Combined fault injection and real-time side-channel analysis for Android Secure-Boot bypassing

Clément Fanjas, Clément Gaine, Driss Aboulkassimi, Simon Pontié, Olivier Potin

Combined Fault Injection and Real-Time Side-Channel Analysis for Android Secure-Boot Bypassing

Clément Fanjas[1], Clément Gaine[1], Driss Aboulkassimi[1], Simon Pontié[1], and Olivier Potin[2]

[1] CEA-Tech, Centre CMP, Équipe Commune CEA Tech - Mines Saint-Étienne, F-13541 Gardanne, France
Université Grenoble Alpes, CEA, Leti, F-38000 Grenoble, France
Email: firstname.lastname@cea.fr
[2] Mines Saint-Etienne, CEA, Leti, Centre CMP, F - 13541 Gardanne, France
Email: olivier.potin@emse.fr

Keywords: Secure-Boot · Synchronization · Frequency Detection · Fault Injection

Abstract. The Secure-Boot is a critical security feature in modern devices based on System-on-Chips (SoC). It ensures the authenticity and integrity of the code before its execution, avoiding the SoC to run malicious code. To the best of our knowledge, this paper presents the first bypass of an Android Secure-Boot by using an Electromagnetic Fault Injection (EMFI). Two hardware characterization methods are combined to conduct this experiment. A real-time Side-Channel Analysis (SCA) is used to synchronize an EMFI during the Linux Kernel authentication step of the Android Secure-Boot of a smartphone-grade SoC. This new synchronization method is called Synchronization by Frequency Detection (SFD). It is based on the detection of the activation of a characteristic frequency in the target electromagnetic emanations. In this work we present a proof-of-concept of this new triggering method. By triggering the attack upon the activation of this characteristic frequency, we successfully bypassed this security feature, effectively running Android OS with a compromised Linux Kernel with one success every 15 minutes.

1 Introduction

Hardware attacks such as Side-Channel Analysis or Fault Injection represent an important threat for modern devices. An attacker can exploit hardware vulnerabilities to extract sensitive information or modify the target behavior. Among hardware attacks, there are two main kinds of attacks which can exploit a physical access to the target:

– Side-Channel Analysis (SCA) relies on the fact that data manipulated by the target can leak through a physical channel like power consumption or

Electromagnetic (EM) emanations. By performing a statistical analysis, an attacker may retrieve these data.
– Fault Injection attacks aims at disrupting the target during the execution of sensitive function. There are multiple Fault Injection methodologies, including optical injection, Electromagnetic Fault Injection (EMFI), voltage and clock glitching or body biasing injection.

For both Side-Channel Analysis and Fault Injection, the attacker needs the best possible synchronization in order to capture the data leakage or disrupt the target behavior. This synchronization issue is even more present on modern SoCs which are more complex than traditional micro-controllers. This paper introduces a new method called Synchronization by Frequency Detection (SFD). This method is based on frequency activity detection in a side-channel as a triggering event. The name of the designed tool to implement this method is the frequency detector. It should not be confused with a frequency synchronizer which is a component used in Software Defined Radio (SDR). A passive probe captures the EM emanations from the target. Then an SDR transposes around 0 Hz a selected frequency band of 20 MHz located between 1 MHz and 6 GHz. The transposed band is then sampled and transmitted to an FPGA which applies a narrow band-filter to isolate a specific frequency. The system output is proportional to the energy of a frequency selected by the user in the target EM emanations. The whole process is performed in real-time. We validate this method by synchronizing an EMFI during the Android Secure-Boot of the same target as [GAP+20] which is a smartphone SoC on developpment board with 4 cores 1.2-GHz ARM Cortex A53. The technology node of this SoC is 28nm litography process. We identified a critical instruction in the Linux Kernel authentication process of the Secure-Boot. Then we found a characteristic frequency that only appears before this instruction. Synchronization of hardware attacks is an old issue which can be hard to overcome with no possibility to use the target I/Os. By using the occurrence of this frequency as triggering event, an EMFI was successfully synchronized with the vulnerable instruction, resulting in the bypass of the Linux Kernel authentication without using any I/Os.

This work provides several contributions:

– We present a novel synchronization methodology. This new method allows to trigger hardware attacks on complex target such as high speed System-on-Chip without using I/Os. The SFD concept could be used in black or grey box context.
– To our knowledge, we present the first successful bypass of a System-on-Chip Secure-Boot using an EMFI. This is also one of the first hardware attack on System-on-Chip which combine SCA and FI. This attack has a high repeatability rate, with one bypass every 15 minutes.
– Although our target is a developpment board, our usecase is still realistic since the targeted System-on-Chip is used in smartphones, and the targeted software is the Android Bootloader.

Section 2 provides the related works in term of synchronization methods and Secure-Boot attacks. Section 3 focuses on the frequency detector system. Sections 4 and 5 describe respectively with the attack set-up and the final experimentation. Finally, the section 6 and 7 are dedicated for discussion, conclusion and perspectives.

2 Related Works

2.1 Bootloader attacks

Other works focus on Secure-Boot attacks which permit a privilege escalation. In [BKGS21] the authors explain how to successfully re-enable a hidden bootloader by using a voltage glitch. This bootloader was destined for testing purposes and grants high privileges to its user. A similar attack is performed in [CH17], an EMFI corrupt the SoC DRAM during the boot process, which causes the bootloader to enter in a debug state. Then the authors use a software exploit to gain privileges over the TEE. [TSW16] demonstrates how to load arbitrary value in the target PC. It describes a scenario using this vulnerability to bypass the security provided by the Secure-Boot and execute a malicious code on the target. In [VTM+18], the authors provide a methodology for optical fault injection on a smartphone SoC, targeting the bootloader.

2.2 Attack synchronization issue

The issue related to the synchronization of hardware attacks has already been identified in several works such as [MBTO13, GAP+20]. The delay between the triggering event and the occurence of the vulnerability needs to be as stable as possible to maximize the attack success rate. The variation of this delay is called jitter, it is highly correlated with the target complexity. Modern architectures make use of optimizations such as speculative execution and complex strategies of cache memory management. Although providing high performance, these mechanisms bring highly unpredictable timing, which may cause an important jitter with the accumulation of operations. One of the most commonly used techniques to overcome the synchronization issue is based on the I/Os target exploitation to generate the trigger signal [BKGS21, TSW16, MDH+13, RNR+15, DDRT12, SMC21, GAP+20]. This method minimizes the amount of operations performed by the target during the interval between the triggering event and the vulnerability. Therefore it reduces the jitter associated with these operations. However other triggering methods are required in scenarios where self-triggering is not possible or in which the synchronization quality is not enough.

2.3 Existing synchronization methods

One alternative to self-triggering is to use essential signals to the target, such as reset or communication bus [BKGS21, SMC21, VTM+18]. However these signals are not always available or suitable for triggering purpose.

Side-Channel attacks require aligned traces to perform an efficient statistical analysis to retrieve the secret information manipulated by the target. Traces alignment can be done during the attack with an online synchronization, but also with signal postprocessing [DSN+11]. In [CPM+18] the authors presented an offline synchronization method to pre-cut traces before alignment[3]. They made only one capture using an SDR during the execution of several AES encryption. They identified a frequency in the orignal capture, which only appears before the AES encryptions. By cutting the original capture accordingly with this frequency occurence, they were able to extract each AES and roughly align the traces in post-processing. This method is close to the SFD method presented in this paper. The main difference is that [CPM+18] presents an offline method which is dedicated for side-channel analysis, whereas the SFD method is an online method, which is also efficient for fault injection.

[HHM+12] presents an attack based on the injection of continuous sinusoidal waves via a power cable to disrupt an AES implemented on an FPGA. This attack does not need synchronization since it is a continuous injection that affects the target during all the encryption. However in [HHM+14] the same authors present a method to improve this attack by injecting the sinusoidal waves only during the last round of the AES. The injection is triggered after the occurence of an activity in the target EM emanations.

In [VWWM11] the authors use a method called "pattern based triggering" to synchronize an optical fault injection on a secure microcontroller. It consists in a real-time comparison of the target power signal with a known pattern which appears before the vulnerability. The FPGA board sampling frequency is 100 MHz. However there can be some higher frequency patterns between two samples. To detect these patterns, the system uses a frequency conversion filter which outputs an envelop of the target power signal. Although being at a lower frequency, this envelop signal represents the high frequency pattern occuring between two samples. The concept of "pattern based triggering" or "pattern matching" is explored in details in [BBGV16], the authors also propose a method based on an envelop signal to represent high frequency pattern in lower frequency. The pattern based triggering method feasibility is strongly correlated with the sampling rate of the monitored signal. Therefore this method is not suited for triggering attacks on target such as SoC which runs at high frequency (i.e. GHz level) unless there is a frequency conversion method between the input signal and the pattern comparison. An alternative would be to use ADC with high sampling rate, however it means that the processing behind the ADC needs to be able to handle more data.

3 Frequency detector

This section proposes a new device for synchronizing hardware security characterization benches. The goal of this tool is to perform real-time analysis of the

[3] This method is partially inspired by https://github.com/bolek42/rsa-sdr which is an offline synchronization method to align SCA traces

EM activity allowing the detection of events that happen inside the SoC. Targeted specifications for the frequency detector are listed bellow as requirements:

- R.1 Be able to detect an event that happens as close as possible to the execution of an instruction vulnerable to fault injection. The number of instructions between the detected event and the vulnerable instruction should be limited to minimize the temporal uncertainty of the vulnerable instruction execution.
- R.2 The event must be detected before the fault injection vulnerability. Due to causality issue, the fault injection setup must be triggered before the EM shot. Moreover, the use-case studied in this paper is more restrictive because our fault injection setup must be triggered at least 150 ns before. For SCA use-cases, the causality constraint is relaxed because an oscilloscope can record the past.
- R.3 Provide a real-time detection. The delay between an event and its detection must be the most constant as possible. This delay is the latency of the EM activity analysis. In practice, this delay will not be constant. The variation of this delay is the temporal uncertainty inserted by the detection operation. To minimize this delay, implementation of this operation must fit a real-time constraint. For example, using a classical computer to perform the detection would insert too temporal uncertainty to fit this constraint. For SCA use-cases, an offline post-treatment can improve synchronization of side-channel traces if the Signal to Noise Ratio (SNR) is acceptable. This post-process relaxes the temporal uncertainty constraints but can not be applied in fault injection use-cases.
- R.4 The kind of event EM signature is specific to the use-case. The requirement for this study is to be able to detect the computation of cryptographic operations executed before, during, or just after a RSA signature check. For example, long-integer arithmetic or hash computation can be targeted. Requirements for the SoC Secure-Boot use-case are:
  - R.4.1 To be able to detect repetitive events.
  - R.4.2 To be able to detect events that require few microseconds of computation as long-integer arithmetic or hash. The duration of events detected in this work is 20 μs.
  - R.4.3 To be able to detect event from the EM activity of an high speed SoC. Knowing that the CPU of the studied SoC runs at clock rates between 800 MHz and 1.2 GHz.

In the next, methodological and technological choices are detailed and the performances of the frequency detector are evaluated.

3.1 Frequency detection methodology

We consider that using an FPGA to perform analysis can meet the real-time constraint R.3. However, this analysis must be able to detect events in a large bandwidth signal (R.4.3). High speed Analog to Digital Conversion (ADC) and the data analysis from high sampling rate signal represent a challenge. To improve the feasibility of our solution, we associated the FPGA with an SDR. As

illustrated by the Figure 1, our tool is based on a passive probe and an amplifier to measure the EM emanations from the target. The SDR shifts one frequency range of this signal to the baseband. The SDR outputs are transmitted to the FPGA. These signals are an image of 20-MHz band selected by the user in the RF signal spectrum. This solution is simpler with an SDR because the FPGA can perform analysis at low sampling rate signals (f_s = 20 Msamples/s). It is compatible with requirement R.4.3 because these signals are an image of an high frequency band.

The choice of SDR output sampling rate is a trade-off. Designing an implementation that meets the real-time constraint R.3 is easier with a smaller sampling rate. Nevertheless, low temporal resolution increases the temporal uncertainty.

Regarding the requirement R.4.1, limiting the detection to repetitive events is acceptable. Execution of loops emits EM activities. If the loop step is regular and associated to a timing period T_{loop} then the EM emanations should be significant for frequency $F_{loop} = \frac{1}{T_{loop}}$ and its harmonics. Observing EM activity of a SoC around a specific frequency should allow to detect a repetitive event.

In this work, we propose to monitor in real-time the activity in a narrow band around a characteristic frequency in order to trigger the fault injection. The frequency is selected by the user, it should be characteristic from a loop in the code executed by the target a short time before the vulnerability. The methodology to identify a characteristic frequency is described in section 4.4. A digital signal processing is performed by the FPGA on SDR output signals in order to focus on the activity in a narrow band. To select this band, a pass-band filter is implemented in the FPGA. This digital filter has a fixed band. Combining the SDR and this filter is equivalent to a high frequency band-pass filter. An user can control the central frequency of the band by configuring the SDR. Digital processing offers the possibility to design more selective filters than analog filters. To respect R.1 and R.2 requirements, the delay introduced by the signal processing in the detection is limited to 3.2 μs by using 64^{th}-order filters. The goal is to only detect the targeted event to avoid false-positive. The output of the frequency detector is an image of the power in a narrow band around the frequency selected by the user in the EM emanations. This image is sent to a Digital to Analog Converter (DAC).

3.2 Frequency detector design

In this section, we describe the design of the frequency detector. The two main components are an SDR and an FPGA. The Figure 1 illustrates the system.

The SDR is a HackRF One from Great Scott Gadgets[4]. It has a half-duplex capability but its reception mode is only used in this work. The input is a RF signal and the output is pair of 8-bit samples to be sent to a computer through an USB connection. In our work, the input is an amplified image of the electromagnetic activity of the SoC target. The output is composed by two sampled signals.

[4] HackRF One: https://greatscottgadgets.com/hackrf/one/

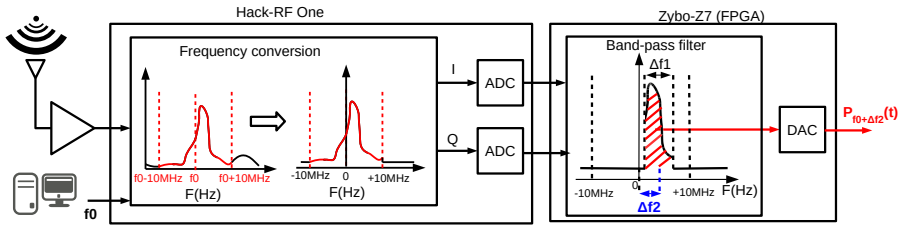

Fig. 1. Block diagram of the frequency detector

These signals are result of an IQ demodulation. User defines a f_0 frequency between 1 MHz and 6 GHz. An analog f_0 sinusoidal signal from an oscillator is mixed to the RF signal with a multiplier to generate I (superheterodyne receiver). I is the *In phase* signal. It is sampled by an ADC after a Low-Pass Filtering (LPF). The mixer shifts the RF signal from one frequency range to another. For example, mixer shifts activity at frequency $|f_{RF}|$ to activities at $|f_{RF} + f_0|$ and $|f_{RF} - f_0|$ (1).

$$\sin\left(2\pi f_{RF}t\right).\sin\left(2\pi f_0 t\right) = \frac{1}{2}\cos\left(2\pi(f_{RF}-f_0)t\right) - \frac{1}{2}\cos\left(2\pi(f_{RF}+f_0)t\right) \quad (1)$$

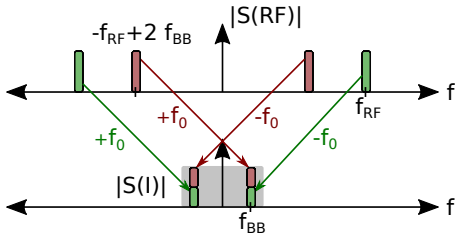

Fig. 2. Issue of image frequency

When only signal I is used, there is a an issue of image frequency. Figure 2 illustrates this problem. If $|f_{RF}|$ is shifted to $|f_{BB}|$ in the baseband with $f_0 = f_{RF} - f_{BB}$, then the image frequency $|-f_{RF} + 2f_{BB}|$ is also shifted to $|-f_{RF} + 2f_{BB} + f_0| = |f_{BB}|$ and produces interference. To solve this issue the signal $I+j.Q$ must be considered instead of only I. A sinusoidal signal with a 90° phase shift is mixed to the RF signal with a multiplier to generate Q. Q is the *Quadrature* signal. Sampled signal I and Q are sent to a computer. In classical application of an SDR, a software signal processing is performed on $I + j.Q$ to demodulate this shifted RF signal. As explained in section 3.1, I and Q samples must be sent to an FPGA. Therefore, we modified the SDR. In an HackRF One,

a Complex Programmable Logic Device (CPLD) gets I and Q samples from the ADC and transmits them to a micro-controller. This micro-controller sends them to a computer through an USB connection. The signal processing can not be performed by the CPLD because it has not enough logic-cells. The CPLD bitstream has been modified to also send I and Q samples to a PCB header. Our patch is publicly available[5].

A Digilent Zybo-Z7 board was connected to this PCB header of the HackRF One to receive these samples. I and Q signals sampled at 20 Msamples/s rate are received by the programmable logic of the Xilinx Zynq 7010 FPGA. This complex signal $I + j.Q$ is an image of a 20-MHz bandwidth around the user-defined frequency f_0.

The requirement is to select only activities of a small bandwidth. To fit this constraint, a narrow band-pass filter was designed. This digital filter will be used by the FPGA to filter I and Q samples. DC offsets in radio system is a known issue [Abi95]. Therefore, the band-pass filter has been centered around the 8-MHz frequency to avoid the system output to be impacted by a ghost DC offset. The band-pass filter was designed as an one-side filter. The filter should select frequencies around 8 MHz and attenuate frequencies arround -8 MHz. This is possible because the FPGA can discriminate negative and positive frequencies in $I + j.Q$.

The digital filter was designed in two steps. The first step consist in designing of a low-pass filter. Targeted characteristics are: a 5-kHz pass band, a cut after 750 kHz, and a 64^{th}-order. The MathWorks Matlab tool was used to design a linear-phase Finite Impulsion Response (FIR) filter targeting the characteristics. The effective bandwidth of the designed low-pass filter is 209 kHz. Equation (2) describes the low-pass filter $H_a(z)$ in the Z-domain. The 65 coefficients a_i are the output of the filter design and are real double values.

$$H_a(z) = \sum_{i=0}^{64} a_i z^{-i} \tag{2}$$

The second step is the design of two complex filters from $H_a(z)$. Shifts (3) and (4) were used to design filters $H_0(z)$ and $H_{\frac{\pi}{2}}(z)$.

$$H_0(z) = \sum_{i=0}^{64} b_i z^{-i} = \sum_{i=0}^{64} a_i . e^{j2\pi(i+1)\frac{8\ MHz}{f_{sampling}}} . z^{-i} \tag{3}$$

$$H_{\frac{\pi}{2}}(z) = \sum_{i=0}^{64} c_i z^{-i} = \sum_{i=0}^{64} a_i . e^{j2\pi(i+1)\frac{8\ MHz}{f_{sampling}} + j\frac{\pi}{2}} . z^{-i} \tag{4}$$

Equation (5) and (6) show how to filter input $I_{in} + j.Q_{in}$ to compute $I_{out} + j.Q_{out}$ with b_i and c_i the coefficients from (3) and (4) respectively.

$$I_{out}(n) = \sum_{i=0}^{64} Re(b_i).I_{in}(n-i) - \sum_{i=0}^{64} Im(b_i).Q_{in}(n-i) \tag{5}$$

[5] HackRF One, CPLD patch: https://github.com/simonpontie/hackrf_cpld_patch/

$$Q_{out}(n) = \sum_{i=0}^{64} Im(c_i).Q_{in}(n-i) - \sum_{i=0}^{64} Re(c_i).I_{in}(n-i) \tag{6}$$

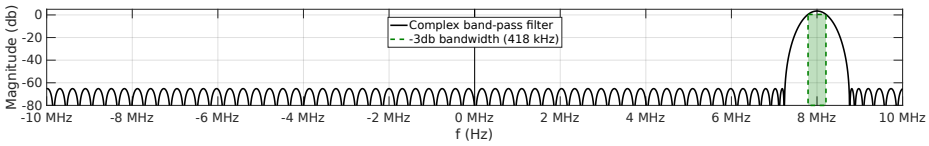

Fig. 3. Complex band-pass digital filter

Figure 3 shows performances of the filter. It is an one-side digital filter around 8 MHz (Δf_2 in figure 1) with a 418-kHz (Δf_1 in figure 1) bandwidth. This signal processing is composed by four FIR filters with real coefficients: $Re(b_i)$, $Im(b_i)$, $Re(c_i)$, and $Im(c_i)$. FIR filters were quantified and implemented with the "FIR compile" tool from the Xilinx Vivado tool suite. A hardware implementation was designed to use these filters and to compute I_{out} and Q_{out}. An image of the $I_{out} + j.Q_{out}$ power is approximated by implementing the equation (7).

$$\widetilde{P}(n) = I_{out}(n)^2 + Q_{out}(n)^2 \tag{7}$$

$I_{out}+j.Q_{out}$ is an image of a frequency band between $f_0+8\ MHz-209\ kHz$ and $f_0+8\ MHz-209\ kHz$ of the RF signal. Because it is a narrow band, the RF signal might be regarded as a sinusoidal signal RF_a (8). The power of RF_a is $\dfrac{A_a^2}{2R}$ (9). Equations (10), (11), and (12) show $\widetilde{P}$ as an image of the RF signal power with this sinusoidal assumption.

$$RF_a(t) = A_a sin\Big(2\pi(f_0+8\ MHz+f_a)t+\phi_a\Big), \quad f_a \in [-209\ kHz, 209\ kHz] \tag{8}$$

$$P_{RF_a} = \frac{\Big\langle A_a^2 sin^2\big(2\pi(f_0+8\ MHz+f_a)t+\phi_a\big)\Big\rangle}{R} = \frac{A_a^2}{2R} \tag{9}$$

$$I_{out}(t) = RF_a(t).\sin\Big(2\pi f_0 t\Big) \simeq \frac{A_a}{2}\cos\Big(2\pi(8\ MHz+f_a)t+\phi_a\Big) \tag{10}$$

$$Q_{out}(t) = RF_a(t).\sin\Big(2\pi f_0 t+\frac{\pi}{2}\Big) \simeq \frac{A_a}{2}\sin\Big(2\pi(8\ MHz+f_a)t+\phi_a\Big) \tag{11}$$

$$\widetilde{P}(t) \simeq \frac{A_a^2}{4}\Big(\cos^2\big(2\pi(8\ MHz+f_a)t+\phi_a\big)+\sin^2\big(2\pi(8\ MHz+f_a)t+\phi_a\big)\Big) = \frac{A_a^2}{4} \tag{12}$$

An uncontroled ϕ_a can delay the time between activation of the frequency and when the output is maximal. IQ demodulation is important because output of our system will be the same regardless of the ϕ_a value (12).

The approximated image of the power in the narrow band (7) can be efficiently computed because it only requires two squares and one addition. This signal $\widetilde{P}$ is sent to a R-2R DAC to be converted as an analog signal. This analog signal is an approximated image of the power in the RF signal between $f_0 + 8\ MHz - 209\ kHz$ and $f_0 + 8\ MHz + 209\ kHz$. By controlling f_0, an user can observe an approximated image of the power in a 418-kHz band chosen in [8 MHz, 6 GHz]. In the next we continue to use the sinusoidal assumption, thus we refer to this narrow band as a frequency.

3.3 Frequency detector performances

This system red is able to detect the activation of a specific frequency between 8 MHz and 6 GHz. The system output is updated by the frequency shift and the signal processing. These operations are stream processes but require a delay to propagate information from the input to the output. To characterize this latency we used a Low Frequency Generator (LFG) to generate an Amplitude-Shift Keying (AFK) modulated signal with a carrier frequency F_i. It is emitted by a probe situated near the probe of the frequency detector. The frequency detector has been set to trigger upon the F_i frequency activation. An oscilloscope is used to measure the delay Δt between the frequency activation (modulation signal) and the output of our system. The standard deviation $\sigma_{\Delta t}$ of this delay corresponds to the jitter induced by our system. The oscilloscope triggers upon the rise of the frequency activation signal. The delay is measured between the oscilloscope trigger time and when the frequency detector output exceed 50% of its maximal value. Several measures (within ten thousand) have been performed. The mean time is equal to an average $\langle \Delta t \rangle$ of 2.56 μs and its standard deviation $\sigma_{\Delta t}$ is equal to 60.9 ns. The standard deviation $\sigma_{\Delta t}$ fits the requirement R.3 because 60.9 ns is a temporal uncertainty close to the temporal resolution. This resolution is 50 ns and it is corresponding to the software radio sampling period of the I/Q signals. 95% of the value belongs to an interval of $\Delta t_{mean} \pm 2\sigma$ which corresponds to the interval $[2.44\mu s; 2.68\mu s]$. To fit the causality requirement R.2, user must explore only characteristic frequency of events that happen $2.83\mu s$ ($\simeq 2.68\mu s + 150\ ns$) before the targeted vulnerability. In addition, the low-pass filter introduced by the DAC limits the minimum period of detectable activity. The frequency activity needs to stay active long enough to let the frequency detector output rise to the desired level. The measured rise-time value for the frequency detector output (between 5% and 95%) is 922 ns. We measure Δt as the delay between the rise of the frequency activation signal and the rising edge of the frequency detector output at 50% of the maximum value. Thus the frequency needs to stay active at least 461 ns (i.e. 50% of the rise-time) in order to be detected. This fits the requirement R.4.2.

4 Attack environment setup

The target used in this work is already described in section 1. [GAP+20] presents a methodology to bypass one of the security mechanisms of Linux OS by targetting a specific core with a clock frequency fixed at 1.2 GHz. The fault model proposed by the authors is based on instruction skipping. This paper reproduces a similar experience with three main differences:

- The software targeted is the Android Secure-Boot.
- The core targeted is different.
- The frequency during the boot phase is set to 800 MHz.

This section describes preliminary experiments to tune fault injection and SFD setups. This exploration includes the search of an EMFI vulnerability in the Android Secure-Boot. For the EM analysis we used a probe (RF-B 0.3-3) and a preamplifier (PA303/306) from Langer. The injection probe is based on the same design as describe in [GAP+20]. To move the probe at the SoC surface, we used an XYZ motorized axis from Owis.

4.1 Electromagnetic Fault Injection

A pulse generator delivers a pulse up to 400 V into an EM injection probe. The target communicates with a host PC by UART. The PC configures the pulse generator voltage and controls an XYZ motorized stage to move the probe at the chip surface. The purpose of this experiment consists in characterizing the EMFI regardless of the triggering method. Therefore, a target with the Secure-Boot disabled was used to validate fault injection experiments on a fully controlled software code. The code used to observe the fault injection effect is composed by a sequence of SUB instructions, which are surrounded by GPIO toggles. This program has deterministic inputs and outputs in a scenario without injecting faults. The GPIO triggers the EMFI during the SUB sequence. The results are sent by the target through the UART bus. These results are compared with the expected value to determine whether a fault has been injected. This experiment is repeated 50 times for each position of the probe with a step of 500μm between two positions. We scanned all the chip which corresponds to an area of 13.5mm by 11mm. The results of the global scan is superimposed on the chip IR imaging as shown in figure 4 (A). We observe the presence of faults in a small area. Consequently, a more accurate scan of this faulty area with a small step of 50μm was performed to identify the best position. This scan result is shown in figure 4 (B). The best fault rate was achieved with a 400 V pulse voltage.

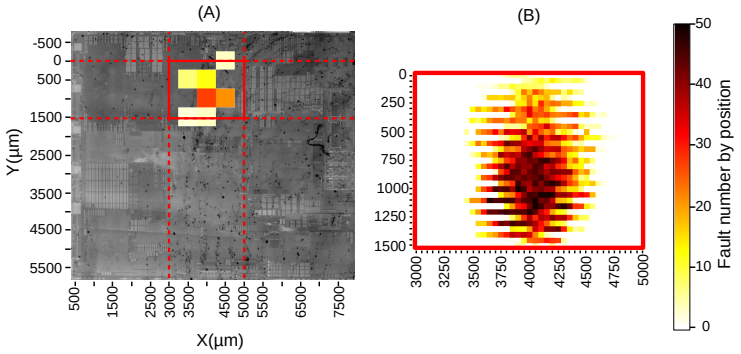

Fig. 4. Fault injection sensitivity scan for 400 V pulse.

4.2 Electromagnetic leakage measurement

A scan of the chip was performed during the execution of a code leaking at a predetermined frequency. The power around frequency was measured for each position above the chip. This experiment was also applied on the other side of the PCB, above the decoupling capacitors. Eventually, the best passive probe position appeared to be above one of the decoupling capacitors.

4.3 Secure-Boot vulnerability

The Secure-Boot is a crucial security feature in a mobile device. It ensure that the running OS can be trusted. It is a chain of programs loaded successively in memory. There is an authentication of each program before executing it to ensure that it is legitimate. In our experimentations, we used a development board with a partially enabled Secure-Boot to start Android. Figure 5 describes the Secure-Boot architecture implemented on our target. The First Stage Boot-

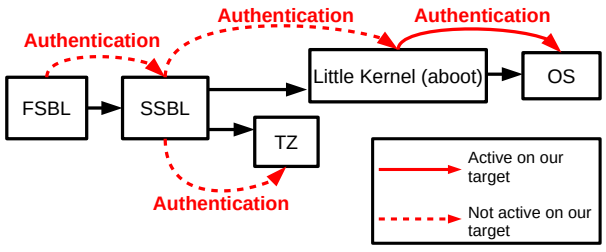

Fig. 5. Secure-Boot Architecture

loader (FSBL) is stored in Read Only Memory (ROM). The FSBL loads the Secondary Stage Bootloader (SSBL) from external memory. The SSBL starts

and loads the Trusted OS executed in secure-mode. Then the SSBL loads and runs Little Kernel, which is the Android Bootloader of the target. Little Kernel loads the Linux Kernel of Android. Since the target is a development board, the Secure-Boot is partially enabled. The authentication of the SSBL by the FSBL is not active, there is also no authentication of Little Kernel by the SSBL. To the best of our knowledge, there is no publicly procedure for activating these authentications. However, the Linux Kernel authentication by Little Kernel can be easily activated by recompiling the Little Kernel code with the right compilation settings. This paper only focuses on the Linux Kernel authentication. During the Linux Kernel compilation, the SHA256 digest of the image is computed and signed with a private key using the RSA algorithm. The signed hash value is stored in the Kernel image. The authentication process is detailed in Figure 6. Little Kernel has the public key which allows to decrypt the signature. During the authentication, Little Kernel computes the SHA256 digest of the current image (ie. HASH_1). Little Kernel decrypts the signed digest available in the image (ie. HASH_2) with the public key, then it compares the two hash values. A comparison result not equal to 0 means that the image is corrupted or the signature is invalid.

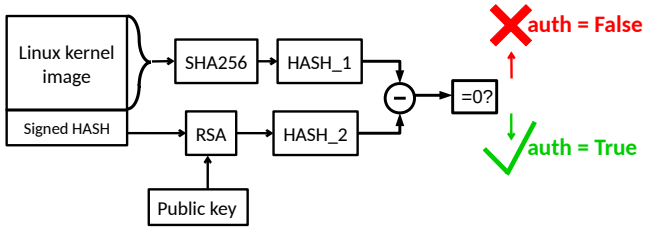

Fig. 6. Authentication process.

The comparison result is used to set the value of the *auth* variable. To load an image, the *auth* variable needs to be set to 1, it is set to 0 by default. By exploring the Little Kernel code, it appears that the comparison of the two hash values is performed by the function *memcmp(HASH_1,HASH_2)*. This function computes the difference between each byte of *HASH_1* and the corresponding bytes of *HASH_2*. The return value of this function is used by a conditional *if* to determine the image validity. This means that faulting the comparison result or the conditional *if* would be interesting to modify the program control flow to avoid setting the *auth* value to 0. We compiled Little Kernel to search a vulnerability in the assembly code. The conditional *if* which verifies the result of *memcmp(HASH_1, HASH_2)* is identified in the ASM code in algorithm 1. The register *r6* is allocated by the compiler to represent the image authenticity (ie. *auth*). The result of memcmp is stored in the register *r0*.

Algorithm 1 ASM and C pseudocodes

C pseudocode	ASM pseudocode
$ret \leftarrow memcmp(HASH\_1, HASH\_2)$	$r0 \leftarrow$ **bl** *memcmp*
if $ret == 0$ **then**	**CLZ** $r6, r0$
$auth = 1$	**LSR** $r6, r6, \#5$
end if	

Figure 7 represents the paths the assembly code can follow after the comparison. The CLZ instruction[6] returns in the output register the number of bits equal to 0 before the first bit equal to 1 in the input value. The LSR instruction[6] translates each bits of the register value on the rigth by a specified number of bits given as input. By analyzing the behavior of this code, it seems that skipping the LSR instruction would keep the value of CLZ *r6,r0* in *r6*. In such case the value in *r6* is in the range $[0, 31]$ if the two digests are different.

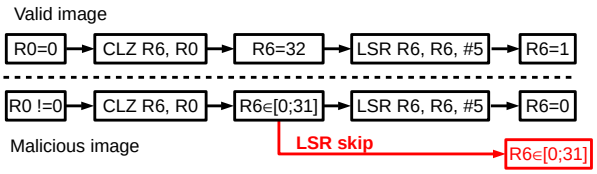

Fig. 7. Algorithm behavior

A modified Linux Kernel with only one different byte from the original was used to validate the potential exploitation of this vulnerability. Since the signed hash did not change, it should be different from the computed hash of the modified image. When the authentication is activated, Little Kernel rejects the corrupted image. However, the image is accepted if the LSR instruction is replaced by a NOP instruction. This confirm that skipping the LSR instruction could be exploited by an attacker to load successfully a corrupted image.

4.4 Characteristic frequency research

Little Kernel is modified to toggle a GPIO state a short time after the vulnerable instruction. An oscilloscope and a passive EM probe are used to measure the target EM emanations. This experience aims at finding a characteristic frequency suitable for triggering purpose. Figure 8 provides the spectrogram generated thanks to the EM measurements from the target. It is possible to identify several characteristic frequencies in the SHA256 computation and in the RSA decryption. The purpose of this methodology is to find a characteristic frequency which happens a short time before the LSR instruction. The frequency at 124.5 MHz was sufficiently detectable by our frequency detector. This frequency appears

[6] See "ARM Architecture Reference Manual ARMV7-A and ARMv7-R edition"

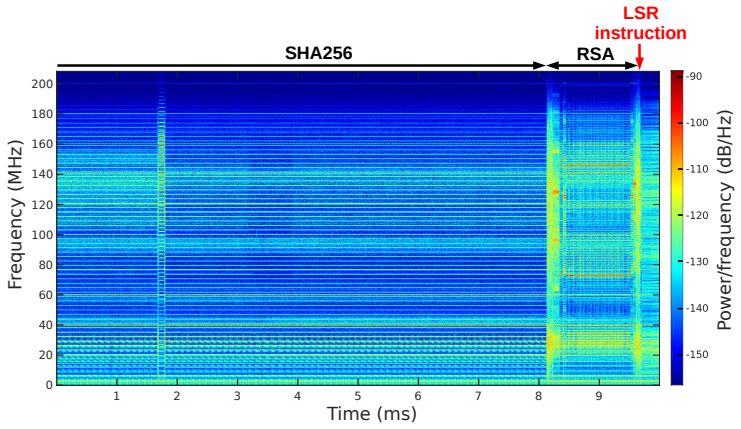

Fig. 8. Target EM emanations spectrogram around the LSR instruction.

during the execution of a loop in the function *BN_from_montgomery_word* from openssl which is used by the RSA decryption function. We set up the frequency detector to generate a trigger signal upon this frequency activation. We measured the delay between the vulnerability and the trigger signal by rising a GPIO a short time after the vulnerable instruction. The mean value of the delay between the rise of the GPIO and the rise of the frequency detector output is 80.57 μs, the standard deviation measured is 476 ns which corresponds to 381 clock cycles at 800 MHz. This result is used to set the delay between the frequency detection and the EM pulse. This value is an approximation since the mean delay of 80.57 μs is measured between the frequency detection and the GPIO, not between the frequency detection and the targeted instruction.

5 Linux Kernel Authentication bypassing on Android Secure-Boot

The previous section shows that it is possible to modify the target control flow by skipping instructions using EMFI. Moreover, a characteristic frequency is identified before a vulnerability in the Android Secure-Boot. Section 5 presents an experiment using all these settings to bypass the Linux Kernel authentication.

5.1 Experimental setup

An oscilloscope generate the trigger signal upon the rise of the frequency detector output. The power supply which reboot the board after each experiment is controled by the PC as described in figure 9. The UART bus allows to monitor the results. The injection probe is placed above the SoC at the best location determined in section 4.1. Note that an EM pulse close enough could be destructive for the frequency detector components. If the two probes are too close then a RF

switch should protect the frequency detector acquisition path during the pulse. An alternative is to place the passive probe below the PCB near the decoupling capacitors. The PCB and the chip act as a shield between the two probes as described in figure 9. Unfortunately, the Linux Kernel authentication is not

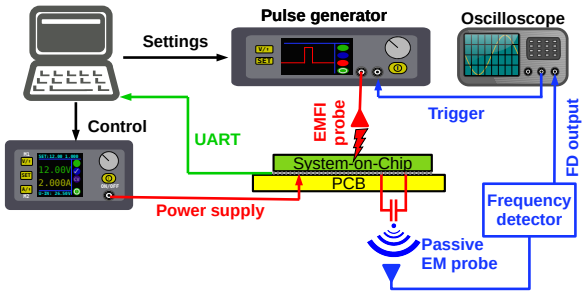

Fig. 9. Experimental setup

activated in the Little Kernel binary of the target. Therefore, an unmodified Little Kernel has been compiled to activate the authentication. Using a modified Linux Kernel image with only one modified byte confirms that the authentication works properly. When loading this image, the authentication fails and the board reboot in recovery mode. The goal of this work is to boot Android with the modified Linux Kernel, which should be impossible. The frequency detector is configured to trigger the injection upon the 124.5 MHz frequency activation. The pulse generator voltage and the delay between the frequency detection and the EMFI are also configured according to the parameters of the section 4. The experiment follows two steps:

- Step 1: The board boots.
- Step 2: The PC gets a message from the UART logs which attests if the authentication succeed or failed.

The authentication happens between these two steps. During the authentication, the 124.5 MHz frequency is activated. It is detected by the frequency detector which triggers the pulse after the 80 μs fixed delay. The step 2 allows discriminating the following scenarios:

1. The "timeout" scenario: the board stops to print log on the UART, the PC never receives the message of step 2. This probably means that the board has been crashed.
2. The "recovery" scenario: the PC gets a message which indicates that the authentication has failed and the board will reboot in recovery mode. This is the expected behavior of the board when no fault has been injected.
3. The "false positive" scenario: the PC gets a message which indicates that the authentication succeeded, but for unknown reasons the board stops printing log just after sending this message.

4. The "success" scenario: The PC gets a first message confirming that the authentication succeeded and the board continues to print logs after this message. It means that the authentication has been successfully bypassed.

5.2 Experimental results

15000 injections has been performed. The total campaign duration is 18 hours. For the "success" case, the boot proceeds during 15 s before rebooting the

Scenario	timeout	recovery	false positive	**success**
Number of attempts	6754 (45.03%)	7912 (52.75%)	251 (1.67%)	**83 (0.55%)**

Table 1. Campaign results

board. It is unlikely that an error would propagate during 15 s and cause the board crash before the end of the boot process. To confirm this hypothesis we performed a new campaign, stopping after the first "success" case and letting the boot proceed. It confirmed that Android has been correctly started with the modified Linux Kernel. 83 "success" over 15000 injections corresponds to a 0.55% success rate. This experiment can succeed in less than 15 minutes if all the settings are properly fixed.

6 Discussion

In this section, we propose a discussion about our results and methodology. Firstly, we want to highlight that we used a smartphone SoC implemented on a development board. Therefore, we have a quite high control over the target, which may not be possible with a true smartphone. For example, we have a physical access to the GPIO through the board connectors, for the setup validation. We also have the board schematics and the target code. Moreover, this attack depends from the target. The vulnerable instruction and the measured delay may change with an other compiler. Also the code leaking at 124.5MHz may change on an other target. Therefore the attack settings need to be adapted for each target.

Secondly, we note that a similar approach based on a commercial solution exists such as the icWaves associated to the Transceiver from Riscure company. The icWaves solution applies a pattern matching method on its input signal to trigger the attack. The Transceiver is used before the icWaves to capture informations in high frequency. This system is based on an SDR coupled with an FPGA as same as our frequency detector but both solutions differ on signal processing. The first difference lays in the signal analyzed by the FPGA: the complex signal $(I + j.Q)$ for the frequency detector versus the module of this signal for the icWave $(|I + j.Q|)$. Thus, the frequency detector has the ability to differentiate frequencies above and below the local oscillator frequency (f_0). In

the signal processing chain of the Riscure solution, a rectifier is used as envelope detection. Furthermore, the envelope detection requires some constraints as the presence of the carrier in the RF signal and a modulation index that is less or equal than 100% to avoid losing information. The signal processing of the SFD method uses a high Q factor filter to increase the selectivity. This method is useful to build a system to only detect activities from a sub part of the SDR output bandwidth. It allows the user to increase the sampling rate of the SDR in order to reduce the temporal uncertainty of the detection by maintaining a narrow sensitive band ensuring the best selectivity. We also note that the SDR used in the frequency detector designed in this work is limited to a 20 MHz sampling rate versus 200 MHz for the Riscure Transceiver. The jitter of our system is correlated with the sampling rate, an higher sampling rate would reduce this jitter. Therefore we could implement our method with better hardware such as the Ettus USRP X310 used in the Riscure Transceiver.

Currently the Riscure solution does not implement our SFD method. The Riscure Transceiver bandwith is selectable between 390kHz and 160MHz. At 2MHz[7] their tool has a 17.3μs delay against a 2.56μs delay for our tool. There is no information about the jitter of the Riscure Transceiver. However it is important to note that the Riscure setup can be optimized and a fair comparison should be based on experiments.

Other triggering methods may exists, such as PCB signals, communication bus or other events in Side-Channel. For comparison purpose, we measured the delay between other basics events and the vulnerability. The mean delay between the rise of the target power supply and the rise of the GPIO is around 1.248 s ($\langle \Delta t \rangle$) and its jitter is approximately 5 ms ($\sigma_{\Delta t}$), which is 10000 greater that the frequency detector temporal uncertainty. This signal is unusable for our attack. We also used an oscilloscope to generate a trigger signal upon the detection of a known message on the UART. We choose the closest UART message to the vulnerability. This message appears in the Little Kernel logs sent over the UART. The mean delay between this trigger signal and the rise of a GPIO after the targeted instruction is around 113 ms ($\langle \Delta t \rangle$) and has a jitter of 2 μs ($\sigma_{\Delta t}$). It is 4 times greater than the temporal uncertainty of our frequency detector with the 124.5 MHz frequency. This trigger signal is usable. The setup is limited to an oscilloscope and it is a simpler setup than our SFD method. The experiment can succeed in a short amount of time with this jitter (estimated to 1 hour for a success). However, the UART logs can be easily disabled during the compilation of Little Kernel. If avoiding printing logs on the UART is quite easy, it is much more difficult to hide completely the EM emanations from the target. The advantages of the SFD method is that it detects internal activity of the SoC and is not limited to I/O.

[7] https://riscureprodstorage.blob.core.windows.net/production/2017/07/transceiver_datasheet.pdf

7 Conclusion

In this paper, we present an hardware attack on a smartphone SoC. This is a combined attack using a real-time analysis of the target EM emanations to synchronize an EMFI. This attack allows to bypass the Linux Kernel authentication step of the Android Secure-Boot, therefore it is possible to load a malicious Linux Kernel despite the Secure-Boot being activated. The mean success rate of this experiment is around one bypass every 15 minutes. To our knowledge, this is the first System-on-Chip Secure-Boot bypass using EMFI. We also present a novel synchronization method for hardware security characterization. Our approach relies on the fact that reducing the delay between the triggering event and the targeted code vulnerability will decrease the jitter associated with this delay. Thus, it will increase the hardware attack success rate. The SFD method uses the activation of a characteristic frequency in the target EM emanations as triggering event. This method is based on an SDR and an FPGA to generate an output signal proportional to the power of the selected frequency. The selected frequency is included in the range between 8 MHz and 6 GHz and is identified thanks to EM emanations analysis. Our system introduces a mean delay of 2.56 μs between the input signal (i.e. the target EM emanations) and the output signal. This mean delay is associated to a jitter of 60.9 ns. Using this synchronization method, we were able to skip a critical instruction by triggering an injection upon the activation of a known frequency. This approach could be used in future work for synchronizing hardware attacks against other targets such as new SoC references or other Secure-Boots. A perspective consist in applying this methodology to bypass the SSBL authentication by the FSBL. Thus it would be possible to get privileges over the TEE.

8 Acknowledgment

The experiments were done on the Micro-PackS^{TM} platform in the context of EXFILES: H2020 project funded by European Commission (No. 88315).

References

[Abi95] Asad A Abidi. Direct-conversion radio transceivers for digital communications. *IEEE Journal of solid-state circuits*, 30(12):1399–1410, 1995.

[BBGV16] Arthur Beckers, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. Design and implementation of a waveform-matching based triggering system. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 184–198. Springer, 2016.

[BKGS21] Otto Bittner, Thilo Krachenfels, Andreas Galauner, and Jean-Pierre Seifert. The forgotten threat of voltage glitching: A case study on nvidia tegra x2 socs. In *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, pages 86–97. IEEE, 2021.

[CH17] Ang Cui and Rick Housley. {BADFET}: Defeating modern secure boot using {Second-Order} pulsed electromagnetic fault injection. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[CPM+18] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes, and Aurélien Francillon. Screaming channels: When electromagnetic side channels meet radio transceivers. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 163–177, 2018.

[DDRT12] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. Electromagnetic transient faults injection on a hardware and a software implementations of aes. In *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 7–15. IEEE, 2012.

[DSN+11] Nicolas Debande, Youssef Souissi, Maxime Nassar, Sylvain Guilley, Thanh-Ha Le, and Jean-Luc Danger. "re-synchronization by moments": An efficient solution to align side-channel traces. In *2011 IEEE International Workshop on Information Forensics and Security*, pages 1–6. IEEE, 2011.

[GAP+20] Clément Gaine, Driss Aboulkassimi, Simon Pontié, Jean-Pierre Nikolovski, and Jean-Max Dutertre. Electromagnetic fault injection as a new forensic approach for socs. In *2020 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6. IEEE, 2020.

[HHM+12] Yu-ichi Hayashi, Naofumi Homma, Takaaki Mizuki, Takafumi Aoki, and Hideaki Sone. Transient iemi threats for cryptographic devices. *IEEE transactions on Electromagnetic Compatibility*, 55(1):140–148, 2012.

[HHM+14] Yu-ichi Hayashi, Naofumi Homma, Takaaki Mizuki, Takafumi Aoki, and Hideaki Sone. Precisely timed iemi fault injection synchronized with em information leakage. In *2014 IEEE International Symposium on Electromagnetic Compatibility (EMC)*, pages 738–742. IEEE, 2014.

[MBTO13] David P Montminy, Rusty O Baldwin, Michael A Temple, and Mark E Oxley. Differential electromagnetic attacks on a 32-bit microprocessor using software defined radios. *IEEE transactions on information forensics and security*, 8(12):2101–2114, 2013.

[MDH+13] Nicolas Moro, Amine Dehbaoui, Karine Heydemann, Bruno Robisson, and Emmanuelle Encrenaz. Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 77–88. Ieee, 2013.

[RNR+15] Lionel Riviere, Zakaria Najm, Pablo Rauzy, Jean-Luc Danger, Julien Bringer, and Laurent Sauvage. High precision fault injections on the instruction cache of armv7-m architectures. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 62–67. IEEE, 2015.

[SMC21] Albert Spruyt, Alyssa Milburn, and Łukasz Chmielewski. Fault injection as an oscilloscope: fault correlation analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 192–216, 2021.

[TSW16] Niek Timmers, Albert Spruyt, and Marc Witteman. Controlling pc on arm using fault injection. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 25–35. IEEE, 2016.

[VTM+18] Aurélien Vasselle, Hugues Thiebeauld, Quentin Maouhoub, Adèle Morisset, and Sébastien Ermeneux. Laser-induced fault injection on smartphone bypassing the secure boot-extended version. *IEEE Transactions on Computers*, 69(10):1449–1459, 2018.

[VWWM11] Jasper GJ Van Woudenberg, Marc F Witteman, and Federico Menarini. Practical optical fault injection on secure microcontrollers. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 91–99. IEEE, 2011.