Shaping the Glitch: Optimizing Voltage Fault Injection Attacks

Claudio Bozzato[1], Riccardo Focardi[2] and Francesco Palmarini[3]

[1] Cisco Talos, cbozzato@cisco.com
[2] Ca' Foscari University of Venice, Cryptosense, focardi@unive.it
[3] Ca' Foscari University of Venice, Yarix, palmarini@unive.it

Abstract. Voltage fault injection is a powerful active side channel attack that modifies the execution-flow of a device by creating disturbances on the power supply line. The attack typically aims at skipping security checks or generating side-channels that gradually leak sensitive data, including the firmware code. In this paper we propose a new voltage fault injection technique that generates fully arbitrary voltage glitch waveforms using off-the-shelf and low cost equipment. To show the effectiveness of our setup, we present new, unpublished firmware extraction attacks on six microcontrollers from three major manufacturers: STMicroelectronics, Texas Instruments and Renesas Electronics that, in 2016 declared a market of \$1.5 billion, \$800 million and \$2.5 billion on units sold, respectively. Among the presented attacks, the most challenging ones exploit multiple vulnerabilities and inject over one million glitches, heavily leveraging on the performance and repeatability of the new proposed technique. We perform a thorough evaluation of arbitrary glitch waveforms by comparing the attack performance against two other major V-FI techniques in the literature. Along a responsible disclosure policy, all the vulnerabilities have been timely reported to the manufacturers.

Keywords: Embedded system security, Fault attacks, Firmware extraction, Microcontrollers.

1 Introduction

Side-channel attacks are considered among the most powerful physical attacks against embedded devices and secure (*e.g.*, smartcards) or specialized hardware (*e.g.*, FPGAs or ASICs). There exist two classes of side-channel attacks: passive and active [SMKM18]. Passive attacks exploit information that is spontaneously leaked by the device such as power consumption [BCO04], timing information [Koc96], electromagnetic emissions [GMO01] or even acoustic emanations [BDG+10]. Active attacks (also known as fault injection attacks), instead, influence the system with internal or external stimuli. For instance, optical fault injection is a powerful technique that exposes the silicon to high intensity light sources, *e.g.*, laser and UV, to induce errors or tamper with the data. Since this technique involves decapsulating [SA02] the chip from its package, technical expertise and specialized equipment are required. Electromagnetic fault attacks (EM-FI) avoid the need of chip decapsulation since faults are injected through the package using an EM injector [SH07]. However, some degree of specialized equipment, *e.g.*, a high precision positioning system [OGM17] or an RF amplifier, can still be necessary to conduct complex attacks.

Given the level of sophistication required by some fault injection attacks, capabilities and performance are not the only relevant factors for classifying and evaluating them: the cost also plays a crucial role. In [BBKN12] Barenghi *et al.* consider as *low cost* the injection methods requiring less than \$3000 of equipment, which are within the means of a single motivated attacker. The authors point out that "these fault injection techniques should be considered as a serious threat to the implementations of secure chips that may be subjected to them". Moreover, the Common Criteria provides a rating for evaluating the attack potential [Com17], that assigns higher severity scores to side-channel and fault injection attacks that require little special equipment and can be mounted under lower cost and expertise. This metric is used in practice by testing laboratories, in order to quantify the resistance of secure devices against these classes of attacks.

In this work, we focus on power supply fault injection, also called voltage fault injection (V-FI), a technique that involves creating disturbances, namely voltage glitches or spikes, on an otherwise stable power supply line. V-FI is a widely used technique because of its effectiveness and, in particular, its low cost. Both in the literature and in industry, efforts have been made to ease mounting complex V-FI attacks [OC14, O'F16]. For instance, commercial tools and open source frameworks such as the ChipWhisperer[1] provide an abstraction layer for controlling the attack parameters at the software level, reducing the electronic skills required. This allows for scientists with different background (*e.g.*, algorithms, statistics, machine learning) to focus on the attack logic and apply their own expertise to the hardware side-channel field. A recent study [ZDCR14] shows that the disturbances induced in the chip via V-FI are effectively caused by the rising and falling edges of the injected glitch. In the literature however, the injected pulses are typically generated as a squared or v-shaped voltage signal, described by a limited set of parameters such as supply voltage, glitch lowest voltage and pulse duration.

In this paper, we move one step forward and propose a new V-FI technique which is based on fully arbitrary voltage glitch waveforms. We analyse the attack performance, repeatability and feasibility, in terms of generating this type of glitches using off-the-shelf and low cost equipment. In order to experimentally assess the effectiveness of the arbitrary glitch waveform approach, we present six unpublished attacks against general purpose microcontrollers from three manufacturers. The injected faults are used to alter the execution-flow of the integrated serial bootloader, skipping security checks or generating side-channels that can, in turn, be exploited to gradually leak sensitive data, including the firmware code. We divide the firmware extraction case studies in two classes based on the design and runtime complexity. In the former, the number of successful faults required is from low to moderate ($\leq 100\,\mathrm{k}$) with a straightforward attack logic. On the contrary, the second class represents particularly challenging attacks that require several days to complete, exploit multiple vulnerabilities and inject over one million glitches. Finally, we perform a thorough evaluation of arbitrary glitch waveforms by comparing the attack performance against two popular V-FI techniques in the literature. All the targets studied in this paper were selected by considering (*i*) general availability and diffusion of the MCU and market share of the manufacturer (*ii*) presence of an embedded serial bootloader for flash programming (*iii*) notable suggested fields and applications such as automotive, healthcare, IoT. Moreover, since our goal is to study the effectiveness of arbitrary glitch waveforms as a general V-FI technique for microcontrollers, we selected target devices having heterogeneous architectures and instruction sets.

We selected firmware extraction for our case studies since firmware protection plays a crucial role in several industrial applications, *e.g.*, for IP and sensitive data protection. Moreover, firmware extraction is a fundamental part of the reverse engineering process performed by researchers to assess the security of embedded systems, ranging from domestic appliances to critical devices such as automotive control units and health devices (*e.g.*, defibrillators, heart rate monitors, implants). In this respect, the six unpublished attacks presented in this work contribute, by themselves, to the state of the art, demonstrating the unsuitability of the attacked devices for security sensitive applications. Along a responsible disclosure policy, all the vulnerabilities that we have found and the firmware

[1] https://newae.com/tools/chipwhisperer

extraction attacks have been timely reported to the manufacturers: STMicroelectronics, Texas Instruments and Renesas Electronics.

Related Work. One of the first attacks exploiting the idea of hardware faults was described (but not tested) by Boneh, DeMillo and Lipton in [BDL97]: it recovered the secret factors p and q of an RSA modulus from a correct signature and an hypothetical faulty one. In the same year, Anderson and Kuhn investigated low cost attacks to tamper resistant devices, focussing on a fault injection attack on pay-TV smartcards [AK97]. Then, in [KK99] Kömmerling and Kuhn described an extensive range of invasive and non-invasive tampering techniques and mitigations. As such, the paper is considered a milestone in the setting of hardware fault attacks, highlighting power supply glitching attacks as the most practical ones. In [ABF+02], authors combined voltage fault injection and power analysis to compromise the confidentiality of cryptographic computations and suggested possible countermeasures. In the following decade, numerous articles (*e.g.*, [BCN+04, HSP09, Goo08, KOP10, KH14]) have further investigated the feasibility of applying voltage glitching to attack both microcontrollers and secure hardware, such as smartcards. In particular, an extensive survey of the current state-of-the-art is provided in [JT12] and [BBKN12]. The power supply fault injection mechanism has been extensively studied and explained as the result of setup time violations in the combinatorial logic [SGD08, SBGD11, ZDC+12, ZDCR14].

Fault attacks against cryptographic implementations is also a very active research topic and, in particular, several papers [SGD08, BBPP09, BBB+10a, BBB+10b] studied the effect of constantly underfeeding a circuit to cause faults. In recent years, fault injection has also been proven effective for achieving privilege escalation on a locked-down ARM processor that was running a Linux-based OS [TM17] and, in the same year, a paper [CPT17] by Cojocar *et al.* proved that two widely used software countermeasures to fault attacks do not provide strong protection in practice.

Firmware extraction from read-protected microcontrollers is a relatively less explored field: in [Goo08] Goodspeed defeated the password protection mechanism found in older Texas Instruments MSP430 microcontrollers. The author used a timing-based side-channel attack to exploit an unbalanced code in the password check routine and, by using voltage glitching, he bypassed the security feature that allows for disabling the serial bootloader (BSL) completely. In [OT17] authors showed that it is possible to downgrade hardware-enforced security restrictions and dump the internal firmware of an STM32F0 MCU, but the attack is invasive as it requires to decapsulate the chip and expose the silicon to UV-C light. In a recent study [Ger17], Gerlinsky has completely bypassed the NXP CRP (Code Read Protection) of the LPC microcontrollers family by injecting a voltage glitch while the bootloader code is checking the CRP status. For what concerns the characterization of V-FI attack parameters, in [CPB+13] and [PBJC14] authors have successfully applied genetic algorithms and other optimization techniques. Finally, the glitch generation using FPGA combined with Digital-to-Analog converters has been studied in the literature (*cf.* [TM17, KOP10]) and adopted by V-FI commercial tools, such as the well-known Riscure VC Glitcher [Risb] and Spider [Risa].

Our paper continues this line of research by focusing on the voltage glitch generation step, in order to optimize the glitching effects that can be exploited by the adversary. In particular we investigate, for the first time, the impact of arbitrary glitch waveforms on the success, performance and repeatability of V-FI attacks. We show that our approach improves on the state of the art by evaluating the attack performance with specific glitch waveforms against the two most popular V-FI generation techniques [HSP09, KOP10]

Contributions. Our contributions can be summarized as follows:

(i) We investigate the effect of different glitch waveforms in the setting of voltage fault injection attacks and, in particular, we propose a new method for the generation of arbitrary glitch waveforms using a low-cost and software-managed setup;

(*ii*) we report on unpublished vulnerabilities and weaknesses in six microcontrollers from three major manufacturers: STMicroelectronics, Texas Instruments and Renesas Electronics. We combine these vulnerabilities and describe the attacks for extracting the firmware from the internal read-protected flash memory. All the attacks are non-destructive and can be performed with a black-box approach, *i.e.*, without any knowledge of the firmware code;

(*iii*) we evaluate the attack performance of our method by comparing the speed, efficiency and reliability of our solution against two popular V-FI techniques.

Paper Organization. In Section 2 we briefly overview the voltage fault injection topic and we describe our experimental setup; in Section 3 we introduce our arbitrary glitch waveform technique and we show how to automatically identify and optimize the glitch shape; in Section 4 and 5 we report on unpublished vulnerabilities of six microcontrollers and describe the attacks for extracting the firmware; in Section 6 we empirically evaluate our technique by comparing it with two popular V-FI techniques, we discuss limitations and propose possible improvements; finally, in Section 7 we draw some concluding remarks.

2 Preliminaries

Voltage fault injection. Voltage fault injection is a non-invasive[2] class of attacks [ABF+02, Sko10] that focuses on creating disturbances on an otherwise stable power supply line in order to cause a misbehaviour in the target. This is the result of setup time violations[3] that can cause incorrect data to be captured [ZDCR14, SBGD11, ZDC+12], allowing an attacker to tamper with the regular control flow, *e.g.*, by skipping instructions, influencing a branch decision, corrupting memory locations, or altering the result of an instruction or its side effects. The disturbances that are induced in the power supply line are called *voltage glitches* or simply *glitches*. A glitch is a transient voltage drop with a duration typically in the ns to µs range, that occurs at a specific instant of time. Glitch timing (also glitch trigger or trigger) is usually calculated as a delay from a specific triggering event such as I/O activity or power-up.

There exist multiple techniques for generating and injecting a voltage glitch into the power supply line of the target device. One of the most commonly used V-FI setup [O'F16, YSW18], also supported by commercial tools such as the well-known *ChipWhisperer* [OC14], is represented in Figure 1a. A transistor, typically MOS-FET, is placed in parallel to the power supply line and it is used to briefly short-circuit `Vcc` to ground.[4] Then, the glitch is triggered by a microcontroller (MCU) or a Field Programmable Gate Array (FPGA) managing the attack timing. The main limitations of this technique are the reduced control over the attack parameters: for instance, additional equipment is required for controlling the voltage levels, and the generated glitch can be unpredictable, (*cf.* Figure 1b) due to variations in both MOS-FET and target electronic properties.

MCUs integrate processor, flash memory and other peripherals in a single package. However, some microcontrollers also integrate a voltage regulator for providing a fixed and stable power supply to the internal processor and memory, independently from the actual input voltage. Depending on the regulator technology, an external filtering capacitor can be required: in this setting, the voltage glitch source can be connected directly to the capacitor pin in order to bypass the internal regulator and avoid any interference of this component during the attack.

[2]Voltage fault injection requires no physical opening and no chemical preparation of the package.

[3]In digital designs the setup time indicates the minimum time required for an input data to be stable before the active edge clock.

[4]Voltage glitches below 0 V are common for particular targets, *e.g.*, smartcards [PBJC14, CPB+13].

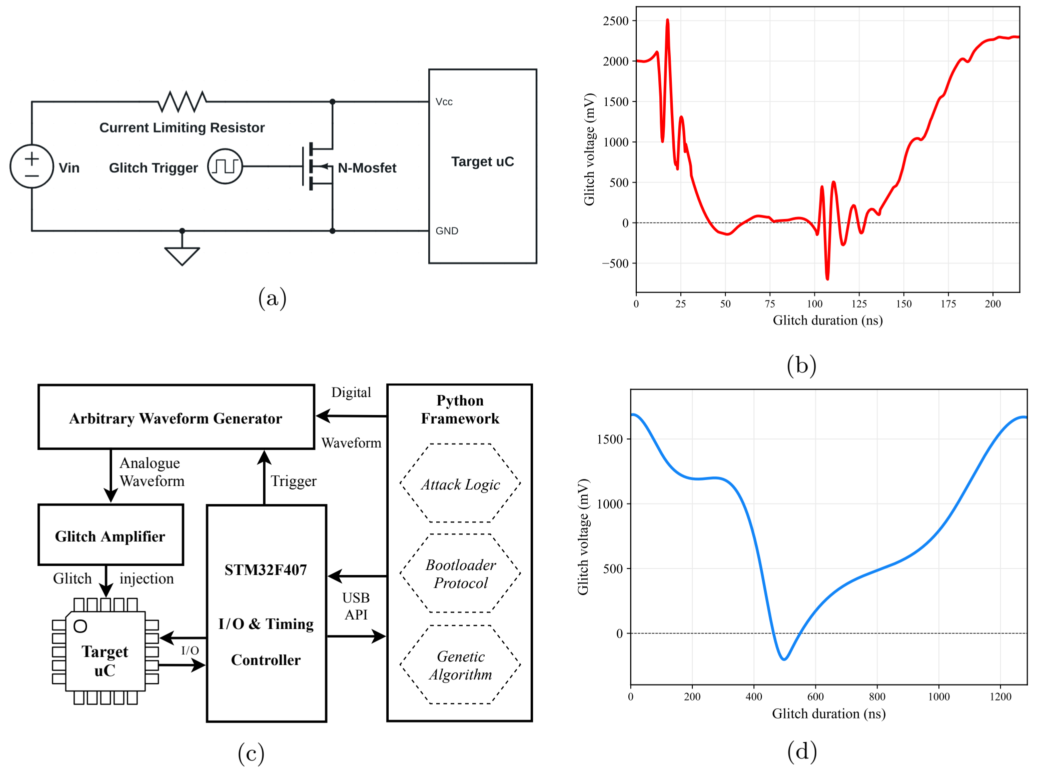


Figure 1: A typical transistor-based V-FI setup (1a) and a generated glitch (1b). The oscillations depend on the target, components in use, and electronic properties of the power supply line. Our V-FI setup for generating arbitrary glitch waveforms (1c) and a generated glitch (1d).

Programming interfaces. The software running on the MCU, namely the firmware, can typically be loaded (also *programmed*) to the internal flash memory using a debug interface exposed by the MCU: the most common is the standard JTAG / SWD interface as it can also be used for debugging the code, inspecting RAM content and accessing the integrated peripherals. Often a serial bootloader, pre-programmed by the manufacturer, exposes a set of API that can be used for, *e.g.*, write, erase or verify the firmware from a computer.

Setup. Based on the work in [KOP10], we developed a low-cost programmable V-FI setup that enables us to overcome the limitations of the transistor-based glitch generation circuit (*cf.* Figure 1a). Our setup is designed around the Digital Direct Synthesis (DDS, *cf.* [Cor04]) technology: a DDS signal generator outputs an arbitrary waveform from a software-defined set of parameters. Similarly to the generation of analogue audio from a digital source, the digital waveform is fed to a Digital-to-Analog Converter (DAC) for producing the equivalent analogue signal. Since in this paper we target general purpose microcontrollers that work at sub-GHz speed, we chose an off-the-shelf DDS device with a reasonable trade-off between performance and price: the FeelTech FY3200S, a very low cost (about 50 $) Arbitrary Waveform Generator with 6 MHz bandwidth and ±10 V output range. This model has an internal waveform memory of 2048 points and allows for controlling the output waveform with 12-bit vertical resolution, using a publicly available protocol over USB.[5] We use this device both as a source of the glitch signal and as an

[5]Notice that, since the waveform upload speed is low, we modified the generator to bypass the built-in upload mechanism, improving the upload speed from about 30 s to 200 ms.

adjustable power supply for regular MCU operation: a logic level change in the external input trigger causes a seamless transition from the constant supply voltage to the glitch waveform loaded in the generator.

As depicted in Figure 1c, we wired the generator to a custom designed board, namely *glitch board*, which has three major functionalities: (*i*) provide the glitch amplification stage and signal path from the generator to the target MCU; (*ii*) interface with the target MCU, handling the low-level communication and time-critical operations; (*iii*) expose a convenient API to control any aspect of the attack from a computer.

The amplification stage is designed after the arbitrary waveform generator bandwidth and the power requirements of a general purpose, low-speed MCU, which is typically well below 100 mA. We used a THS3062[6] current-feedback, high slew rate operational amplifier with 145 mA output current capability working in a 2-stage, non-inverting configuration. This component allows for the target MCU to be constantly powered by the generator, without requiring an adjustable power supply. A reed relay is placed in-between the amplifier output and the target, allowing to fully cut off the device power supply when needed, *e.g.*, for powering off and hard-resetting the MCU, and for preventing harmful voltage fluctuations while loading a waveform in the generator. The output of the relay is exposed via a pin strip header, along with a set of digital I/O lines for communication and control, in order to ease the wiring of new targets. To reduce external interferences and minimize the power trace length, we directly soldered each MCU on a dedicated breakout board without decoupling capacitors; this PCB was then attached to the main glitch board through the pin header connector. We performed all the voltage measurements (*e.g.*, Figure 2 and 4) with a probe placed within 10 mm from the MCU power supply pin.

An ARM STM32F407 microcontroller operating at 168 MHz is responsible for running the firmware that controls the board. We designed the firmware using the minimum code required for handling the low-level communication with the target MCU (*e.g.*, UART, I^{2}C, SPI) and glitch triggering. Upon detection of an external event, the built-in hardware timer guarantees a 10 ns resolution for signalling the generator to inject the glitch after a specific time delay. An API allows for controlling the board and interacting with the target MCU from a computer via a USB link.

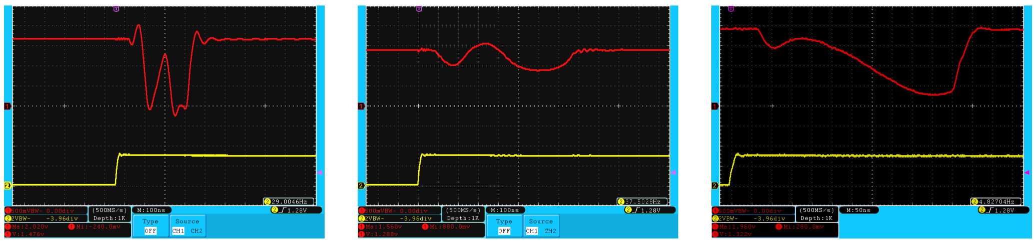
All the complex tasks or algorithms, the attack logic and the specific communication protocol used by the target are implemented in a custom Python framework that assists the design and execution of an attack. As a result, the task of mounting an attack and switching to a different target is substantially simplified. The framework is also responsible for commanding the waveform generation and for controlling the attack parameters, including the search and optimization phase (*cf.* Section 3.1).

3 Arbitrary Waveform Voltage Glitch

The DAC-based voltage glitch generator described in Section 2 enables high flexibility by allowing the attacker to control all typical V-FI parameters (*i.e.*, power supply voltage, glitch voltage, timing and duration) in software and to produce both negative and positive voltage spikes. Up to minor variations due to trace capacitance and impedance, the generated waveform is also repeatable and predictable and it is not influenced by the characteristics of the particular MOS-FET transistor in use, *e.g.*, on-state resistance, capacitance, rise and fall times. However, the most important feature of this setup that we are interested in, is the ability to generate glitch waveforms with arbitrary shape.

In the literature it has been shown (see [ZDCR14]) that rising and falling edges of a voltage glitch play a crucial role in producing oscillations of the core voltage of an FPGA. Since these oscillations cause computation errors that amount to setup time violations in the circuit, in the present paper we move a step forward and experiment on the effectiveness of

[6] `http://www.ti.com/lit/ds/symlink/ths3062.pdf`



(a) STM32F373 (b) TI MSP430F5172 (c) Renesas 78K0R

Figure 2: Oscilloscope trace of the voltage glitch for the STM32F373 (2a), the TI MSP430F5172 (2b) and the Renesas 78K0R (2c). The red (upper) trace represents the waveform of the voltage glitch, while the yellow (lower) is the trigger.

using non-standard glitch waveforms for fault injection on general purpose microcontrollers. In the literature (*cf.* [TM17, KOP10]) Digital-to-Analog converters have already been proven effective for V-FI. However, to the best of our knowledge this work is the first that investigates on using a DAC as a source of arbitrary glitch waveforms, which range from sharp pulses to smooth and variegate waveforms, as exemplified in Figure 2. Depending on its characteristic, the generated waveform can induce effects resembling regular voltage glitches or, for instance, a combination of underpowering [BBB$^+$10a, SBGD11, ZDC$^+$12], negative, positive or multiple voltage glitches. Our experiments (*cf.* Section 6) suggest that, when performing a fault injection attack, the attack success is strongly influenced by the particular waveform of the glitch. However, we point out that thorough investigations and possibly specific equipment, *e.g.*, on-chip delay-based voltmeters (*cf.* [ZDCR14]), are still necessary to identify the precise, low-level effects of different waveforms on the setup time of combinational logic. We leave this as a future work.

Parameter space. Typically, the set of attack parameters that need to be adjusted are timing, glitch length, glitch voltage and possibly power supply voltage [CPB$^+$13, PBJC14, O'F16]. Our technique adds extra parameters for defining the glitch waveform, described as a function of time where the result is the instantaneous voltage generated. This function is translated into the parameter space as a finite set (from 4 to 10) of (x, y) coordinates that are interpolated with cubic interpolation on a 2048-by-4096 grid, and fed into the DAC. Then, the glitch length is encoded as frequency or period of the arbitrary waveform generated.

3.1 Parameter Search and Optimization

We define parameter search as the task of finding one set of parameters that successfully induces one or more faults, implementing a given attack logic. To improve the attack performance, a further optimization phase can be employed for identifying the set of injection parameters that *maximize the probability* of a successful fault. The search phase is mandatory for designing and mounting an attack while the optimization step is subject to the specific requirements and complexity of the attack.

In the following we summarize the the steps performed by the attacker for designing and optimizing a V-FI attack: (i) perform an initial parameter search; (ii) implement the attack scheme and identify $N_{glitches}$ and T_{glitch}, that is the amount of successful faults required and the time spent for injecting one fault, respectively; (iii) define a target time T_{attack} representing the duration under which the attack is considered practical. This can vary from hours to days depending on the attack complexity and attacker's expectations; (iv) define a maximum time $T_{timeout}$ for the optimization step, typically as a fraction of T_{attack}; (v) iterate the optimization step until the success rate $R_{success}$ of the fault attack is such that $R_{success} \geq (T_{glitch} \cdot N_{glitches}) \,/\, T_{attack}$, or the timeout $T_{timeout}$ is reached.

Given the increased number of parameters introduced by our technique, it is important to devise efficient techniques that make optimization feasible and practical. In the following, we describe a semi-automated supervised search (*cf.* Section 3.1.1) and a fully automated unsupervised search based on genetic algorithms (*cf.*, Section 3.1.2).

3.1.1 Supervised Search

Since finding the correct parameter setting is a highly nondeterministic process [PBJC14], during our early experiments we used a human-supervised random search approach inspired by the *Adaptive zoom & bound* method proposed by Carpi *et al.* in [CPB+13]. First, we randomly generate and interpolate the set of (x, y) points describing the candidate arbitrary glitch waveform. Then, we iteratively select a random sample from each parameter interval and test the obtained combination. This process is repeated and the results are manually evaluated, reducing the parameter space accordingly until one solution is found. Clearly this approach is slow to converge and requires expertise for evaluating the results. Additionally, the parameters are not independent: for instance, altering the glitch waveform or duration can affect the glitch trigger position.

3.1.2 Unsupervised Genetic Algorithm

Based on the work of Picek *et al.* [PBJC14], we developed a framework that enables for identifying and optimizing the attack parameters in an unsupervised way. It is designed over a classic genetic algorithm (GA) structure, where an initial population of candidate solutions (the attack parameters) is randomly sampled and an iterative process is responsible for finding a solution that maximizes a *fitness value* F. The fitness value is typically represented by the number of successful glitches produced by a specific set of parameters, but it can be further improved by accounting for additional factors, *e.g.*, the success / failure ratio and the amount of target hangs or reset, as a penalty factor. As an example, in Section 5.1.1 we assign a negative score to the fitness value in the case of a false positive, *i.e.*, an incorrect byte extracted. A solution is composed of one combination of all the members in the parameter space and, at each *generation*, the solutions *evolve* and the attack results are used to evaluate the new fitness value. Since our goal is to find a working solution which is also optimal, *i.e.*, the parameters providing the best attack performance, we repeatedly test one candidate solution and calculate $F(solution) = \frac{S}{T}$, where S is the number of successful attacks and T the total number of tests. We start with 50 tests per candidate and increment this value at each generation: as a result, the first generations enable to test more solutions, while the last are more accurate in evaluating the candidates performance.

At each generation, the population of solutions is improved through repetitive application of the selection, mutation, crossover, and replacement operators. We tuned these operators to the specific characteristics of our V-FI technique:

Selection we tested both the *fitness proportionate selection* and the *tournament selection* standard GA methods of selecting an individual from the population of individuals, and found that both produce acceptable results;

Crossover we use a *uniform crossover* so that, in particular, every single (x, y) point in the glitch waveform can be mixed between two parents with a 0.5 probability;

Mutation every parameter has a different mutation probability. The glitch duration parameter has the highest probability; the glitch waveform has a greater probability in the first generations, together with a higher likelihood of mutating by a small extent;

Replacement a *replace-worst* strategy is adopted, which replaces the worst individual of the current population.

Results. With respect to the parameters optimized by an expert using the supervised approach of Section 3.1.1, our experimental results show that the solutions identified by this algorithm produce the same, or higher attack performance. For each case study presented in this paper, in fact, the attack parameters (*cf.* Section 4.3 and Section 5.2) have been automatically optimized using GA. As described above in Section 3.1, item (v), the optimization converges when a solution that delivers an acceptable performance level is found, *i.e.*, when the success rate $R_{success}$ is good enough to make the attack complete within time T_{attack}; otherwise it is stopped when the timeout $T_{timeout}$ is reached. During our experiments the average time to converge was in the range of 30 minutes to 10 hours, depending on the target device, the vulnerability and the size of the parameter search space. Finally, we point out that the attack and the parameter optimization can be interleaved so to achieve a continuous performance improvement, avoiding unnecessary voltage glitches dedicated exclusively to the optimization phase.

4 Scattered-glitch Attacks

In this section, we assess the effectiveness of arbitrary glitch waveforms described in Section 3 against two case studies of low/moderate complexity (Section 4.1 and Section 4.2). Specifically, the presented attacks exploit a single vulnerability, require a limited amount of glitches (≤ 100 k) and can be completed in a short time frame: from minutes to a few hours. All the presented attacks are novel and extract the firmware from the internal flash memory of the target microcontroller, by exploiting vulnerabilities either in the bootloader or in the debug interface.

In Section 5 we will consider a third case study of increased complexity.

Attacker model. We assume the attacker knows the MCU model under attack and has physical access to the target device. As such, the attacker can directly connect to the exposed pins of the chip, desolder it from the PCB or tamper with the PCB in order to isolate the chip from the other electronic components, minimizing interference. The attacker has no information about the running firmware and the flash memory content but, although not mandatory, she has the ability to inspect the bootloader code in order to identify a suitable instruction to fault.

4.1 Case Study 1: STMicroelectronics

We consider two STM32 ARM MCUs belonging to the F1 and F3 series and manufactured by STMicroelectronics. We select this microcontroller family since it is one of the most widespread in consumer electronics, with over 1 billion units sold between 2007 and 2015 [STM16]. Most STM32 can be programmed either via JTAG / SWD or via the integrated serial bootloader. On USB-enabled MCUs the standard Device Firmware Upgrade (DFU) protocol [USB] is often available.

4.1.1 STM32 F1

We select the STM32F103 as a representative of the F1 series. This model is equipped with a 32-bit ARM Cortex-M3 core operating at 72 MHz.

Security mechanisms. The bootloader offers a security mechanism to lock the device and prevent any read or write operations on the flash memory. In particular, we are interested in the *Readout Protect* command that enables the read protection (RDP) feature. If enabled, the bootloader returns a negative response (NACK) when a *Read Memory* command is issued. The *Readout Unprotect* command disables the read protection at the cost of a complete flash memory erasure.

Attack. We easily bypass this protection mechanism by attacking the *Read Memory* command. After the user requests a read operation, the CPU checks the RDP value and returns the positive (ACK) or negative (NACK) response. By injecting a fault during the RDP checking phase, the bootloader can be deceived into returning an ACK despite the active read protection mechanism. Thus, it is enough to issue a *Read Memory* command over a memory block followed by a voltage glitch, and repeat this until an ACK is received and the content of the selected memory block is returned. The attack is then iterated over the subsequent memory blocks.

4.1.2 STM32 F3

We select the STM32F373 as a representative of the F3 series, equipped with an ARM Cortex-M4 core.

Security mechanisms. A hardware memory protection unit (MPU) implements runtime access control to memory and the SRAM parity errors are checked in hardware. The CPU power supply is provided by an internal voltage regulator and a power supply supervision (PVD) circuit is responsible for holding the device in reset state while the input voltage is outside the working range. Compared to the F1 series (*cf.* Section 4.1.1), the flash memory read protection mechanism is enhanced by using a configurable RDP with three levels of protection. At Level-0 the MCU is unprotected. Level-1 grants access to main memory only when in user mode. *i.e.*, when executing regular firmware code. If, instead, the CPU is running the bootloader or is in debug mode (*e.g.*, via JTAG / SWD), then the flash memory is inaccessible. Finally, the Level-2 protection disables the bootloader and any CPU debugging capability. Moreover, programming the RDP to Level-2 is an irreversible operation both for the user and for STMicroelectronics. Interestingly, the reference manual [STM] points out that the RDP is not a software protection mechanism but it is rather implemented at the hardware level, possibly in the MPU, since any access to protected memory generates a bus error and a hard fault.

Attack. In a recent paper [OT17], the Level-2 protection of a STM32 F0 microcontroller has been bypassed by decapsulating the chip and using UV-C light to alter the value of RDP byte stored in flash memory.

We bypass the Level-2 protection by glitching the MCU during the power-up phase, in order to interfere with the RDP security mechanism. The first step of the attack is to identify the correct timing. Since the bootloader is disabled, the glitch trigger cannot be synchronized to a bootloader command as in the case of the STM32F1. The reference manual [STM] suggests that the RDP loading takes place at the beginning of the boot process thus, ideally, the glitch should be triggered right after the start of the boot process. In fact, the MCU can be successfully downgraded to Level-1 by injecting a glitch at just 11 µs after the boot starts. The attack is repeatable and makes both the bootloader and the JTAG / SWD accessible. Notice that detecting the start of the boot process in not immediate: the presence of a *Power-On Reset* circuit[7] makes it necessary to observe the reset pin (NRST) in order to recognize when the CPU effectively starts booting.

Notice that the high value of the internal pull-up resistor ($\sim 40\,\mathrm{k}\Omega$) increases the rise time of the reset pin and, in turn, this could affect the trigger precision. However, in this case study we found that the rise time is stable and consistent across repeated measurements, indicating that an external pull-up resistor of lower value is not required.

Security implications. The attack can effectively downgrade the RDP from Level-2 to Level-1 but not to Level-0. This can be explained by observing the RDP values for the various levels, reported in Table 1. Since Level-1 is enabled by any value different from `0xCC33` (Level-2) and `0xAA55` (Level-0), it is enough to corrupt a single bit to switch to

[7]This circuit holds the microcontroller in reset state for 1.5 ms to 4.5 ms after power on, allowing the power supply to stabilize.

Table 1: Values of RDP and complement ($\overline{\text{RDP}}$) bytes with respect to the RDP protection levels in the STM32 F3. Notice how a single bit flip can downgrade the protection mechanism to Level-1.

RDP Level	RDP	$\overline{\text{RDP}}$	Security Features
Level-0	`0xAA`	`0x55`	None (unprotected)
Level-1	*Any other value*		Debug w/o flash memory access
Level-2	`0xCC`	`0x33`	No debug (maximum protection)

Level-1 from the other levels. Instead, downgrading to Level-0 would require to precisely alter the value to `0xAA55` which might not be feasible through voltage glitching. At Level-1, the flash memory is not accessible when in debugging mode. However, the debugger is still allowed to read any RAM address or register value. This feature enables an attacker to dump sensitive data (*e.g.*, encryption keys and passwords) by attaching the debugger when a particular firmware routine is being executed. Additionally, an automatic checksum verification of the firmware is often used by vendors to ensure flash data integrity: for instance, the ARM application note 277 [Mat] suggests to perform a CRC-based ROM self-test as part of the boot process. In such a scenario an attacker could extract the firmware by iteratively attaching the debugger and dumping RAM and registers content while the checksum code is being executed. In [OT17] the authors have successfully mounted this attack against a microcontroller of the STM32 F0 series.

4.2 Case Study 2: Texas Instruments

The MSP430 line from Texas Instruments (TI) integrates a 16-bit CPU and it is optimized for low power applications. These microcontrollers can be found in a high number of consumer and industrial devices [Tex], ranging from utility meters and burglar alarms, to safety-critical applications such as fire detectors, medical equipments and physical access control systems. Similarly to the STM MCUs (see Section 4.1), the MSP430 integrates a software bootloader (BSL) that allows the user to program and verify the firmware.

4.2.1 MSP430 F5xx ultra-low power

The first device under test is the MSP430F5172.

Security mechanisms. The main security mechanism is a user-defined password that guards every data access command. The BSL can also be set to automatically erase the flash memory whenever an incorrect password is provided. The microcontroller has a Supply Voltage Supervisor (SVS) and a Brownout Reset (BOR) circuit that reset the device in the case of low voltage.

Attack. We have found that the user is asked to authenticate only when the first read command is issued. Every subsequent command is executed without asking for the password again. We suppose that an authentication flag is stored in RAM and checked before the execution of every read operation; for this reason, we target the flag check routine of the *TX Data Block* command, which can read up to 250 bytes. However, the attack allows us to only dump a single byte, and a subsequent analysis of the BSL code has confirmed that the authentication flag is checked for *every* byte read.

The attack iterates over the following steps, for all addresses *addr* that need to be dumped: (*i*) request a single byte at address *addr*; (*ii*) the BSL responds with ACK, (byte `0x00`) indicating that the command is well formatted; (*iii*) apply a delay T_{trig} starting from the ACK reception, to align with the instruction that checks the authentication flag; (*iv*) inject a voltage glitch in the power supply line; (*v*) if the BSL responds positively, it also returns the value of requested address from flash memory.

Table 2: Results of the attacks on STMicroelectronics and Texax Instruments microcontrollers described in Section 4.1 and Section 4.2.

	Extraction time	**Total glitches**	**Successes**	**Parameter search**	**Repeatability**
STM32F103	1 m (128 kB)	9 k	5 %	20 m	High
STM32F373[*]	N/A	~25	~4 %	2 h	Moderate
MSP430F5172	16 m (32 kB)	34 k	98 %	1 h	High
MSP430FR5725	50 m (8 kB)	100 k	8 %	3 h	Moderate

[*]STM32F373: results for one Level-2 to Level-1 downgrade; complete firmware dump not feasible using fault injection only.

4.2.2 MSP430 FRxx FRAM nonvolatile memory

We consider a second target manufactured by Texas Instruments, the MSP430FR5725.

Security mechanisms. This microcontroller adopts a Ferromagnetic RAM (FRAM) non-volatile memory technology, instead of the regular flash memory. In particular, the presence of an integrated FRAM error correction coding (ECC) circuit makes this MCU family an interesting case study to assess the effectiveness of our voltage glitching technique. Similarly to MSP430F5172 (*cf.* Section 4.2.1), the BSL of this microcontroller is password-protected.

Attack. We successfully applied the same attack logic used for the MSP430F5172, described in Section 4.2.1.

4.3 Experimental Results and Considerations

The attack performance results for the two case studies are highlighted in Table 2. In the table we indicate, for each microcontroller model: the extraction time, which is the total time required to dump the firmware of the target MCU (the flash memory size is reported in parenthesis); the total number of injected glitches during the attack; the percentage of successful faults over the total injected glitches; the time required for the genetic algorithm to search for optimal parameters (including the glitch waveform) used during the attack (see Section 3.1); the repeatability[8], *i.e.*, the effort for reproducing the attack against a different microcontroller of the same model, loosely indicated as High or Moderate. Higher repeatability scores indicate, in particular, that switching MCU do not require a full parameter search and optimization, since the attack parameters can be largely reused for attacking the new target.

Since the STM32F103 is quite sensitive to voltage glitches, and the maximum length for a read operation is 256 bytes, we managed to dump a 128 kB firmware in under 60 seconds. On average, the attack requires a total of just 9000 glitches, which corresponds to a success ratio of about 5 %. We did not manage to perform a flash dump of the STM32F373 using fault injection only, thus the result represents a single triggering of the Level-2 to Level-1 downgrade vulnerability. As an example, in the case of the CRC32 attack (*cf.* Section 4.1.2) the downgrade must be successfully performed once for every extracted byte.

The performance of the attack against the MSP430F5172 microcontroller is excellent and we managed to dump over 2 kB per minute. Since the ratio of successful glitches over the total is above 98 %, the attack speed is limited only by the low data rate (9600 bps) of the BSL serial interface. On the contrary, the MSP430FR5725 attack success rate is noticeably lower than the previous target, despite the prolonged parameter search phase. As a result, 1 kB of FRAM memory is dumped every 6 minutes, thus one order of magnitude slower.

[8]We point out that this term is also used in the literature (*cf.* [SBK10]) as metric for the ability to inject a specific fault and obtain the same result.

The MSP430 microcontrollers have a Supply Voltage Supervisor (SVS) and a Brownout Reset (BOR) circuit that resets the device in the case of low voltage. Interestingly, the genetic algorithm (*cf.* Section 3.1) managed to identify the correct set of parameters that are sufficient to induce a fault in the computation, without triggering any of the two monitoring circuits. Note that the resulting waveform (see Figure 2b), does not resemble the typical squared glitch shape and cannot be generated using the MOS-FET V-FI setup described in Section 2. The power supply voltage identified by the algorithm is close to the minimum working value, which makes this MCU exceedingly sensitive to minimal power disturbances. As a result, the waveform voltage range is extremely compressed and, moreover, the smooth transitions from the power supply voltage (1400 mV) to the lower glitch voltage (880 mV) allow for injecting the glitch undetected.

5 Complex Attacks

The result of the attacks discussed in Section 4 indicates that arbitrary glitch waveforms can help to automatically bypass on-chip protection mechanisms such as brownout detectors or voltage supervisors. In this section we investigate the usage of voltage glitching against particularly challenging attacks, where the total time required is in the range of several days and the number of *successful glitches* is measured in the order of 100 k to over 1 M.

In Section 5.1 we present a third case study that we conducted on two MCU families manufactured by Renesas Electronics. Similarly to those presented in Section 4, these attacks are novel and unpublished and enable for extracting the firmware from the read-protected internal flash memory. We point out that the firmware can only be dumped if multiple vulnerabilities of the on-chip serial bootloader are combined and exploited; the data leaked by each vulnerability alone is indeed insufficient. The attacker model is similar to the one described in Section 4, although with an interesting distinction: the attack is conducted following a full *black-box* approach, *i.e.*, with no information about the running firmware or the flash memory content and, in particular, without reverse engineering the bootloader code.[9]

In Section 5.2 we show the attack results and discuss the issues and the challenges that have emerged and that are specifically related to this class of complex attacks.

5.1 Case Study 3: Renesas Electronics

We tested two microcontrollers from the 78K family manufactured by Renesas Electronics, specifically series 78K0/Kx2 (8-bit core) and 78K0R/Kx3-L (16-bit core). The manufacturer suggests that these MCUs are suitable for a wide range of applications, from home appliances to more critical ones such as healthcare and automotive. A 2016 document from Renesas [Renc] reports that 920 millions MCUs / SoCs have been sold in 2015 and, on average, every new vehicle contains 11 Renesas MCUs installed in the onboard Electronic Control Unit.

The attacks described in this section target the 78K Flash Memory Programming Interface [Rena, Renb] (FMPI), *i.e.*, the bootloader used to load a firmware into the internal flash memory. The microcontroller can be set to boot from the FMPI, exposing the common programming functionalities, *e.g.*, write, erase, verify, to the user.

Security mechanisms. As opposed to what found in Section 4.1 and Section 4.2, this interface does not provide a command to directly read a memory address. All the commands that could potentially leak the flash memory content (*e.g.*, checksum, verify) are enforced to operate on 256 bytes aligned memory blocks. This constraint disallows, for instance, to attack the block checksum using one-byte increments, or to perform an

[9]During regular operation the bootloader code is not memory mapped and thus cannot be dumped.

efficient brute-force by verifying a single byte at a time. Additionally, the 78K offers a mechanism to further protect the content of the flash memory in production devices: a security flag field, controlled by the *Security set* command, can be set to disallow *Boot block rewrite*, *Programming*, *Block erase* and *Chip erase* commands. Since the security flag can be reverted only using a full memory erase, disabling the *Chip erase* command is an irreversible operation.

5.1.1 FMPI Vulnerabilities

In this section we provide a description of the vulnerabilities that we found in the FMPI interface. In Section 5.1.2 we combine these vulnerabilities to mount three different attacks for dumping the flash memory content.

FlagBypass. Restrictions on *program*, *erase* and *chip erase* commands can be bypassed by injecting a fault while the *Security flag* value is being evaluated. We have found that in order to attack this command, two separate glitches are required, and thus the rate of success is very low, *i.e.*, about one per minute or lower.

ShortVerify and **ShortChecksum.** By glitching the routine that checks *start* and *end* parameters sent to the *verify* and *checksum* commands, we are able to force these commands to operate on 4 bytes rather than the intended 256 bytes.

ChecksumLeak. The *checksum* command can be exploited as a side-channel by causing an error during its calculation, which amounts to iteratively subtracting each byte from the starting value `0x10000`. An error introduced by the glitch could cause the checksum routine to miss one byte during its calculation, making it possible to compute the value of this byte through a subtraction from the correct checksum. We point out that this vulnerability can produce false positives (*i.e.*, bytes that are not actually in memory) and that it can be difficult to precisely recover the position of the leaked byte in the flash memory.

Bitflip. In flash memories, the write operation changes the state of a bit from 1 to 0 while, on the contrary, the erase operation switches it back to 1 (see [CGL+17]). Since the bootloader does not enforce a flash erase before writing, the *program* command can be used to alter existing flash memory content. As a consequence, by using solely the *program* command we are able to turn `101` into, *e.g.*, `100` or `000` but not into `111`.

5.1.2 Mounting the Attacks

By combining the vulnerabilities of Section 5.1.1 we mounted three different attacks for dumping the read-protected internal flash memory of the 78K 8-bit and 16-bit MCUs.

SequentialDump. By combining the ShortVerify, ShortChecksum and ChecksumLeak vulnerabilities it is possible to discover four bytes from the flash. The process, depicted in Algorithm 1, works as follows: (*i*) use the ShortChecksum vulnerability (line 2) to obtain the checksum value of the 4 target-bytes; (*ii*) use ShortChecksum and ChecksumLeak vulnerabilities to leak 4 byte values (lines 5 and 6) (*iii*) process and combine (line 8) leaked bytes to obtain a new set of 4-bytes candidates, that have not been checked already; (*iv*) perform a first check (line 10) that filters out the candidates whose checksum does not match the expected *bytesChecksum* (this does not require any interaction with the hardware); (*v*) verify each candidate using the ShortVerify vulnerability (line 11).

The attack is feasible thanks to the ShortVerify and ShortChecksum vulnerabilities, that allow to selectively work on 4 bytes. The API would only allow to perform *verify* and *checksum* of blocks of 256 bytes.

Algorithm 1 Attack for extracting 4 bytes.

```
function FourBytesDump($addr$)
    $bytesChecksum \leftarrow$ ShortChecksum($addr$)
    $oldCandidates \leftarrow \varnothing$
    $leak \leftarrow \varnothing$
    while $|leak| < 4$ do
        $leak \leftarrow leak \cup$ ChecksumLeak($addr$)
    while True do
        $newCandidates \leftarrow$ Combine($leak$) $\setminus oldCandidates$
        for all $guess \in newCandidates$ do
            if Checksum($guess$) $= bytesChecksum$ then
                if ShortVerify($addr, guess$) then
                    return $guess$
        $oldCandidates \leftarrow oldCandidates \cup newCandidates$
        $leak \leftarrow leak \cup$ ChecksumLeak($addr$)
```

Algorithm 2 Memory dump using the *SequentialDump* attack.

```
function FlashDump($startAddr$, $endAddr$)
    $data \leftarrow \varnothing$
    while $startAddr \neq endAddr$ do
        $data \leftarrow data \,\|\,$ FourBytesDump($startAddr$)
        $startAddr = startAddr + 4$
    return $data$
```

Erase & Write. We inject a custom software routine in the firmware that directly dumps the firmware through a serial communication channel with a computer. The attack is mounted as follows: (*i*) use Algorithm 2 to dump the first n bytes of the flash;[10] (*ii*) use the FlagBypass vulnerability to erase the first n bytes; (*iii*) use the FlagBypass vulnerability once more to write the custom routine into the erased memory; (*iv*) set the microcontroller to boot from the custom routine and receive the dump from the serial interface.

This translates into a considerable performance improvement: a full flash dump can be performed in about three to five hours, while the number of required glitches is about one order of magnitude lower with respect to the full *SequentialDump* attack. This attack was tested on 78K0R only. Attacking the 78K0 series might also be possible, although our preliminary tests have been unsuccessful.

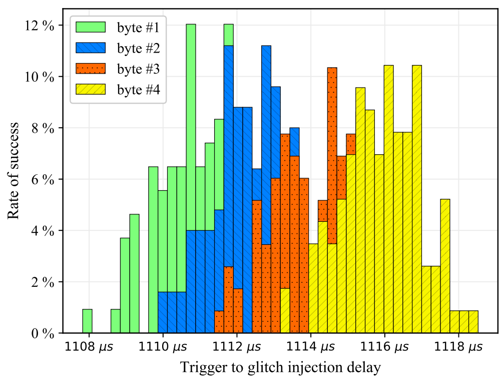
Bitflip & Write. We found that the firmware of several commercial devices does not fill the available flash space completely. For instance, an unused blank[11] memory segment could be left for future firmware updates. A checksum or verify command is sufficient to locate any blank segment in the flash memory; we exploit these empty segments to further optimize the firmware extraction strategy.

The attack is mounted as follows: (*i*) use the `Bitflip` vulnerability to store the firmware-dump routine of the *Erase & Write* attack in an unused memory area; (*ii*) use Algorithm 1 to dump the first 4 bytes to identify the location of the boot section;[12] (*iii*) use Algorithm 2 to dump the first 256 bytes of the boot section; (*iv*) analyse the dumped bytes to identify a suitable candidate for bit-flipping: for instance, a `FF FF` sequence is sufficient for encoding a branch instruction; (*v*) use the `Bitflip` vulnerability to replace all the instructions up to the `FF FF` sequence with NOPs (`0x00`), followed by a branch

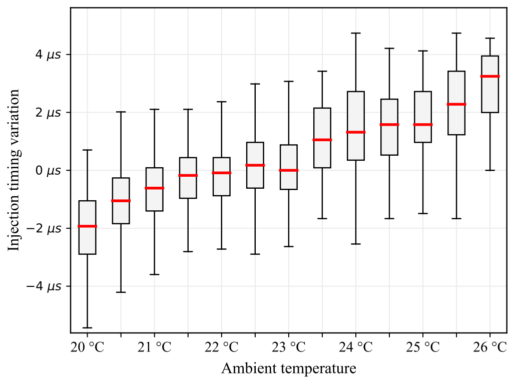
[10] Since the minimum erase size is 1024 bytes, then $n \geq 1024$.

[11] A segment is considered blank if all bytes have value 0xFF, *i.e.*, the segment is erased.

[12] In the 78K architecture the first 4 bytes of the flash memory hold the address of the firmware boot section (*i.e.*, the entry point).



(a)



(b)

Figure 3: Frequency distribution of 4 bytes leaked by the Renesas *ChecksumLeak* vulnerability (3a): on the x-axis the delay between the trigger and glitch injection, on the y-axis the rate of successful faults. The effect of ambient temperature variations on the optimal injection timing (3b). The reference measurement is at 23 °C and 1110 μs.

Table 3: Results of the three attacks on Renesas 78K microcontrollers described in Section 5.1.2. The values are obtained by averaging the results of three complete firmware extractions.

	Extraction time	**Total glitches**	**Successful glitches**	**Parameter search**	**Repeatability**
SequentialDump	2 d 12 h	3.3 M	549 k	5 h	Moderate
Erase & Write	~3 h	513 k	45 k	1 h	High
Bitflip & Write	<1 h	204 k	15 k	30 m	High

instruction to the firmware dump routine; (*vi*) set the microcontroller to boot from the custom routine and receive the dump from the serial interface.

5.2 Experimental Results and Considerations

In Table 3 we summarize the results of the Renesas attacks described in Section 5.1.2. As one would expect, the *SequentialDump* attack is the slowest one as it dumps all the flash memory by using fault injection only, thus requiring a very high glitch count. On the contrary, the software dump routine loaded using the *Erase & Write* and *Bitflip & Write* attacks leads to a major improvement in firmware extraction time. To this end, since these attacks extract few bytes using fault injection, a trade-off can be achieved between the bare extraction speed (*i.e.*, the glitch success ratio) and the time required for the parameter optimization phase. However, we point out that to trigger the FlagBypass vulnerability, required by the *Erase & Write* and *Bitflip & Write* attacks, two repeated glitches are necessary, resulting in an extremely low success ratio: we managed to achieve one success in about 15 to 30 minutes.

The *SequentialDump* attack can run fully automated and unsupervised with a reasonable degree of repeatability. Indeed, we performed several full firmware dumps from different 78K0 and 78K0R microcontrollers. Typically, switching MCU requires a re-optimization phase of the attack parameters, including the glitch waveform, in order to achieve good glitch success ratio. Interestingly, we experienced that each exploited vulnerability best performs with a specific glitch waveform. Although a single glitch waveform can be sufficient to trigger multiple vulnerabilities, the success ratio of such a waveform is low.

Our tests revealed also that, even with the correct parameters, both the attack performance and the repeatability of long-running attacks can be influenced by timing errors and external variables such as the ambient temperature.

5.2.1 Injection Timing

Fault injection aims at causing an error during the computation of a specific task, so the timing is a critical parameter. We refer to *injection timing* as the delay that we introduce between an external trigger event and the injection of the glitch. During our experiments we experienced glitch timing inaccuracy that affected the attack performance. In general, this could be caused by external and physical variables such as temperature, clock stability, trigger precision or interferences. As an example, Figure 3a depicts the effect of timing variations on the output of the Renesas *ChecksumLeak* vulnerability presented in Section 5.1.1: it appears evident the correlation between injection delay and the probability of leaking one of the four bytes. The ± 2 µs error is introduced by the inaccurate trigger event (*i.e.*, the command transmission to the bootloader) combined with the fluctuation in the checksum computation time caused by small variations in the internal oscillator frequency. As a result, this timing error makes it difficult to recover the exact position of the leaked byte because of the overlapping probability distributions. We point out that timing errors could be minimized with the use of synchronization techniques such as frequency locking [Sko11, OC15] or side-channel power analysis [OC14].

5.2.2 Ambient Temperature

Extreme temperatures are known to facilitate fault injection and side channel attacks on several targets [HSH+09, HS13, QS02]. In fact, we have found that even small variations in the ambient temperature can affect the attack performance, requiring multiple adjustments of the injection parameters and thus affecting the attack repeatability. When targeting microcontrollers that are running on the integrated oscillator, attacks using long injection delays (≥ 100 µs) can be particularly sensitive to temperature changes. This is particularly interesting when performing attacks that span over more than one day: for instance, heating or cooling systems can be turned off during the night causing a noticeable temperature variation. To verify our observations we measured the impact of a ± 3 °C ambient temperature excursion on the Renesas *ChecksumLeak* vulnerability. We repeated the attack for one hour by using, at each iteration, a different injection timing that is randomly sampled in the range 1100 ± 5µs. The box plot in Figure 3b collects all the glitch timings that lead to a successful attack: the plot suggests that the value of the glitch delay is proportional to the ambient temperature.

This behaviour is caused by slight variations in the frequency of the internal oscillator (*cf.* [Rob, VF05]). In particular, an increase in ambient temperature causes a decrease of execution speed in the Renesas 78K microcontroller, which misaligns the target instruction with respect to the injected glitch. We managed to reduce the error caused by temperature variations by applying a ~ 0.1 %/°C compensation factor to the injection timing. This factor can be easily calculated from the results of the above test.

6 Evaluation

In order to evaluate the attack performance of our approach, referred to as `AGW` in this section, we conducted a series of tests against the two other main voltage glitching techniques: an ubiquitous transistor-based setup, namely `Mosfet`, and its generalization using a DAC-generated pulse that we will refer to as `Pulse`.

6.1 Performance Analysis and Comparison

In the following we describe the three setups that we used during the tests:

`Mosfet` This is the classic configuration often adopted in the literature [OC14, O'F16], similar to the one described in Section 2 and depicted in Figure 1a. Specifically,

Table 4: Performance comparison of three vulnerabilities described in Section 5.1.1 using different voltage glitching techniques. Results are obtained by averaging 4 independent runs of 10 minutes each. Values inside the parenthesis indicate the percentage relative to the total glitch count.

Vulnerability	Technique	Success	False Positive	Reset	Reset⁄Success	Glitch Count
ShortVerify	`Mosfet`	668 (2.6 %)	1	1780 (6.9 %)	2.66	25701
	`Pulse`	969 (3.7 %)	0	1685 (6.4 %)	1.74	26180
	`AGW`	1291 (6.8 %)	1	2786 (14.6 %)	2.16	19044
ShortChecksum	`Mosfet`	474 (2.1 %)	1	1862 (8.3 %)	3.93	22322
	`Pulse`	689 (2.8 %)	1	1632 (6.6 %)	2.37	24931
	`AGW`	728 (4.4 %)	2	2912 (17.7 %)	4.01	16475
ChecksumLeak	`Mosfet`	412 (4.9 %)	254 (3.0 %)	2481 (29.8 %)	6.02	8329
	`Pulse`	455 (5.6 %)	158 (1.9 %)	2510 (30.9 %)	5.52	8136
	`AGW`	687 (8.6 %)	42 (0.5 %)	2515 (31.5 %)	3.66	7977

we use a VN2222 N-channel MOS-FET paired with an ADP3623 driver to ensure fast and sharp switching times. This setup allows for configuring glitch duration and timing. The MCU power supply voltage can be varied manually and the glitch voltage (*i.e.*, the peak low voltage of the glitch waveform) is fixed at 0 V since the source pin of the MOS-FET is tied to ground.

`Pulse` With respect to the previous setup, this allows for improved configurability and control over the generated glitch. The glitch is, in fact, more predictable and it is not influenced by the characteristics of the specific MOS-FET in use. We implemented this setup using our arbitrary function generator (*cf.* Section 2), enabling us additional control over the power supply voltage and the glitch voltage, duration, timing and, in particular, the rise and fall times of the glitch edges. To the best of our knowledge, this setup resembles the typical usage (*cf.*, [CPT17, Elz18]) of the industry standard Riscure Spider [Risa] and VC Glitcher [Risb] tools.

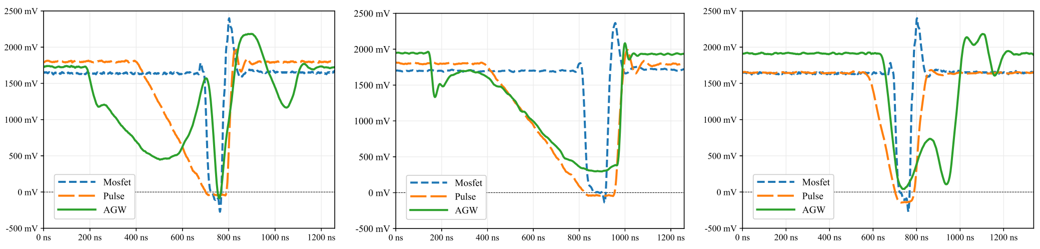
`AGW` Our proposed setup (*cf.* Section 2) enhances the output capabilities of the `Pulse` method, allowing the attacker to produce fully arbitrary glitch waveforms. Moreover, this setup is capable of producing voltage glitches with 20 V peak-to-peak amplitude and ±10 V output range, for supporting a broad variety of targets.

While the hardware of existing commercial tools (*e.g.*, Riscure products) might be capable of producing AGWs we point out that, according to the documentation and literature, they seem to be employed for producing pulsed glitches by adjusting glitch duration and voltage only. Instead of relying on commercial tools, it was more convenient for us (in terms of cost and flexibility) to develop our proof-of-concept setup.

The high runtime complexity of the Renesas attacks (*cf.* Section 5.1) makes the *ShortVerify*, *ShortChecksum* and *ChecksumLeak* vulnerabilities an interesting benchmark for evaluating the performance of these three V-FI techniques. The experiments were conducted by attacking a microcontroller of the 78K0/Kx2 family, pre-programmed with known memory content so to verify the correctness of the extracted data. For each glitching technique, all the attack parameters were computed by running the algorithm described in Section 3.1.2 for 8 hours, in order to guarantee fairness and comparability of the results.[13] In Table 4 we present the performance results obtained by averaging 4 independent runs of 10 minutes each that we conducted for every combination of vulnerability and technique. Between each run, the glitch timing was adjusted to compensate for temperature variations (*cf.* Section 5.2.2).

In the table we indicate: (*i*) the number of successes, *e.g.*, the amount of glitches that

[13]Although in some cases the optimal set of parameters could have been found in a shorter period, we forced the algorithm to run for 8 hours.



(a) ShortVerify (b) ShortChecksum (c) ChecksumLeak

Figure 4: Comparison of the voltage glitch waveforms for the three V-FI techniques and vulnerabilities evaluated in Table 4.

Table 5: Full firmware extraction from a 60 kB flash memory using the Renesas *SequentialDump* attack. Overview of the performance variation among different voltage glitching technologies.

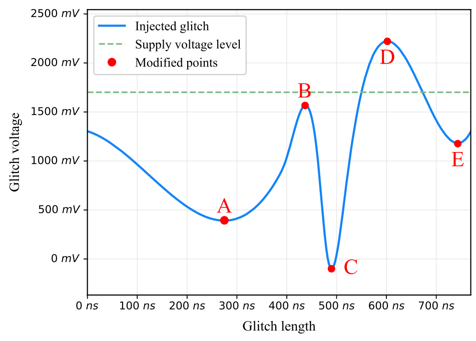
Technique	**Tested combinations**	**# ShortVerify**	**# ChecksumLeak**	**# ShortChecksum**	**Total glitch count**	**Total dump time**
`Mosfet`	351 k	13.9 M	3.1 M	699 k	18.1 M	6 d 19 h
`Pulse`	142 k	3.8 M	2.6 M	582 k	7.1 M	3 d 16 h
`AGW`	105 k	1.5 M	1.5 M	351 k	3.3 M	2 d 12 h

lead to a successful verify operation or to extract the correct byte; (*ii*) the number of false positives, such as an incorrect byte extracted or a bad short-checksum; (*iii*) how often we reset the microcontroller for becoming unresponsive after a glitch; (*iv*) the total glitch count, injected during the 10 minutes run. The results show that `AGW` outperforms the other techniques. In particular we observe that the absolute number of successful glitches is noticeably higher. Similarly, `AGW` presents a higher ratio of success over the total injected glitches, thus our technique is both faster and more efficient. The false positive count for the *ChecksumLeak* vulnerability is 6 times and almost 4 times lower with respect to `Mosfet` and `Pulse` techniques; interestingly, in Section 6.1.1 we show how this enables a major reduction in the firmware extraction time. When a glitch makes the microcontroller non-responsive, a reset operation is performed at the cost of additional overhead, due to the bootloader re-initialization. The results show that, in general, our technique induces more resets in the target, thus limiting the number of glitches injected in the 10 minutes run. This limitation is, however, compensated by a higher fault efficiency, which contributes in rising the overall attack performance.

Finally, in Figure 4, for each vulnerability we plot the glitch waveforms of the three V-FI techniques. The `Mosfet` setup shows the shorter glitch duration and the maximum voltage amplitude as a consequence of both undershooting below 0 V and overshooting above `VCC`. While the edge rise time is sharp for all the three vulnerabilities, the fall time in the `Pulse` setup appears to be much longer, about 400 ns in the case of Figure 4b. Interestingly, the duration of the low-end (close to 0 V) part of the `AGW` glitch is similar to those of the other two techniques.

6.1.1 Firmware Extraction Time

After evaluating the performance of the single vulnerabilities, we tested the *SequentialDump* attack presented in Section 5.1.2. The results, depicted in Table 5, represent a full 60 kB flash memory dump. Notice that, for the sake of brevity, each technique was tested on 5 consecutive 256-bytes memory blocks only; the 60 kB result was obtained after calculations. Our technique managed to dump the firmware 32 % and 63 % faster than `Pulse` and `Mosfet`, respectively. Interestingly, our approach is also very efficient, reducing the total number of glitches required to complete the attack: the `Mosfet` produced about five time the number



(a) Points of perturbation

Point	Variation	Success / minute
A	+200 mV	−22 %
A	+550 mV	−58 %
B	−260 mV	−16 %
C	−170 mV	Not working
C	+140 mV	+7 %
D	−510 mV	−33 %
E	+480 mV	−1 %

(b) Performance variation

Figure 5: Voltage glitch waveform for the Renesas *ShortVerify* vulnerability (5a). Red points are shifted in the y-axis (voltage), reporting the performance effect of each variation in the table (5b).

of glitches, followed by the `Pulse` method which doubled the value of `AGW`. In fact, since these techniques show a higher false positives ratio in the *ChecksumLeak* vulnerability, the number of extracted combinations that require to be verified (*cf.* Algorithm 1) is also higher.

6.1.2 Glitch Waveform Characterization

We conducted a final experiment to characterize how minor perturbations in the *ShortVerify* waveform would impact the success rate. This waveform, used for the `AGW` tests and depicted in Figure 5a, is capable of producing about 130 successful verify operations per minute. The test is performed as follows: a point of local minimum or maximum, labelled with letters from A to E, is selected and moved along the y-axis so to lower or raise its voltage; after interpolating the 5 points, the resulting waveform is tested for 10 minutes and the number of successful verify operations is collected. The results of Figure 5b highlight a correlation between a specific perturbation and the attack success rate. As an example, if point A is raised by 200 mV, then the performance decreases by about 22 %. Interestingly, lowering point C by 170 mV does not produce any success at all, while raising this point by 140 mV increases the number of successful *verify* operations by 7 %. This particular result indicates that refinements in the parameter optimization algorithm (see Section 3.1) could leave room for further performance improvements of the `AGW` technique.

6.2 Limitations and Further Improvements

The experimental campaign conducted proved that the success rate of an attack can be improved by selecting specific glitch waveforms and, as described in Section 4.3, some countermeasures such as integrated voltage supervisors can be automatically bypassed. However, this improvement comes at the cost of increased complexity in the glitch parameter search (*cf.* Section 3.1). In particular, searching and optimizing the glitch waveform might be time consuming, possibly requiring numerous glitches. The choice of the voltage fault injection technique should, thus, account for both the security mechanisms employed by the target (if any) and the overall attack complexity. As a compromise between attack performance and parameter search duration, it might in fact be advantageous to reduce the degrees of freedom of the waveform generation. For instance, starting the glitch waveform optimization from a small, predefined set of shapes, could substantially ease the task of the optimization algorithm. It is thus worth considering `AGW` in complex scenarios, where the total number of glitches can be significantly reduced (and consequently the probability of target device failures) resulting in faster and more reliable attacks.

We plan to conduct more experiments on new microcontrollers and other classes of fault attacks, *e.g.*, against cryptographic implementations, and to target secure microcontrollers or hardware (*e.g.*, smartcards, USB tokens) and high speed Systems-on-a-Chip (SoCs). To this end, our low cost generator (see Section 2), which is limited to produce arbitrary waveforms with a maximum frequency of about 6 MHz, will be upgraded to increase both bandwidth and output sampling rate. Finally, we also leave as a future work the study of an improved version of the genetic algorithm presented in Section 3.1.2, and the investigation of other classes of optimization algorithms.

7 Conclusion

In this paper we have studied, for the first time, how voltage glitches with arbitrary waveforms affect the success and efficiency of an attack. We have also investigated the feasibility of identifying a valid set of attack parameters, including the glitch waveform, in an automated and unsupervised way, and showed the feasibility of generating these type of glitches using low cost equipment. Furthermore, we have presented novel attacks on six widely used microcontrollers from three manufacturers. These attacks target the bootloader interface and allow for extracting the firmware from the internal protected flash memory. Following a responsible disclosure policy, we have timely reported the security flaws to STMicroelectronics, Texas Instruments and Renesas Electronics. Finally, we have evaluated the performance improvement provided by the arbitrary glitch waveforms against two other major voltage glitching techniques. The results showed an increment in the firmware extraction speed and, in particular, a significantly lower amount of injected glitches required to complete the attack.

Regardless of the arbitrary glitch waveform technique, we believe that the presented attacks are valuable. They provide evidence that an attacker, even with limited resources, can use voltage fault injection to bypass the protection mechanisms offered by the microcontrollers under test. Thus, we certainly discourage the adoption of this kind of microcontrollers for security or safety-involved systems. Even when microcontrollers come with some basic protection mechanisms and voltage supervisors, we have shown how these can be easily and systematically bypassed, allowing for efficient firmware extraction. Moreover, firmware extraction is problematic by itself for intellectual property. So, independently of the criticality of the application, companies should be aware that the protections implemented in budget microcontrollers are insufficient to protect the know-how in the firmware and, consequently, devices could be cloned or tampered by criminals with a low effort and investment.

References

[ABF$^+$02] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and Jean-Pierre Seifert. Fault attacks on RSA with CRT: concrete results and practical countermeasures. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, pages 260–275, 2002.

[AK97] Ross J. Anderson and Markus G. Kuhn. Low cost attacks on tamper resistant devices. In *Security Protocols, 5th International Workshop, Paris, France, April 7-9, 1997, Proceedings*, pages 125–136, 1997.

[BBB$^+$10a] Alessandro Barenghi, Guido Bertoni, Luca Breveglieri, Mauro Pellicioli, and Gerardo Pelosi. Low voltage fault attacks to AES. In *HOST 2010, Proceedings of the 2010 IEEE International Symposium on Hardware-Oriented Security* *and Trust (HOST), 13-14 June 2010, Anaheim Convention Center, California, USA*, pages 7–12, 2010.

[BBB^{+}10b] Alessandro Barenghi, Guido Bertoni, Luca Breveglieri, Mauro Pellicioli, and Gerardo Pelosi. Low voltage fault attacks to AES and RSA on general purpose processors. *IACR Cryptology ePrint Archive*, 2010:130, 2010.

[BBKN12] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, 2012.

[BBPP09] Alessandro Barenghi, Guido Bertoni, Emanuele Parrinello, and Gerardo Pelosi. Low voltage fault attacks on the RSA cryptosystem. In *Sixth International Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2009, Lausanne, Switzerland, 6 September 2009*, pages 23–31, 2009.

[BCN^{+}04] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *IACR Cryptology ePrint Archive*, 2004:100, 2004.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, pages 16–29, 2004.

[BDG^{+}10] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic side-channel attacks on printers. In *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, pages 307–322, 2010.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, pages 37–51, 1997.

[CGL^{+}17] Yu Cai, Saugata Ghose, Yixin Luo, Ken Mai, Onur Mutlu, and Erich F. Haratsch. Vulnerabilities in MLC NAND flash memory programming: Experimental analysis, exploits, and mitigation techniques. In *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4-8, 2017*, pages 49–60, 2017.

[Com17] Common Criteria Working Group. CC v3.1. Release 5 — Common Methodology for Information Technology Security Evaluation, April 2017. https://www.commoncriteriaportal.org/files/ccfiles/CEMV3.1R5.pdf.

[Cor04] Lionel Cordesses. Direct digital synthesis: A tool for periodic wave generation (part 1). *IEEE Signal processing magazine*, 21(4):50–54, 2004.

[CPB^{+}13] Rafael Boix Carpi, Stjepan Picek, Lejla Batina, Federico Menarini, Domagoj Jakobovic, and Marin Golub. Glitch it if you can: Parameter search strategies for successful fault injection. In *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, pages 236–252, 2013.

[CPT17] Lucian Cojocar, Kostas Papagiannopoulos, and Niek Timmers. Instruction duplication: Leaky and not too fault-tolerant! In *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*, pages 160–179, 2017.

[Elz18] Ivo Van Der Elzen. Using fault injection to weaken rsa public key verification. Technical report, Riscure, 2018.

[Ger17] Chris Gerlinsky. Breaking Code Read Protection on the NXP LPC-family Microcontrollers, 2017. https://recon.cx/2017/brussels/resources/slides/RECON-BRX-2017-Breaking_CRP_on_NXP_LPC_Microcontrollers_slides.pdf.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, number Generators, pages 251–261, 2001.

[Goo08] Travis Goodspeed. A side-channel timing attack of the msp430 bsl. *Black Hat USA*, 2008.

[HS13] Michael Hutter and Jörn-Marc Schmidt. The temperature side channel and heating fault attacks. In *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, pages 219–235, 2013.

[HSH$^+$09] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009.

[HSP09] Michael Hutter, Jorn-Marc Schmidt, and Thomas Plos. Contact-based fault injections and power analysis on rfid tags. In *Circuit Theory and Design, 2009. ECCTD 2009. European Conference on*, pages 409–412. IEEE, 2009.

[JT12] Marc Joye and Michael Tunstall, editors. *Fault Analysis in Cryptography*. Information Security and Cryptography. Springer, 2012.

[KH14] Thomas Korak and Michael Hoefler. On the effects of clock and power supply tampering on two microcontroller platforms. In *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2014, Busan, South Korea, September 23, 2014*, pages 8–17, 2014.

[KK99] Oliver Kömmerling and Markus G. Kuhn. Design principles for tamper-resistant smartcard processors. In *Proceedings of the 1st Workshop on Smartcard Technology, Smartcard 1999, Chicago, Illinois, USA, May 10-11, 1999*, 1999.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pages 104–113, 1996.

[KOP10] Timo Kasper, David Oswald, and Christof Paar. A versatile framework for implementation attacks on cryptographic rfids and embedded devices. *Trans. Computational Science*, 10:100–130, 2010.

[Mat] Matthias Hertel, ARM. AN277, ROM Self-Test in MDK-ARM. http://www.keil.com/appnotes/files/an277.pdf.

[OC14] Colin O'Flynn and Zhizhang (David) Chen. Chipwhisperer: An open-source platform for hardware embedded security research. In *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, pages 243–260, 2014.

[OC15] Colin O'Flynn and Zhizhang Chen. Synchronous sampling and clock recovery of internal oscillators for side channel analysis and fault injection. *J. Cryptographic Engineering*, 5(1):53–69, 2015.

[O'F16] Colin O'Flynn. Fault injection using crowbars on embedded systems. *IACR Cryptology ePrint Archive*, 2016:810, 2016.

[OGM17] Sébastien Ordas, Ludovic Guillaume-Sage, and Philippe Maurine. Electromagnetic fault injection: the curse of flip-flops. *J. Cryptographic Engineering*, 7(3):183–197, 2017.

[OT17] Johannes Obermaier and Stefan Tatschner. Shedding too much light on a microcontroller's firmware protection. In *11th USENIX Workshop on Offensive Technologies, WOOT 2017, Vancouver, BC, Canada, August 14-15, 2017.*, 2017.

[PBJC14] Stjepan Picek, Lejla Batina, Domagoj Jakobovic, and Rafael Boix Carpi. Evolving genetic algorithms for fault injection attacks. In *37th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2014, Opatija, Croatia, May 26-30, 2014*, pages 1106–1111, 2014.

[QS02] Jean-Jacques Quisquater and David Samyde. Eddy current for magnetic analysis with active sensor. In *Proceedings of eSMART*, volume 2002, 2002.

[Rena] Renesas Electronics. 78K0/Kx2 Flash Memory Programming. https://www.renesas.com/en-eu/doc/DocumentServer/024/U17739EJ3V0AN00.pdf.

[Renb] Renesas Electronics. 78K0R/Kx3-L Flash Memory Programming. https://www.renesas.com/en-eu/doc/DocumentServer/026/U19486EJ1V0AN00.pdf.

[Renc] Renesas Electronics. Renesas Mid-Term Growth Strategy. https://www.renesas.com/en-in/media/about/ir/event/presentation/2016-12-q2-strategy.pdf.

[Risa] Riscure. Spider. https://www.riscure.com/uploads/2017/07/spider_datasheet_1.pdf.

[Risb] Riscure. VC Glitcher. https://www.riscure.com/uploads/2017/07/datasheet_vcglitcher.pdf.

[Rob] Robin Walsh, Atmel. RC Oscillator Frequency Drift Compensation. http://ww1.microchip.com/downloads/en/DeviceDoc/article_ac9_atmegaxx8pa-15-rc-oscillator.pdf.

[SA02] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, pages 2–12, 2002.

[SBGD11] Nidhal Selmane, Shivam Bhasin, Sylvain Guilley, and Jean-Luc Danger. Security evaluation of application-specific integrated circuits and field programmable gate arrays against setup time violation attacks. *IET Information Security*, 5(4):181–190, 2011.

[SBK10] Daniel Skarin, Raul Barbosa, and Johan Karlsson. GOOFI-2: A tool for experimental dependability assessment. In *Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2010, Chicago, IL, USA, June 28 - July 1 2010*, pages 557–562, 2010.

[SGD08] Nidhal Selmane, Sylvain Guilley, and Jean-Luc Danger. Practical setup time violation attacks on AES. In *Seventh European Dependable Computing Conference, EDCC-7 2008, Kaunas, Lithuania, 7-9 May 2008*, pages 91–96, 2008.

[SH07] Jörn-Marc Schmidt and Michael Hutter. Optical and em fault-attacks on crt-based rsa: Concrete results. In *Proceedings of 15th Austrian Workshop on Microelectronics*, pages 61–67, 2007.

[Sko10] Sergei Skorobogatov. Flash memory 'bumping' attacks. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, pages 158–172, 2010.

[Sko11] Sergei Skorobogatov. Synchronization method for SCA and fault attacks. *J. Cryptographic Engineering*, 1(1):71–77, 2011.

[SMKM18] Raphael Spreitzer, Veelasha Moonsamy, Thomas Korak, and Stefan Mangard. Systematic classification of side-channel attacks: A case study for mobile devices. *IEEE Communications Surveys and Tutorials*, 20(1):465–488, 2018.

[STM] STMicroelectronics. RM0313 Reference manual. `http://www.st.com/resource/en/reference_manual/dm00041563.pdf`.

[STM16] STMicroelectronics. STMicroelectronics Reports 2015 Fourth Quarter and Full Year Financial Results, January 2016. `http://www.st.com/web/en/resource/corporate/financial/quarterly_report/c2792.pdf`.

[Tex] Texas Instruments. MSP430 ultra-low-power MCUs - Applications. `http://www.ti.com/microcontrollers/msp430-ultra-low-power-mcus/applications.html`.

[TM17] Niek Timmers and Cristofaro Mune. Escalating privileges in linux using voltage fault injection. In *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2017, Taipei, Taiwan, September 25, 2017*, pages 1–8, 2017.

[USB] USB Implementers Forum, Inc. Device Class Specification for Device Firmware Upgrade. `http://www.usb.org/developers/docs/devclass_docs/DFU_1.1.pdf` (Version 1.1 - Aug 5, 2004).

[VF05] Federico Vincis and Luca Fanucci. A trimmable rc-oscillator for automotive applications, with low process, supply, and temperature sensitivity. In *12th IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2005, Gammarth, Tunisia, December 11-14, 2005*, pages 1–4, 2005.

[YSW18] Bilgiday Yuce, Patrick Schaumont, and Marc Witteman. Fault attacks on secure embedded software: Threats, design, and evaluation. *Journal of Hardware and Systems Security*, pages 1–20, 2018.

[ZDC+12] Loic Zussa, Jean-Max Dutertre, Jessy Clédiere, Bruno Robisson, Assia Tria, et al. Investigation of timing constraints violation as a fault injection means. In *27th Conference on Design of Circuits and Integrated Systems (DCIS), Avignon, France*, 2012.

[ZDCR14] Loïc Zussa, Jean-Max Dutertre, Jessy Clédière, and Bruno Robisson. Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter. In *2014 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2014, Arlington, VA, USA, May 6-7, 2014*, pages 130–135, 2014.