of the number of logic gates in the ARM cores leads to 27 million, over a surface of $9\,\mathrm{mm}^2$.

3.2 Bench Hardware

We used a commercially available fault injection bench [18]. A single-mode infrared laser, emitting at $978\,\mathrm{nm}$ with Pulse-On-Demand capabilities at the nanosecond scale, is embedded within a dedicated microscope station. The maximum peak power is $2\,\mathrm{W}$. The optical system setup consisted of a laser microscope mounted on X, Y, Z translation stages and fixed on a marble arch for stability. We were thus able to scan the SoC entirely with the laser spot, keeping the sample stationary and did not meet any ergonomic or positioning issues.

The setup is able to reach sub-micrometer laser spot size on the surface of the sample with an adequate objective lens. We used a 20X microscope objective which allows for long working distances. We measured a $3.4\,\mu\mathrm{m}$ spot size at Full Width Half Maximum (FWHM) using the knife-edge technique. The laser spot size was probably slightly larger within the silicon. This is huge compared to the transistor size ($32\,\mathrm{nm}$ gate and $110\,\mathrm{nm}$ contact pitch at metal 1 level). A large spot size is not a problem when empirically scanning the device for fault effects. However, a smaller one helps to differentiate faults effects and localize more precisely areas of interest. Indeed, even when the spot size does not allow to target a single logic cell, it remains possible to inject single-bit faults in the chip as demonstrated by Agoyan [8]. We will describe this in the next section, on fault characterization.

As the substrate is thin and the technology is low power, it is natural to think that a physical disruption in the core logic is easily achievable. However, a high laser power precision is beneficial for two related reasons. First to generate more exploitable fault effects, as it is necessary to target the thin power band between an energy level without injected fault and a threshold where the device mostly crashes. And most importantly, to effectively increase the spot locality: factually, at the given gate density, a $10\,\mu\mathrm{m}$ spot covers 942 gates. However, when considering the Gaussian intensity distribution of the single-mode laser, 109 gates receive at least half the maximum intensity inside the $3.4\,\mu\mathrm{m}$ spot at

FWHM; and half the total energy is located in a $1.95\,\mu m$ spot, illuminating only 35 gates. With a single-mode laser, finding the critical energy required to disrupt the logic allows to refine the faulted area to much smaller dimensions, in the order of magnitude of the $32\,nm$ technological node.

The CPU is clocked at $1.4\,GHz$ during most of the boot sequence. This corresponds to a $710\,ps$ period. The laser jitter is very low ($< 8\,ps$) which is important for the repeatability of the tests. Most of our tests were performed using a specially made trigger setup which generated a significantly higher jitter of $3\,ns$, corresponding to 4 operations. With this setup, fine timed attacks have to be repeated several times to confirm a fault effect.

3.3 Software Setup

A dedicated software controls all the instruments used in the bench to automate the tests as much as possible.

We developed a bare-metal software environment at firmware level on the phone in order to configure the smartphone SoC and board, communicate with the main computer and execute our Function Under Tests (FUT). The device is set up to actively wait for a command, acknowledge its reception and immediately jump into the test loop. The laser pulse is then triggered on emission of a signal by the phone to eliminate as much jitter as possible between the laser pulse and the software execution.

Hardware modifications have been made to power-up the smartphone automatically via the programmable power supply, without human intervention, so that the software could decide to reboot the device when necessary.

An area to scan is selected with help of the infrared camera imaging. The position parameters on the X-Y-Z axis, the pulse width, the diode current and the pulse delay are all set respectively to a minimum value, a maximum value and a step size in the software. Then the program loops through each combination of these parameters.

For each sequence of tests, the initial answer of the DUT is stored in the computer as a standard answer, it depends on the internal SRAM reset values, and was reset after each reboot of the device. There are 3 cases for the response of a laser pulse:

- The device delivers the expected result, no fault is detected
- The device delivers no answer, which usually means it has crashed.
- The device delivers an answer different to the standard one, meaning that a fault was successfully injected in the chip.

In cases 2 and 3, the answer (or lack thereof) is logged in the file along with all bench parameters, and the device is rebooted. Then, the next combination of parameters is tested.

It is important to highlight that ARM exceptions provides fault resilient architecture to a certain degree: in our test FW, we implemented exception handlers from scratch and configured the device in order to log special events (data abort, prefetch abort, undefined instruction, etc). Nevertheless laser fault injection is able to put the device in an unreachable state where it generally crashes without logs.

Most parameters, except for the positions in X and Y, are either fixed or with a small number of iterations for a given test. After scanning roughly the selected zone, more tests can be done on areas where faults were successfully injected in the device, but with a smaller step size to better localize the interesting points.

3.4 Estimated durations

To start exploring the huge dimensional space of the experimental parameters we decided to have a simple coarse-fine approach, in which we proceeded to an overall scan in order to have an idea of the adequate energy levels and sensitive locations of the chip, as described in the next section.

The scanning process is very time-consuming, so the limits and step sizes must be considered carefully when initiating a sequence of tests. The delay between pulses is actually limited by the communication speed, which allows only 115200 characters per second. Indeed the more data you decide to log, the longer the tests will last. But the device also needs to be reset when it crashes, which takes approximatively two seconds.

To give the reader an order of magnitude, scanning over the $9\,mm^2$ CPU block with 10 values of pulse width and energy per location, will require 2.25 million pulses at $20\,\mu m$ steps, which is about 4 times the spot size; 20 million at spot size and 100 millions at half the spot size. Even with an average of 5 laser pulses per second, it represents dozens of weeks of experimental testing.

But still, the temporal exploration is not yet initiated. As the device is quite fast, even focusing 1 ms of the 500ms-long secure boot, corresponds to the processing of more than a million instructions...

4 Characterization of the Results

The misbehaviors appeared quickly and allowed to identify core 0, handling the boot sequence and all our functions under test.

The goal of characterizing the hardware behavior is to understand what will be possible to achieve with our setup, in term of software disruption, in order to later consider relevant attack paths. In this experiment we try to set aside the time dimension and observe the fault sensitivity only.

4.1 Surface Exploration

The Figure 3 shows the result of fault sensitivity characterization of a simple FUT: a loop counting until a given limit. Blue areas led to successful fault injection and in the orange area, the output can be considered as crash. This scan was performed with a hundred unique combinations of pulse width and diode current. In the end, each effect is stacked in order to conglomerate glow intensity. The resulting color corresponds to the likelihood of an effect over the range of assessed energies.

It delimits the area of the main integer pipeline and execution areas of tested instructions. Changing the FUT helps finding area of each instructions processing. For example, this can be done for floating point instructions to highlight the NEON pipeline, or with store/load instruction to highlight the memory pipeline.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2860010, IEEE Transactions on Computers

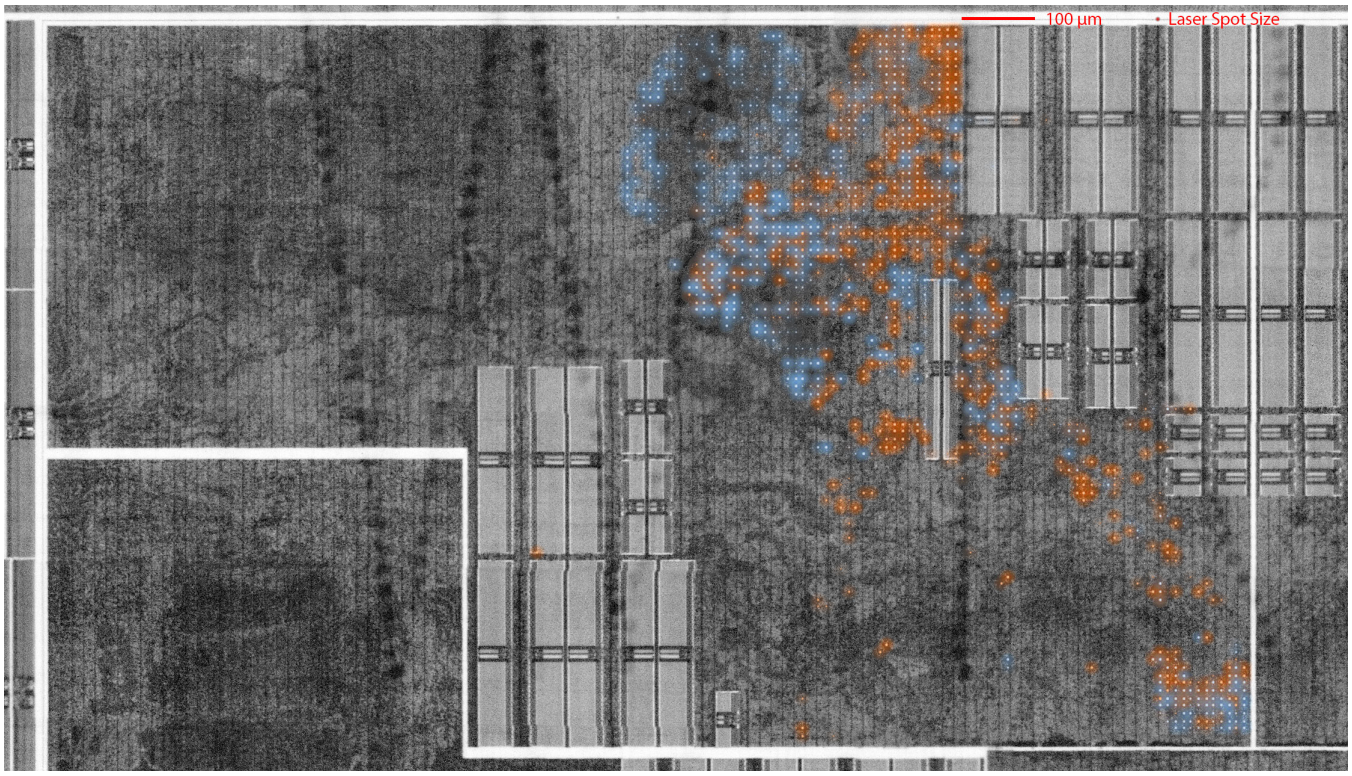

Fig. 3: Core0 sensitive areas

4.2 Crash Analysis

Depending on the areas, coarse scans resulted in between 5 to 40% effective fault injections. About a half of them can be considered as crashes.

Indeed the most common case is that the phone is left unresponsive during the logs, but we could identify 26% of the crashes as caught exceptions, probably caused by a bad pipeline feeding. In this case the device can recover from its faulty behavior and switch in exception mode to log valuable information about its internal state. The location of this crash effects highlights the areas of prefetching, instruction decode and data manipulation.

Over some regions, the chip behavior can be so unpredictable that some faults cannot be categorized. For example 4% of them contained glitches in the log sequence and required human analysis. Due to the high amount of data generated, and with an attacker state of mind, we decided to focus on simpler faults and discard complex/multiple or glitchy fault effects.

The remaining half of faulty behaviors, where the device did not crash and the log could be statistically examined, showed a considerable number of effects on the main register values. An exhaustive classification would allow to understand what is happening at architectural level through the 8-stage pipeline. However, this is hardly conceivable without dedicated tools such as an ARM in-circuit debugger and tracer. Unfortunately this requires additional privileges on the hardware, unavailable with our black-box approach.

4.3 Laser Focus

Initially, the Z parameter was fixed at a value where the laser spot size appeared to be minimum on the camera image. This is generally close to, but not the best value to concentrate the beam inside the silicon, at the transistor level. Using the Z-axis stage to explore this dimension is inconvenient as we loose the image focus, and only estimate if we are correctly focused. Moreover, the pulling force that separated the stacked packages, and natural imperfection of the silicon wafer backside, make the SoC surface uneven and concave. It is impossible to get the whole chip in focus, and takes time to have a correct flatness over a single CPU area. Thus, laser spot size will vary during scans, especially on such large areas.

To overcome this problem, we added an optomechanical system allowing to directly adapt the laser beam focus depth [19]. This installation allows to move a lens within the optical system, between the laser fiber and microscope objective. With $0.5\,\mu\mathrm{m}$ resolution for a course of $1.5\,\mathrm{mm}$, it allows to increase or decrease the laser spot size at will, without moving the microscope or the chip. This solution, combined with the analysis of images transmitted by the camera open the way to finer focus settings. Assuming that the thickness of the silicon layer remains the same, it gives the opportunity to correct the spot focus in situ, without loosing the camera focus on the transistors.

As can be seen in Figure 5, the number of effects linearly varies by a factor of 12 with lens moving over a $300\,\mu\mathrm{m}$ range. Additionally, the ratios between crashes, successful faults, exceptions and total number of effects stay roughly the same independently of the lens position. However, the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2860010, IEEE Transactions on Computers

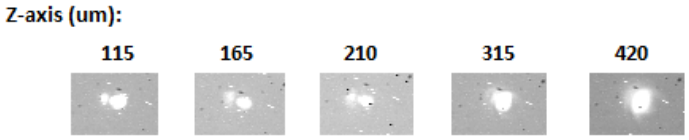

Fig. 4: Laser spot shape for different Z-axis offsets

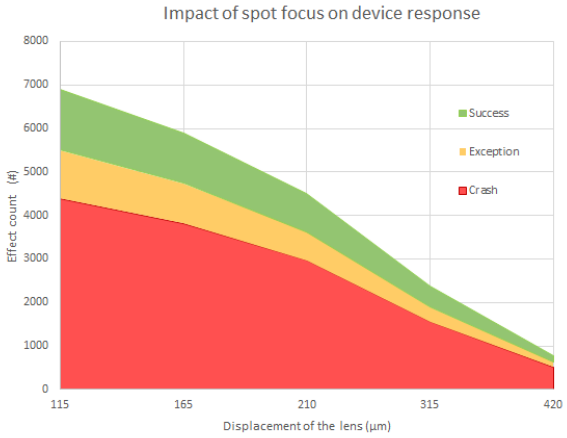

Fig. 5: Number of effects observed depending on the spot focus

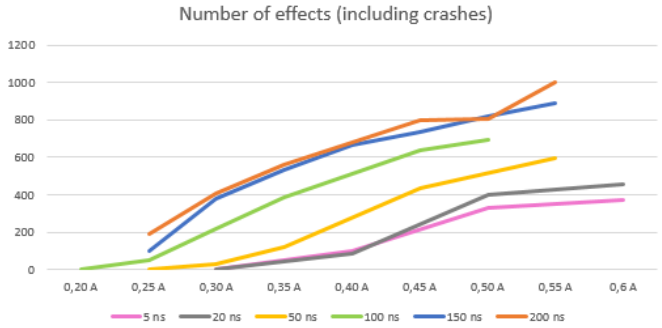

Fig. 6: Number of effects depending on the diode current and pulse width

spot aspect, visible in Figure 4 splits in two below 300 µm, because it is now possible to discern the spot reflecting on the transistors and on the substrate surface.

Before adding this optomechanical focusing system, an experimenter familiar with the laser bench had taken about half an hour to set a decent flatness and had settle a unique and small laser spot, corresponding to a lens offset around 380 µm. Only by adjusting the spot size inside the silicon we increase likelihood of effects, and successful fault injection, by a factor 5.

4.4 Laser Energy Sensitivity

Optimal parameters range could be guessed from these experimental scans, to perform finer scans around area of interest afterward, with the right laser pulse width and intensity. The idea behind this is to reduce parameter exploration to values where the pulse will most likely result in a fault.

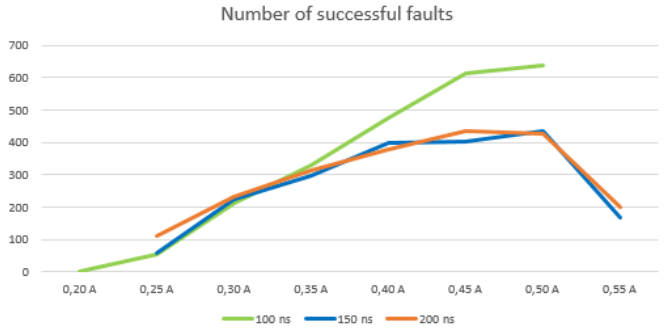

Fig. 7: Number of successful faults depending on the diode current and pulse width

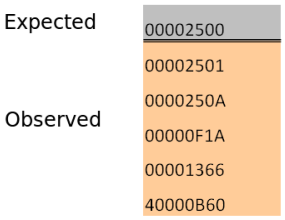

Fig. 8: Examples of dynamic fault injections on a loop counter, obtained with various experimental configurations

The Figure 6 illustrates how the laser energy impacts the number of effects observed over the whole core area. Surprisingly, the sweet spot of pulse width is around 100 times the clock period, way higher than expected.

We notice a saturation effect for pulse larger than 100 ns. A longer laser pulse will not affect the physical response of the chip. In particular, Figure 7 shows that above the critical pulse width, the number of successful fault is dropping significantly at high current, meaning that most pulses crashed the device.

It seems that a large pulse width allows for new effects to appear. At constant pulse energy, a longer pulse is less intense and thus spatially refined.

We assume that the accumulation of induced charges is able to inject faults even in part of the circuit that are not actively used. Thus, we separate two distinct fault models: dynamic fault, and static faults, that we are going to describe.

4.5 Dynamic Faults

In this work, dynamic faults refers to alterations occurring during the processing of a specific operation. Basically the fault effect can affect a given state (Instruction Queue, Decode, Renaming, Execute, Write Back) of the pipeline; causing the CPU to retrieve false operands, execute unexpected instructions or write erroneous results in a possibly different output register.

We witnessed a large number of these effects over the parametric space available. But as the time dimension is by far the richest one, mastering dynamic fault effects would have required advanced tests.

Expected	00000000	FFFFFFFF
Observed	00002000	FFFFEFFF
	00008000	FDFFDFFF
	00400000	FFFFFFFD
	00008000	FFFFF9FE
	00408000	00000000

Fig. 9: Examples of static fault injections over unused registers, obtained with various experimental configurations

For example, it is possible to characterize a short sequence of instructions from our firmware, searching for desired effect in the program output, expecting to obtain identical behaviors during the actual boot sequence. However, extra care must be given when extrapolating such tests, due to speculative execution. In such complex CPU architectures, it is inappropriate to characterize the chip behavior on a sequence of thousands consecutive additions unrolled and then generalize results to an equivalent for-loop. Because of the instruction cache, enabled as early as possible during the boot sequence, the branch prediction, or Out-of-Order execution, the target instruction processing might depend on what is executed before and after itself.

Dynamic faults results led to numerous attack paths. However, as the FW is compact, most of them required an instruction level fault, without impact on the surrounding instructions. Indeed the so called timed attacks are quite hard to implement and simpler attack scenarios, removing the time dimension were later intended.

4.6 Static Faults

On the contrary static faults happen when the laser is focused on an area that is not necessarily active during the pulse. It is quite difficult to validate the fault nature on such a complex architecture, but we witnessed a sufficient evidence: static faults can induce a bit-flip of currently unused registers at any given time of the processing. If the register is actively processed, there is a possibility of overwriting the fault effect.

As such, static faults differentiate themselves by their simplicity, as we loosen the temporal constraints almost completely; and their efficiency, as they represent a powerful tool for an attacker.

The examples given in Figure 9 show that most of the imaginable scenarios can happen. The precision of laser fault injection helps to separate bit-flip behaviors locally. Strangely in our experiments, faults often affected R6 and R7 general purpose registers, more than others.

We obtained a high repeatability of these fault effects. Mostly because the pulse width was way greater than the clock cycle and we were not bothered by timing considerations at all.

First, we noticed on Figure 10 the possibility to fault the Current Program Status Register (CPSR) which holds the comparison flags as well as the core configuration. Thus it was possible at any given time to:

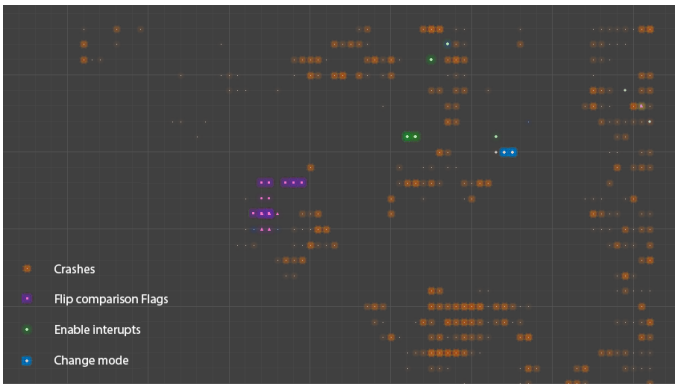

Fig. 10: Static faults on CPSR register

- Green dots: enable or disable interrupts (IRQ, FIQ and Imprecise Aborts)
- Blue dots: make the processor switch to another mode
- Purple dots: change the value of comparisons flags (Negative, Zero, Carry or Overflow) as well as Greater than or Equal (GE) flags of SIMD instructions.

On the Figure 10, orange glow represents the probability of crashes. We can see that CPSR faults are located in reliable areas and can be reproduced with accuracy. As individual bits of CPSR can be targeted, this can be exploited in several scenarios. However, the impact of a faulty behavior on comparison flags remains unknown, as the boolean logic might be processed Out-of-Order.

We believe that this kind of static fault injection can be done with every control register of the ARM core. The large amount of possibilities extends the experimental process with additional (and slow) logs.

5 Attack Scenarios

As we now have a clear understanding of our laser setup capabilities, it is possible to search the firmware for exploitable attack paths and challenge the platform security.

The Algorithm 1 gives an overview of the target code execution we want to disturb.

5.1 Scenario 1: Bypass

As mentioned in the section 2.2 on reverse engineering, several security bypasses are already present in ROM firmware. The most efficient one concerns the R4 register, loading a flag from e-Fuses to check if the secure boot is enabled or not. In this remarkable textbook case, any value other than '00000001' will bypass the secure boot protections totally.

This attack looks approachable. Indeed tampering with either of the three following operation would lead to a positive result: the flag reading, the integer comparison or the conditional branching. Looking at the previous characterization results, this should easily be achievable with a

Input: Null
Result: Jumping in first stage bootloader (BL1) only if it is valid and authentic

```
Reset: Hardware Init
...

R4 ← secure_boot_flag;
...

BL1_Copy( Bootmedia );
BL1_RSA_Getversion( );
BL1_Verify_Checksum( );

if R4 == 1 then
    BL1_Verify_Pubkey( );
    BL1_Verify_Signature( );
    BL1_Decrypt( );
end
BL1_Jump( );
```

Algorithm 1: Simplified ROM Boot Sequence

dynamic fault. Nevertheless, the attack remains challenging as it has two major temporal constraints:

- The laser needs to be precisely synchronized with the target software instructions, unwinding at around 3 every nanosecond.
- The pulse should keep a limited impact on the other instructions, as they are critical for the device initialization.

Keep in mind that typically, an instruction is processed in the pipeline for at least 8 clock cycles, hopefully softening a bit these constraints.

To time the attack, we used a traditional side-channel approach by measuring the power fluctuations at the chip level as shown in Figure 11. A first trigger is set to catch the reset glitch. The boot sequence being highly predictable and the related execution simple, the jitter remains low. However, the whole process spans over $500\,\mathrm{ms}$.

In order to get a better insight of the correspondence between power trace and ROM code execution, different firmwares were crafted and executed. For each, we knew where the boot sequence would stop, and recorded power traces where the device hang at the beginning of BL1 Copy, Getversion, and verifications of checksum, public key or signature (Figure 12). Thus efficiently delimiting each function call in the time.

However, the duration of attack experiments roughly synchronized with the instructions turned out to be prohibitive, mostly because we must reset the device after each pulse. Effectively we could reach a pulse per second, with millions of possible time offsets to test and the X-Y degree of freedom left.

A possible workaround would be to find the instruction cycle more precisely, with near-field electromagnetic side-channel analysis for example. But due to the architectural complexity of the core, it is near impossible to make a clear

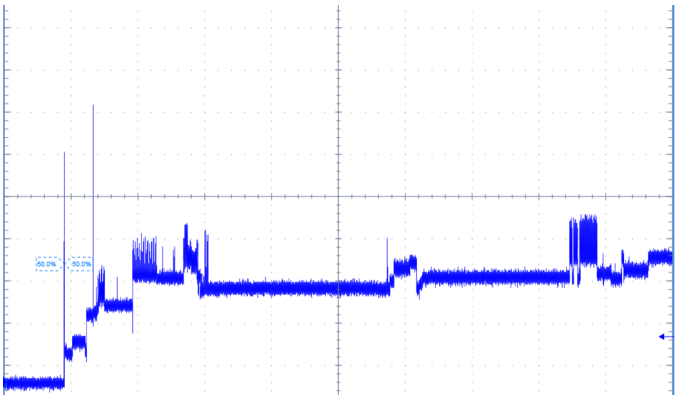

Fig. 11: Power consumption during the whole Boot - $500\,\mathrm{ms/div}$

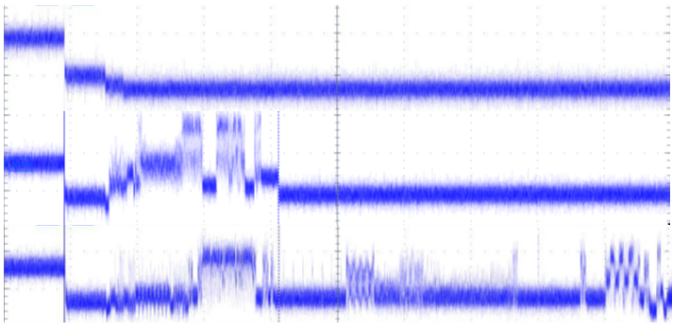

Fig. 12: Intentional software failure during - (top) BL1_Copy - (mid) BL1_RSA_Getversion - (bot) BL1_Checksum

correlation between clock cycles and instructions. They are fetched and issued in parallel, cached, possibly reordered or even stalling the pipeline during hazards...

Another possibility is to statically target R4 when it contains the secure boot flag instead of the sequence of instructions. In our test firmware, we found several small zones effectively changing the value of R4 when it is not used. But again, during the boot sequence, the flag is effectively exposed in R4 during 30 instructions, because functions calls stores it into the stack until its final usage in the 'if'.

In conclusion, we were not able to succeed in this attack path in spite of a high number of attempts. Indeed, we could not achieve the right combination of time delay and perturbation due to the dimensions of the exploration space.

5.2 Scenario 2: DeadLoop Escape

In order to avoid the pitfall of precise timing encountered in the previous scenario, unlikely to succeed in a limited amount of time, another approach was tried.

The idea behind this second attack scenario is to use static fault injection in order to get out of the infinite loop caused by any error in the boot sequence. This valuable path is left open because the device does not implement a safe and secure reset sequence when an error occurs, but instead, simply wait $7\,\mathrm{s}$ for the watchdog timer to do its job.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2860010, IEEE Transactions on Computers

Loading a non-encrypted software payload with a correct checksum will fail the signature verification. During this time, it is quite easy to disturb the infinite loop and hope to reach our malicious payload. As in paper [23], the difficulty lied in the ability to control the jump location. Fortunately, a predictable and elegant attack scenario exists: as shown in Algorithm 2 the ARM interruption handlers are not implemented, except for IRQ and FIQ. Even if the interrupts are disabled during the whole boot sequence, the ROM is still configured to redirect IRQ and FIQ to BL1 with an insecure jump. This major issue highlights that the firmware is not designed to be resilient against hardware attacks.

ARM Assembly code at physical address 0x00000000:

```
B reset
B .
B .
B .
B .
LDR  PC, BL1_IRQ_handler
LDR  PC, BL1_FIQ_handler
```

Algorithm 2: BL0 Interrupt Vector Table

As seen in the section 4.6 on static faults, ARM configuration registers are prone to bit-flips. Especially, it is possible to reliably reset bits 6 and 7 of CPSR (respectively enable FIQ and IRQ) at any given time in the boot sequence. Then, triggering an IRQ or FIQ in any way would bypass all cryptographic verifications and jump to our malicious firmware, leading to a successful attack.

We hoped to induce a second static fault in the hardware responsible of receiving interrupt requests inside the ARM core, finalizing the attack with 2 consecutive laser pulses performed before the watchdog timeout. Intensive scanning was undertaken to find this part of the circuit, and then extended a little bit to external peripherals and areas shared in common between cores, in search of the Generic Interrupt Controller (GIC) hardware block.

Unfortunately, we have not been able to successfully get the seond fault, but we are confident that it was a matter of time. Paradoxically, this scenario is substantially easier to achieve than the first one. In the meantime, we came across a simpler, single fault, attack scenario.

5.3 Scenario 3: SCR Fault Injection

The most critical register in the secure boot architecture is the Secure Configuration Register (SCR), which holds the privileges granted to that core. In particular, the Non Secure bit (NS), at location 0, is set to 1 to indicate that the core is in non-secure mode. Without hardware attacks in mind, a single bit is sufficient to hold against software attacks, but in practice, it turns out to be physically sensitive to static fault injection.

On Figure 13 green dots represent the probability to reset the bit 0 of SCR, blue dots represent bit-flips on the other bits of SCR and red dots crashes. We managed to obtain flipped

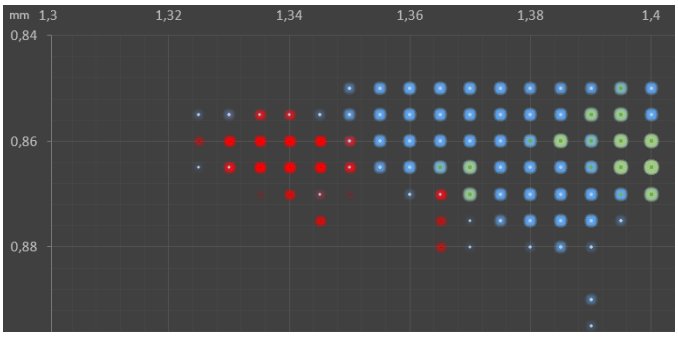

Fig. 13: Static faults on SCR register

TABLE 1: Repartition of more than 2000 observed SCR bit reset effects, from 0x3F value (all-1)

Bit reset count (#)	1	2	3	4	5	6
Occurences (%)	37.4	41.6	10.2	8.0	0.5	2.1

bits on the majority of positions in the register, even if this particular ARM architecture effectively uses only the 6 least significant bits. For instance, we were able to set the most significant bit to 1, which cannot be achieved by any CPU instruction as it is reserved for future use.

When focusing on these 6 operating bits, we obtain most of the time single or double bit effects, as shown in Table 1 for a bit-reset scenario (similar results are observed for a bit-set scenario). To be more generic, these tests could have been performed with several random initial values instead of all-0 and all-1 values...

Once the correct parameters were found, the NS (bit 0) reset fault can be done without crash, with a success rate of more than 95%. This fine scan allows to inject single bit flips in the SCR register with a large spot. The overall granularity of this scan is $5\,\mu\mathrm{m}$, which is about the spot size. Single and reproducible bit-flips are in the end achievable with a relatively large spot and very long pulses.

This latest result is particularly sensitive as faulting the mobile phone in the green area grants the highest privilege levels on the device, bypassing TrustZone architectural protections without having to understand the cryptographic implementation or achieve a fine timed attack. Indeed, static fault attacks are not limited to the context of secure boot, and might also be considered on a fully operational device. In such cases, the ability of faulting a CPU located under its DRAM remains an open question. But, even if the Package-on-Package is a major constraint in the context of mobile devices, this static fault attack scenario can emerge for IoT devices designed with separated DRAM chips.

6 Mitigations

We performed this attack on a widespread but specific device. Nonetheless, the ARM architecture dominates the mobile market and the targeted hardware differs little from latest technologies. We believe that same implementation issues and physical challenges will apply on future technological nodes such as $10\,\mathrm{nm}$, even if the transistor dimensions meets the atomic size. Especially for

the attacker, it is critical to ensure mastery of the laser beam shape and its energy distribution in the three dimensions. Right now we are getting to a point where this kind of attack scenarios became practicable and present a growing risk.

However mitigations techniques exist and have shown their efficiency in other industries, such as the smartcard or pay-TV. At the physical level first, it is possible to prevent or detect a fault injection by applying dedicated filters or sensors respectively. The drawback remains certainly the silicon surface to protect and therefore the cost it would incur. But simpler and affordable protections such as hardware redundancy or error correcting codes could easily be implemented for the core configuration registers, without exponential costs; and the ROM software in charge of the secure boot could be hardened by adding checks of the program flow and fault resilience.

With consciousness of the importance of the boot process, our opinion is to take a better care of this sequence regardless the complexity of physical attacks. Even if these protections are not at the state-of-the-art, it possibly might be enough to prevent exploitable scenarios. The difficulty lies on the ability to define where the protections should be implemented. The right balance between cost of the protections and security level can only be achieved by a good knowledge of the threats and possibilities offered by fault injection attacks, and their related countermeasures.

7 Conclusion

In this paper, we showed that despite its apparent complexity, a laser fault injection attack on a mobile phone embedding recent technologies, such as the Package-on-Package, is achievable. Targeting the secure boot sequence, we managed to compromise the security architecture with a laser pulse and consequently get code running with the highest privileges on the phone model. Getting this level of privilege grants rights over a number of secure resources in the phone, such as the TEE or the hardware cryptoprocessor.

This study confirmed that implementing such an attack requires to overcome a number of technical issues. First, an in-depth exploration and reverse engineering of the software layers and the related security mechanisms. But also physical access to the main processor, which is the most challenging issue, particularly because recent devices are highly integrated. However, all hardware modifications we applied could be undone.

The experimental setup allowed us to characterize the laser fault effects empirically. It will require further work to obtain precise information and build a fault model. Nevertheless we showed that the laser fault injection is capable of modifying the behavior of a SoC, and more particularly a high-speed Cortex A9, with various, numerous and repeatable fault effects. Among them, static fault effects detach themselves by their simplicity and efficiency within a real-world attack scenario.

Such a technique is an effective tool to explore in depth the core device security with a high success rate and the ability of a deep characterization. It does not appear relevant for large audience attack, but remains powerful for sophisticated attacks when targeting critical assets, performing forensic analyses, or simply evaluating the risk behind a security function.

References

[1] A. Barenghi, G. M. Bertoni, L. Breveglieri and G. Pelosi, *"A fault induction technique based on voltage underfeeding with application to attacks against AES and RSA"*, Journal of Systems and Software, 86. 1864-1878

[2] R. Anderson and M. Kuhn, *"Low Cost Attacks on Tamper Resistant Devices"*, Security Protocols, 5th International Workshop, Paris, France, April 7-9, 1997, Proceedings

[3] P. Maurine, K. Tobich, T. Ordas and P. Yvan Liardet, *"Yet Another Fault Injection Technique : by Forward Body Biasing Injection"*, YACC2012: Yet Another Conference on Cryptography, Sep 2012, Porquerolles Island, France.

[4] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson and E. Encrenaz, *"Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller"*, Proceedings of the IEEE, vol. 100, n111, pp. 3056-3076, Nov 2012.

[5] A. Barenghi, L. Breveglieri, I. Koren and D. Naccache, *"Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures"*, 10th workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC 2013, Santa-Barbara : United States

[6] S. P. Skorobogatov and R. J. Anderson, *"Optical Fault Induction Attacks"*, Cryptographic Hardware and Embedded Systems - CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers, B. S. Kaliski, K. Ko and C. Paar, ds., Berlin, Heidelberg, Springer Berlin Heidelberg, 2003, pp. 2-12.

[7] A. Morisset, *"Interaction laser-silicium et transport fibr pour le test de circuits intgrs par stimulation photolectrique non-linaire"*, Ph.D. dissertation, Universit Bordeaux 1, 2013. ¡NNT : 2013BOR14792¿¡tel-00920339¿

[8] M. Agoyan, J. M. Dutertre, A. P. Mirbaha, D. Naccache, A. L. Ribotta and A. Tria, *"How to flip a bit?"*, IEEE 16th International On-Line Testing Symposium, 2010.

[9] E. Biham and A. Shamir, *"Differential fault analysis of secret key cryptosystems"*, Proc. CRYPTO, 1997, pp. 513525.

[10] P. Loubet-Moundi, F. Olivier and D. Vigilant, *"Static Fault Attack on Hardware DES Registers"*, IACR Cryptology ePrint Archive (2011)

[11] C. Giraud, *"DFA on AES"*, Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers

[12] G. Canivet, P. Maistri, R. Leveugle, J. Cldire, F. Valette and M. Renaudin, *"Glitch and Laser Fault Attacks onto a Secure AES Implementation on a SRAM-Based FPGA"*, Journal of Cryptology, vol. 24, n12, pp. 247-268, 2011.

[13] C. Roscian, J. M. Dutertre and A. Tria, *"Frontside laser fault injection on cryptosystems - Application to the AES' last round -"*, IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2013.

[14] D. Boneh, R. A. DeMillo and R. J. Lipton, *"On the Importance of Eliminating Errors in Cryptographic Computations"*, Journal of Cryptology 14(2), pages 101119, 2001.

[15] R. Leveugle, P. Maistri, P. Vanhauwaert, F. Lu, G. D. Natale, M. L. Flottes, B. Rouzeyre, A. Papadimitriou, D. Hly, V. Beroulle, G. Hubert, S. D. Castro, J. M. Dutertre, A. Sarafianos, N. Boher, M. Lisart, J. Damiens, P. Candelier and C. Tavernier, *"Laser-induced fault effects in security-dedicated circuits"*, 22nd International Conference on Very Large Scale Integration (VLSI-SoC), 2014.

[16] M. Karpovsky, K. J. Kulikowski and A. Taubin, *"Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard"*, International Conference on Dependable Systems and Networks, 2004

[17] R. Torrance and D. James, *"The State-of-the-Art in IC Reverse Engineering"*, CHES '09 Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems

[18] Alphanov, *PDM+ (Pulse-on-Demand Modules) and S-LMS (Single Laser Microscope Station)*, http://www.alphanov.com/40-optoelectronics-systems-and-microscopy-single-spot-laser-station.html, [Online; accessed 2017]
[19] B. Chassagne, A. Morisset, L. Bon and S. Ermeneux, *Système et procédé de test de circuit intégré par faisceau laser*, Alphanov Patent Pending, 2017
[20] D. Shen, *"Exploiting Trustzone on Android"*, In Black Hat US, 2015
[21] Laginimaineb, *"Extracting Qualcomm's KeyMaster Keys - Breaking Android Full Disk Encryption"*, http://bits-please.blogspot.fr/2016/06/extracting-qualcomms-keymaster-keys.html, [Online; accessed 2017]
[22] N. Redini, A. Machiry, D. Das, Y. Fratantonio, A. Bianchi, E. Gustafson, Y. Shoshitaishvili, C. Kruegel and G. Vigna, *"BootStomp: On the Security of Bootloaders in Mobile Devices*, 26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.
[23] N. Timmers, A. Spruyt and M. Witteman, *"Controlling PC on ARM Using Fault Injection"*, Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), 2016.
[24] K. Gutierrez and G. Coley, *"PCB Design Guidelines for 0.4mm Package-On-Package"*, Application Report SPRAAV1B May 2009, and SPRAAV2A November 2013

Aurélien Vasselle has graduated in Integrated Circuit Design from Grenoble INP - Phelma. He joined eshard's team as hardware security researcher, with main focus on physical attacks on complex architectures. Involved in several projects of laser and electromagnetic fault injection on advanced targets, and concurrently preparing a Ph.D. thesis in side-channel on mobile, he developed a practical sense of smartphone security and strongly understands its stakes.

Hugues Thiebeauld is CEO and founder of eshard. He has more than 15 years of experience in the security field of embedded devices. He built solid skills in setting up practical laser fault injections and side-channel analyses on secure devices. His first works started in 2001 for a major smart card vendor in Europe, and subsequently he became security lab director for UL, Underwriters Laboratory. Hugues has been a strong contributor in the field, holds many patents and is the author of technical publications in cryptography side-channel, fault injection, software attacks. Hugues interest cover a large spectrum in the embedded security. With eshard, the company he created in 2015, he keeps extending further his skills in the fast moving and captivating technical environment of the embedded security.

Quentin Maouhoub was born in Rennes in 1991. After a preparatory class in Mathematics and Physics in the Lyce Chateaubriand, Rennes, France, he went on to ENSEIRB-MATMECA Engineering School, Talence, France, where he specialized in Computer Science. He joined ALPhANOV, Talence, France, in 2013 for his end-of-course internship, and has continued to work there since. He is involved in software/HMI development and conducting projects.

Adèle Morisset holds an Engineering Degree from Polytech' Orléans (France) and a BSc degree in Electronics and Photonics from Heriot-Watt in Edimburgh (Scotland). She joined ALPhANOV (Talence, France) in 2010. She then received her PhD degree on ultrafast optical developments for integrated circuit testing based on novel laser system architectures from the IMS laboratory (University of Bordeaux, France) in 2013. She is involved in conducting projects for the developments of opto-electronic systems at ALPhANOV and manages a group of technicians and engineers.

Sébastien Ermeneux graduated in Physical Engineering from INSA Rennes. He also holds a PhD in laser materials, and a degree as "senior officer" from ESCP Europe. He joined ALPhANOV in 2007 after successive positions as Product Manager at Highwave Optical Technologies, Project manager at Alcatel Optronics, and marketing manager at Eolite Systems. He is currently leading a Business Unit of $\sim$20 people developing innovative laser systems for several technology markets.