On the susceptibility of Texas Instruments SimpleLink platform microcontrollers to non-invasive physical attacks[⋆]

Lennert Wouters, Benedikt Gierlichs, and Bart Preneel

imec-COSIC, KU Leuven, Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium
`firstname.lastname@esat.kuleuven.be`

Abstract. We investigate the susceptibility of the Texas Instruments SimpleLink platform microcontrollers to non-invasive physical attacks. We extracted the ROM bootloader of these microcontrollers and then analysed it using static analysis augmented with information obtained through emulation. We demonstrate a voltage fault injection attack targeting the ROM bootloader that allows to enable debug access on a previously locked microcontroller within seconds. Information provided by Texas Instruments reveals that one of our voltage fault injection attacks abuses functionality that is left over from the integrated circuit manufacturing process. The demonstrated physical attack allows an adversary to extract the firmware (i.e. intellectual property) and to bypass secure boot. Additionally, we mount side-channel attacks and differential fault analysis attacks on the hardware AES co-processor. To demonstrate the practical applicability of these attacks we extract the firmware from a Tesla Model 3 key fob.
This paper describes a case study covering Texas Instruments SimpleLink microcontrollers. Similar attack techniques can be, and have been, applied to microcontrollers from other manufacturers. The goal of our work is to document our analysis methodology and to ensure that system designers are aware of these vulnerabilities. They will then be able to take these into account during the product design phase. All identified vulnerabilities were responsibly disclosed.

Keywords: SimpleLink · Firmware recovery · Fault injection · Side-channel analysis.

1 Introduction

Embedded devices are often interconnected using a broad variety of wireless technologies. Texas Instruments (TI) offers the SimpleLink microcontroller platform to enable the development of such connected embedded devices. The platform offers code portability, enabling the reuse of the same code (i.e. Intellectual

[⋆] This preprint has not undergone peer review (when applicable) or any post-submission improvements or corrections. The Version of Record of this contribution is published in COSADE 2022, and is available online at https://doi.org/[insert DOI]

Property (IP)) on a variety of microcontrollers with different functionalities. The microcontroller lineup includes Bluetooth Low Energy (BLE) and Wi-Fi enabled microcontrollers but also sub-1 GHz and multi-protocol enabled microcontrollers. According to TI these microcontrollers are suitable for a wide variety of applications ranging from home automation to automotive and medical applications as well as critical infrastructure applications [56].

TI advertises the SimpleLink microcontrollers to implement secure boot and other security features that allow protection of user data and IP [52, 53]. While TI does not claim any resistance to physical attacks for the SimpleLink platform microcontrollers, it is clear that many of the products designed using these microcontrollers will be deployed in an open and possibly hostile environment.

This hostile environment may comprise physical attackers who want to extract IP or compromise the secure boot chain. Extracting the IP or firmware also enables remote attackers to more easily identify application specific software vulnerabilities [17, 48, 58]. Therefore, performing a physical attack on a single device can lead to remote attacks that scale without having to perform the physical attack on each device [25, 45, 59].

In this paper we take on the role of the physical attacker and use non-invasive physical attacks. We use Voltage Fault Injection (VFI) to extract the contents of non-volatile memory of SimpleLink microcontrollers that have all debug functionalities disabled. Additionally, we demonstrate secret key extraction by targeting the Advanced Encryption Standard (AES) hardware accelerator using Side-Channel Analysis (SCA) and Differential Fault Analysis (DFA).

While the physical attacker is outside of the attacker model used by TI it is still valuable to assess the physical security of these products. Furthermore, we argue that the purpose of debug security features is to protect from an attacker who has physical access to the device, as those debug features are only available to someone who already has physical access. The analysis presented in this work allows device manufacturers to make better informed decisions during their initial threat modeling phase.

1.1 Related work

Embedded systems have been the subject of physical attacks for over 20 years [2, 3, 27, 29]. Nevertheless, those same physical attack techniques can still be used today to extract secret information from many embedded systems.

The passive physical attacker has physical access to the device and observes its normal operation. For example, when performing a physical side-channel attack an adversary will observe one or multiple physical properties of the device. These physical properties, or side-channels, can be analysed to extract secret information from the device under attack [27]. Over the years researchers have shown that side-channels such as instantaneous power consumption [27, 29], ElectroMagnetic (EM) emanations [16, 43], execution time [26], temperature and photonic emissions [14] can all be used to recover secret information from a target device. In most cases statistical analysis is used to extract secret information from the side-channel measurements. Among the most widely used techniques are Differential Power Analyis (DPA) [27], Correlation Power Analysis (CPA) [6], Template Attacks [8] and more recently machine learning techniques [23].

The active physical attacker tries to transiently or permanently disrupt the device's normal operation. The most common non-invasive techniques for fault injection include Voltage Fault Injection (VFI) [29], clock glitching [1, 29] and EM Fault Injection (EMFI) [10, 29]. However, for some modern chip packaging standards (e.g. Wafer Level Chip Scale Package (WLCSP) and Flip Chip Ball Grid Array (FCBGA)) the list of non-invasive techniques can be extended with techniques that used to be considered semi-invasive such as optical fault injection [50] and body bias injection [33, 40].

Both passive and active physical attacks have been demonstrated to be applicable in real world scenarios. Embedded microcontrollers are frequently the target of attacks, as they can contain proprietary code and hardcoded secrets. Consequently, it is not uncommon that a physical attack mounted on a single device leads to a system-wide compromise, these are also known as break-once run everywhere (BORE) attacks.

Goodspeed demonstrated a practical timing side-channel attack targeting the TI MSP430 microcontroller's BootStrap Loader (BSL), allowing to recover the BSL password [20, 21]. Meriac extracted firmware from a Microchip PIC18F microcontroller by erasing a single block of program memory and loading it with a program to dump the remaining blocks [34]. Similarly, the popular STMicroelectronics STM32 series of microcontrollers has been the subject of multiple physical attacks that aim to bypass code readout protection features [37, 38, 47].

Practical side-channel attacks against Microchip's KeeLoq cipher, used in vehicle immobilisers and remote keyless entry products, were demonstrated on software and hardware implementations [13, 25]. In some cases these side-channel attacks allowed to recover the master key, effectively compromising all devices of the same manufacturer by mounting a single side-channel attack [25]. Wouters et al. performed several practical attacks on DST80-based immobiliser systems [59]. The authors demonstrated fault injection attacks to bypass debug security features in automotive microcontrollers, and mounted both unprofiled and profiled side-channel attacks that allowed to extract cryptographic keys. For some DST80-based deployments the authors were able to compromise every immobiliser after carrying out physical attacks on a single device. Van den Herrewegen et al. demonstrated practical attacks targeting embedded bootloaders of several commercially available microcontrolllers [22]. Additionally, they provide a list of anti-patterns that can help guide the design of secure implementations.

Countless additional examples of physical attacks are available online [31, 18, 32] and in the academic literature [4, 5, 9, 35, 44], unfortunately we cannot cover all of them here. Interested readers can find more examples of physical attacks performed on embedded devices in the review paper by Shepherd et al. [49].

1.2 Contributions

The contributions of this paper can be summarised as follows:

- **ROM bootloader analysis.** We extracted and analysed the ROM bootloader of two SimpleLink microcontrollers. Our analysis includes emulating the ROM bootloader to augment static analysis, and revealed two potential avenues for fault injection to bypass code readout protection. One of these code paths is, under normal circumstances, only used during the integrated circuit manufacturing process.
- **Voltage fault injection to enable debug features.** We perform two voltage fault injection attacks allowing to enable debugging features on a previously locked down microcontroller. These physical attacks allow to bypass all IP protection functionality and secure boot features provided by the manufacturer. We perform both attacks on two distinct development boards with different microcontrollers of the SimpleLink series and demonstrate firmware recovery on a commercial product.
- **Side-channel analysis and differential fault analysis on hardware AES.** We mount a successful correlation power analysis attack on the hardware AES implementation included in these microcontrollers. Additionally, we successfully perform differential fault analysis on the hardware AES coprocessor.
- **Open-source implementations.** We provide open-source Python notebooks that can be used to reproduce and extend the experiments covered in this paper[1].

2 Experimental setup

In Sect. 1 we introduced the SimpleLink platform and noted the similarity between the microcontrollers, and the portability of IP. Given these similarities it is likely that there are also similarities in the underlying hardware of the SimpleLink microcontrollers. While we cannot evaluate all 137 parts that are offered as part of the SimpleLink platform, it is likely that the physical attacks documented in this work can be adapted to work on most SimpleLink parts.

Nevertheless, the experiments documented in this work were performed on two distinct microcontrollers that are representative for the entire CC13xx and CC26xx lineup [55]. The first target is a CC2640R2F BLE microcontroller, the main application firmware is executed by an ARM Cortex-M3 CPU. Later, in Sect. 4.4 we also extract the firmware from the automotive variant, the CC2640R2F-Q1. The second target is a CC2652R1F multiprotocol wireless microcontroller using an ARM Cortex-M4F CPU to execute the main application firmware. By default both microcontrollers run at a clock frequency of 48 MHz generated by an internal RC oscillator or derived from an external crystal oscillator. Both microcontrollers use the internal RC oscillator during the execution of the ROM bootloader. Note that both targets include a secondary ARM Cortex-M0 CPU that is responsible for the lower level RF communications.

All of the physical attacks covered in this work were evaluated on commercially available development kits of the target microcontrollers. Throughout this

[1] https://github.com/KULeuven-COSIC/SimpleLink-FI

work we use the NewAE ChipWhisperer Husky platform to acquire side-channel traces and to perform voltage fault injection. Similar results were obtained using the open-source NewAE ChipWhisperer-Lite [41].

2.1 Target modifications

All microcontrollers in the CC13xx and CC26xx lineup use a similar power supply configuration [55]. Most notably, an internal low-dropout regulator is used to generate the 1.28 V supply for the ARM Cortex CPU core. This internal core voltage rail is exposed on the `DCOUPL` pin of the package to add an external decoupling capacitor.

The availability of the internal CPU core voltage on an external pin of the microcontroller is convenient for the non-invasive physical attacker. As a result we choose to focus on voltage fault injection attacks and power side-channel analysis, but similar results are likely achievable using other techniques. To perform voltage fault injection we can momentarily short the core voltage supply to ground; this is also known as a crowbar voltage glitch [39]. Similarly, we can use an external power supply to supply our own voltage (larger than 1.28 V): this will disable the internal regulator and will allow us to measure the instantaneous power consumption over a shunt resistor.

For the experiments covered in this paper we modified two development boards, namely a LAUNCHXL-CC26x2R1 and LAUNCHXL-CC2640R2. Figure 1 shows the modified development boards. In both cases we removed the capacitor connected to the reset pin (C20) and the decoupling capacitor connected to the `DCOUPL` pin (C19). We also added a 10 Ohm shunt resistor and an SMA connector to the `DCOUPL` pin. The SMA connector can be connected to the ChipWhisperer for both fault injection and side-channel analysis.

Side-channel measurements were acquired using a sample rate of 240 Megasamples Per Second (MSPS) while supplying 1.45 V to the `DCOUPL` pin using an external power supply. All fault injection experiments were performed with the ChipWhisperer configured to use a 200 MHz clock, resulting in a glitch offset and glitch width resolution of 5 ns. The targets were connected to the ChipWhisperer using a 50 cm SMA cable; note that the length of this cable can influence the glitch parameters.

3 The ROM bootloader

The SimpleLink microcontrollers include a bootloader that is stored in Read-Only Memory (ROM). This bootloader is executed after an initial power-up or reset of the microcontroller. The ROM bootloader is responsible for initialising the microcontroller and enables or disables certain features based on settings stored in the Customer Configuration (CCFG) page, the Factory Configuration (FCFG) page and eFuses. As their names suggest, the CCFG can be programmed by the customer or device manufacturer, the FCFG is programmed by TI and cannot be modified by the customer. Both of these configuration pages are stored

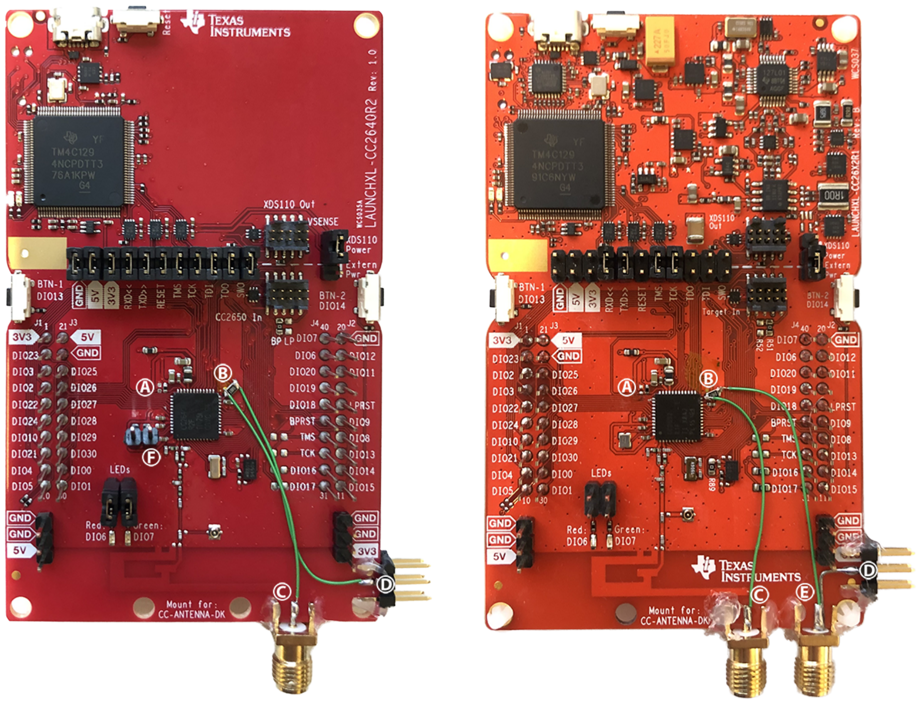

Fig. 1. The LAUNCHXL-CC2640R2 (left) and LAUNCHXL-CC26x2R1 (right) development boards modified for side-channel analysis and voltage fault injection. Both boards have C20 and C19 removed, indicated by `A` and `B` respectively. The SMA connector indicated by `C` allows to capture the voltage drop over the 10 Ohm shunt resistor inserted at `B`. We supply 1.45 V through connection `D` while acquiring side-channel measurements, which disables the internal regulator. SMA connector `E` is optional and allows for differential measurements. Finally, we removed the crystal oscillator on the LAUNCHXL-CC2640R2 (left), which allows us to supply our own clock for synchronous sampling; this modification is optional and is indicated by `F`. The SMA connectors are grounded on the bottom side of the boards.

in the internal and non-volatile flash memory of the microcontroller. Similarly, eFuses are blown by Texas Instruments during chip manufacturing, but their state can be read by customers. The ROM code additionally implements a serial bootloader interface that allows to perform basic operations such as reading and writing memory. The serial interface is only started when no valid firmware image is present in flash memory, or when the bootloader backdoor functionality is enabled and used [54].

Many of the security features implemented by the SimpleLink microcontrollers rely on an unaltered behaviour of the ROM bootloader. Debug security features allow a developer to disable access to the serial bootloader interface and to disable the Debug Access Port (DAP); these settings are stored in the CCFG and are parsed by the ROM bootloader. Disabling these debug features is paramount for the IP protection and secure boot features [53]. Texas Instruments recommends disabling the bootloader serial interface and the DAP in the CCFG of production hardware [54].

3.1 Extracting and analysing the ROM bootloader

As indicated earlier, the ROM bootloader is responsible for disabling debugging features. The ability to analyse this code is helpful to gain a better understanding of how certain features are enabled or disabled, and to identify potential vulnerabilities. However, we have to obtain a copy of the ROM bootloader code before we can analyse it.

We implemented a basic Python module that allows communication with the bootloader's serial interface over UART. Using this bootloader interface it is possible to read memory, including the ROM that stores the bootloader itself. We dumped the ROM bootloader from both of our target microcontrollers on development boards over which we had full control. In both cases the ROM could be extracted by reading data starting at address `0x10000000`.

We used the free and open-source Ghidra software reverse engineering tool to statically analyse the ROM code. We used the SVD-loader plugin [46] to automatically populate the Ghidra memory map for our target, including all documented peripheral registers. Afterwards we were able to identify the code responsible for disabling debug interfaces, by searching for references to the `CCFG:TAP_DAP_x` fields.

This initial analysis revealed that, in the case of the CC2640R2F, the `AON_WUC:JTAGCFG` register is used to enable or disable the Joint Test Action Group (JTAG) interface. The equivalent register for the CC2652R1F is referred to as `AON_PMCTL:JTAGCFG`. In the remainder of this paper we will refer to both registers by `JTAGCFG`. According to the publicly available documentation the least significant byte of the `JTAGCFG` is reserved. By reading the value of this register when JTAG is disabled and when JTAG is enabled, it is clear that these lower bits are used to enable or disable specific TAPs and DAPs. This observation will later help us to speed up initial fault injection campaigns in Sect. 4.2.

3.2 ROM bootloader emulation

To further extend the available information during static analysis we also emulated the ROM bootloader using Unicorn engine's Python bindings. We parsed a System View Description (SVD) file for our target to automatically generate the correct memory mappings; this is similar to what the SVD-loader plugin does in Ghidra. Additionally we have to manually guide the emulation the first time, as the bootloader may get stuck waiting for (non-emulated) peripherals or interrupts. Once these hurdles are identified they can be overcome by registering simple callback functions or patching the bootloader code. To ensure that all debug security related features were emulated correctly we also loaded valid CCFG and FCFG flash regions in the emulator. Additionally, we read the eFuse memory of our target using the SDK functions provided by TI and emulated the peripheral.

Execution coverage traces of the emulated bootloader can be exported and visualised in Ghidra using plugins such as Dragon Dance [24] or Emerald [36]. This visualization highlights parts of the code executed during normal operation and makes it easier to understand the execution flow of the bootloader. This visualization helped us to identify a, normally unused, code path which writes `0x6f` to the `JTAGCFG` register.

Figure 2 summarises our analysis of the bootloader and depicts the security-related actions performed by the bootloader. The first decision made by the ROM bootloader is based on the value of the ninth eFuse row. In commercially available chips the most significant fuse bit in this fuse row will be blown (i.e. set to 1). This results in the `JTAGCFG` register being set to zero, disabling all TAPs/DAPs. The alternative execution path writes 0x6F to the `JTAGCFG` register and sets a GPIO pin high. Information provided by TI indicates that this functionality is used during the integrated circuit manufacturing process. Note that subsequent updates of the `JTAGCFG` do not clear any bits that are already set. Afterwards, the FCFG and CCFG are parsed, and a TAP or DAP will only be enabled if it is enabled in both configurations.

The emulated bootloader can be further expanded to simulate fault injection or to fuzz the serial command interface using e.g. AFL++ in Unicorn mode [15]. Additionally, a side-channel trace can be emulated and compared to real side-channel measurements to determine approximate offsets in time for glitch attempts [30].

As part of the paper's artifacts we provide example code to communicate with the ROM bootloader's serial interface. Additionally, we provide information on how to load an extracted ROM bootloader in Ghidra and how to emulate the bootloader using the Unicorn engine.

4 Bypassing debug security

We will assume an attack scenario in which the target device is in the most locked down state possible. In other words the target is configured to disable the

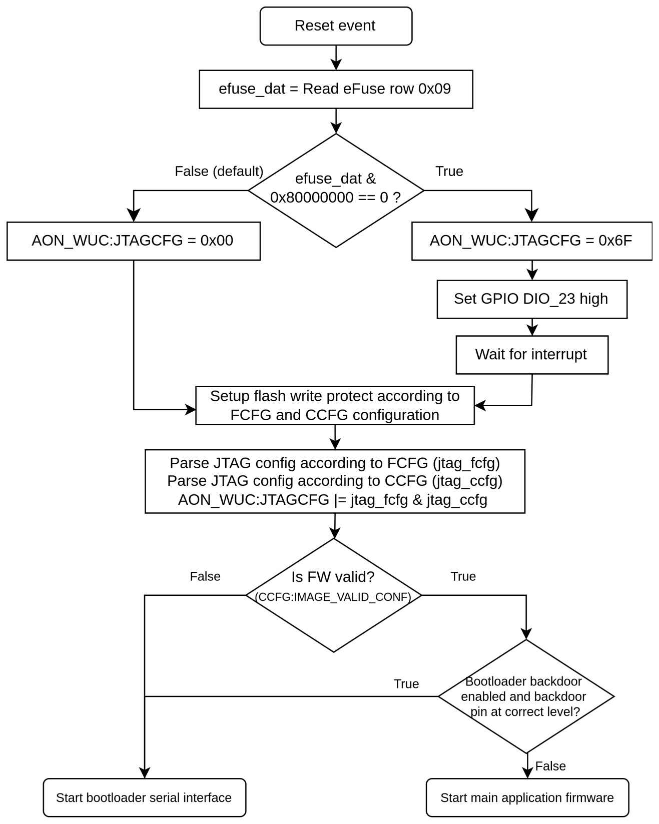
Reset event
efuse_dat = Read eFuse row 0x09
efuse_dat & 0x80000000 == 0 ?
False (default)
True
AON_WUC:JTAGCFG = 0x00
AON_WUC:JTAGCFG = 0x6F
Set GPIO DIO_23 high
Wait for interrupt
Setup flash write protect according to FCFG and CCFG configuration
Parse JTAG config according to FCFG (jtag_fcfg)
Parse JTAG config according to CCFG (jtag_ccfg)
AON_WUC:JTAGCFG |= jtag_fcfg & jtag_ccfg
Is FW valid? (CCFG:IMAGE_VALID_CONF)
False
True
Bootloader backdoor enabled and backdoor pin at correct level?
True
False
Start bootloader serial interface
Start main application firmware

Fig. 2. Simplified ROM bootloader execution flowchart. This flowchart is based on the ROM bootloader extracted from a CC2640R2F chip. Note that the bootloader will additionally check if the bootloader serial interface option is enabled in the `CCFG:BL_CONFIG` register before executing incoming commands.

serial bootloader interface and all JTAG access is disabled. An attack that is able to compromise a device in this state will also work for a device using a less secure configuration. In this scenario our goal as the adversary is to disable the debug security features, as this allows us to completely compromise the device. On the one hand such an attack allows us to obtain a copy of the firmware stored in flash (i.e. the IP). This enables device cloning, vulnerability research and can expose device secrets. On the other hand, such an attack also compromises the secure boot functionality and any security feature relying on secure boot (e.g. remote attestation) [53].

Section 3, and in particular Fig. 2, reveal two potential avenues for enabling JTAG access using fault injection. We can try to inject a glitch during the first decision in the flowchart or during parsing of the CCFG JTAG configuration.

In the remainder of this section we will determine a suitable range of glitch parameters. Afterwards we evaluate voltage fault injection as a means to bypass debug security by targeting the CCFG parsing and the debug security eFuse check.

4.1 Determining a suitable glitch width

A crowbar voltage glitch is characterised by two main parameters: the glitch width and and the glitch offset [39]. The glitch width is the amount of time the glitch MOSFET is enabled. The glitch offset is the offset in time from a reference signal.

We initially used a development board over which we have full control to determine a suitable range for the glitch width. We used a common fault injection target program consisting of two nested for loops that increment a counter value [7, 39, 40]. Figure 3 shows the results when targeting this dummy program. The most promising glitch width was selected as the one resulting in most faults overall (i.e. incorrect counter output), and is used as an initial value in the remaining experiments. Enabling the glitch MOSFET for 100 ns resulted in the most faulted counter outputs on the CC2640R2F using our setup. For the CC2652R1F we determined that a 610 ns glitch width resulted in most faulted outputs.

Note from Fig. 3 that for the CC2640R2F a narrow range (approximately 70 ns to 110 ns) of glitch widths is applicable; a smaller width will not produce a faulty output and a longer glitch will always crash the device. The CC2652R1F produces faulty counter outputs over a glitch width range of approximately 90 ns to 730 ns. Presumably these differences are related to the different underlying micro-architectures (Cortex-M3 versus Cortex-M4F).

4.2 Debug security bypass: CCFG configuration parsing

From our analysis of the ROM bootloader we know that the JTAG configuration is read from the CCFG. Depending on the values stored in the `CCFG:CCFG_TAP_DAP_x`

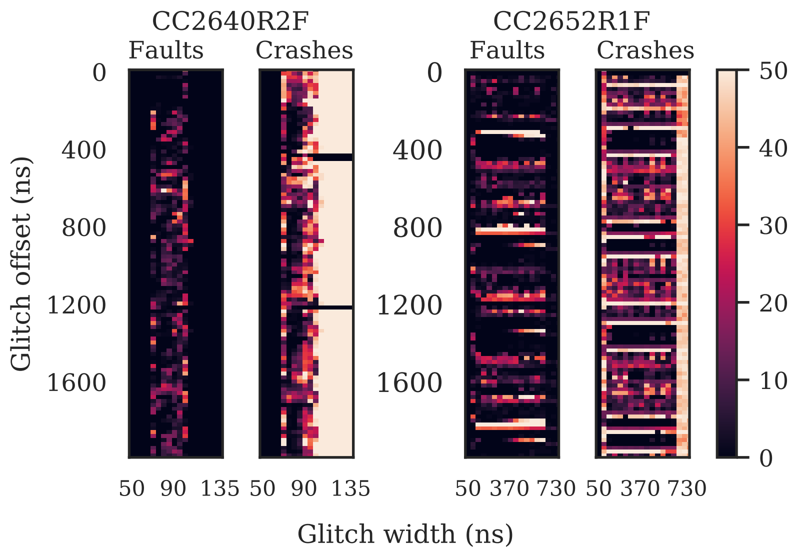

Fig. 3. Number of faulty counter outputs and crashes per combination of glitch offset and glitch width. We performed 50 attempts for each combination of glitch parameters, for a total of 90 k attempts per device. Note that the range of glitch widths (x-axis) is not the same for both targets.

registers certain parts of the JTAG interface are enabled or disabled. As the adversary we will attempt to inject transient faults by momentarily shorting the core supply to ground while the ROM bootloader is parsing the CCFG.

We used side-channel analysis to determine the approximate offset in time from the reset signal when the ROM bootloader would be parsing the CCFG JTAG configuration. Figure 4 depicts two power traces from each of our targets that cover the execution of the ROM bootloader after the microcontroller has been reset. A noticeable difference can be observed between the black power traces, corresponding to the execution of the ROM bootloader when a valid firmware image is present in flash, and the gray power traces corresponding to the execution of the ROM bootloader when the flash is erased (i.e. invalid firmware).

Recall from Fig. 2 that an erased microcontroller will start the serial bootloader interface and will be waiting for incoming commands over a UART or SPI interface. The configuration of the `JTAGCFG` register based on the CCFG and FCFG is performed right before the serial interface is started. When enumerating the glitch offset we thus work our way back from the point where the two traces deviate.

In a realistic setting an adversary would have to reset the target microcontroller, inject a glitch and attempt to connect to the target using a JTAG debugger to verify whether the glitch was successful. Connecting to the target

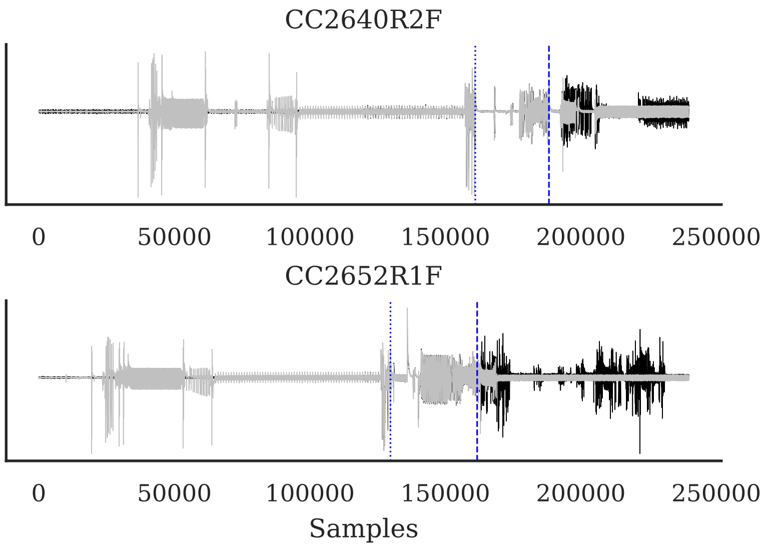

Fig. 4. Power traces covering the ROM bootloader execution of a CC2640R2F (top) and CC2652R1F (bottom) microcontroller. In gray the power trace when the flash memory is empty (i.e. invalid firmware), in black the power trace when a valid firmware image is present in flash. The vertical dotted lines indicate the offset in time when the debug security eFuse is checked. The vertical dashed lines indicate parsing of the CCFG debug security settings; note that this line is close to when the power traces start deviating. The power traces in this figure were acquired at 200 MSPS.

using JTAG is relatively slow, and limits the rate at which attempts can be made. To speed up the initial glitch parameter enumeration we used a custom firmware image that sends the contents of the `JTAGCFG` register over UART to a control PC. In this way we can determine the state of the JTAG peripheral without having to connect using a JTAG debugger. Recall from our analysis of the ROM bootloader that parts of the JTAG peripheral are enabled if the ROM bootloader writes a non-zero value to the `JTAGCFG` register.

After basic glitch parameter enumeration we identified that glitching the CC2640R2F between 188,300 and 188,400 cycles (of the 200 MHz ChipWhisperer clock) after the reset signal goes high is likely to result in enabling JTAG access. We were able to obtain a success rate of approximately 5%. For the CC2652R1F we determined this offset to be between 161,700 and 162,000 cycles. On this target we achieved a success rate of approximately 1%. The glitch offsets are visualised in the side-channel traces shown in Fig. 4.

Using the aforementioned trick to speed up glitch attempts we could perform 10 glitch attempts per second on both targets. In a more realistic scenario in which the adversary tries to connect to the chip using a JTAG debugger the rate is reduced. When using the XDS110 debugger available on the development board in combination with the UniFlash command line interface we were able to perform one glitch attempt every 2.5 seconds.

4.3 Debug security bypass: eFuse readout

Figure 2 reveals a more interesting avenue for fault injection, namely the first decision. Recall from Sect. 3.2 that in a normal scenario the false branch is taken, setting `JTAGCFG` to 0. Our goal is to use voltage fault injection to divert the ROM bootloader execution into the true branch, enabling access to all JTAG TAPs and DAPs. Conveniently the ROM bootloader will signal that our fault injection attempt was successful by pulling GPIO pin 23 high. This means that, even in a realistic scenario, we can inject glitches at a much higher rate. In our experiments we were able to perform up to 100 glitch attempts per second.

By enumerating glitch parameters we found that the CC2640R2F can be forced to take the alternative execution path by injecting a glitch between 161,100 and 161,200 cycles after releasing the microcontroller from reset. We found that slightly increasing the glitch width to 115 ns resulted in a success rate of 10%. With the ability to inject 100 glitches per second and a success rate of 10% it should not take more than a second to successfully enable all debugging features.

The same experiment was also performed on the CC2652R1F target, and successfull glitches were injected at offsets between 129,800 and 129,900 clock cycles. Similar to the previous experiments the glitch width did not seem to have a big impact on the success rate for the CC2652R1F target. Additionally we observed a success rate of approximately 0.1%. Even though this success rate is significantly lower, it would not take more than a few seconds to to enable all debugging features.

4.4 Extracting firmware from the Tesla Model 3 key fob

We extracted the firmware from a Tesla Model 3 key fob to evaluate the practical applicability of our attack. The key fob uses the CC2640R2F-Q1 chip, the automotive variant of the chip we targeted before. All debugging features were disabled, so simply reading the firmware using a debugger was not possible.

While it may be possible to perform our debug security bypass attack in-circuit, we chose to desolder the chip from a target key fob. We also removed the CC2640R2F from our LAUNCHXL-CC2640R2 development board and replaced it with the CC2640R2F-Q1 from the key fob. Using this setup and our previously gained knowledge about the target chip it was straightforward to enable the debug functionality. This allowed us to recover the proprietary firmware stored in the key fob.

Recovering this firmware enables vulnerability research that may result in a practical attack [58]. In the context of this work we performed some basic static analysis of the firmware in Ghidra, and identified an AES key stored in flash memory. Further analysis is required to determine the purpose of this key, but it may be used to protect the confidentiality of firmware updates. We note that the key fob firmware appears to also verify the authenticity of incoming firmware updates, and that a separate secure element is likely responsible for performing cryptographic operations related to unlocking or starting the car.

5 The hardware AES co-processor

The SimpleLink microcontrollers come with a hardware AES co-processor. According to the technical reference manual one AES operation takes $2 + 3 \cdot r$ clock cycles, where r denotes the number of rounds [54]. One AES-128 operation thus takes 32 clock cycles to complete, and the implementation operates on the full 128-bit state.

In this section we perform side-channel analysis and differential fault analysis of the hardware AES co-processor. In both scenarios we target AES-128 in a simple evaluation program that allows us to transmit the key and plaintext for one block operation to the target over UART. We modified the standard library code to insert a GPIO trigger signal as close to the AES operation as possible. The main reason for this modification is to ensure that our glitches are inserted during the actual AES operation instead of during the hardware accelerator setup.

5.1 Side-channel analysis

We used known-key analysis to determine the leakage model of the target implementation [28]. To that end we acquired a set of 100 k side-channel traces with random, but known, plaintexts and keys. For each of these traces we generated all intermediate states and performed CPA with the full state Hamming weight. Additionally, we perform CPA with the Hamming distance between all possible combinations of intermediate states.

The known-key analysis reveals Hamming weight leakage of the plaintext and ciphertext, marking the start and end of the encryption operation respectively. Within each round we observe Hamming distance between the `SubBytes` operation output and the `ShiftRows output`. Additionally, we observe Hamming distance leakage between round r `AddRoundKey` output and round $r+1$ `AddRoundKey` output. Finally, we observe Hamming distance leakage between the `AddRoundKey` operation output in round 9 and the ciphertext, a common leakage model in unprotected hardware implementations that is easily exploitable [42].

During the initial known-key analysis we noticed that the captured traces could be divided into two distinct sets. Traces from the same set can be easily aligned using a sum of absolute difference alignment. However, traces that do not belong to the same set do not align well. To resolve this issue we first split the traces into the two sets and align them; afterwards each set is standardized (zero mean, unit variance scaling). Finally both sets are combined again and used during the attack phase. This preprocessing pipeline was automated in Python and reused for all further attacks.

We mounted CPA attacks targeting the Hamming distance between the ciphertext and the `AddRoundKey` operation output in round 9. Figure 5 shows the average guessing entropy of all 16 subkey bytes, averaged over 50 attacks with a random key for each microcontroller. In total we carried out 100 attacks, using 100 k traces per attack for a total of 10 M traces. Using the segmented memory feature of the ChipWhisperer Husky we were are able to acquire 100 k traces in approximately 1.5 minutes. The traces captured from the CC2640R2F were acquired synchronously by supplying a 12 MHz clock from the ChipWhisperer to the target, internally this clock is divided to a CPU operating frequency of 24 MHz. The CC2652R1F target was running at 48 MHz and traces were acquired asynchronously.

From Fig. 5 it is clear that some key bytes (e.g. byte 10 for the CC2640R2F) can be recovered consistently with a small number of traces. Other key bytes (e.g. byte 12 for both targets) cannot be consistently recovered, so there will be some key enumeration remaining. This also indicates that the Hamming distance model is not optimal in this scenario. The attack performance can likely be further improved using linear regression [11] or (non-)profiled deep learning approaches [57]. Alternatively one could attack multiple key bytes simultaneously, this should reduce the amount of algorithmic noise (recall that the implementation operates on the full state) at the cost of increased computational complexity.

5.2 Differential Fault Analysis

Side-channel analysis revealed Hamming distance leakage between the `SubBytes` operation output and the `ShiftRows output`. This indicates that it may be relatively straightforward to inject a fault before the last `MixColumns` operation. Such faults can be exploited using DFA to recover the cryptographic key [12, 19]. A single byte fault before the last round `MixColumns` operation will result in four faulted bytes in the ciphertext. A valid ciphertext and two such faults for each column are then sufficient to recover the cryptographic key.

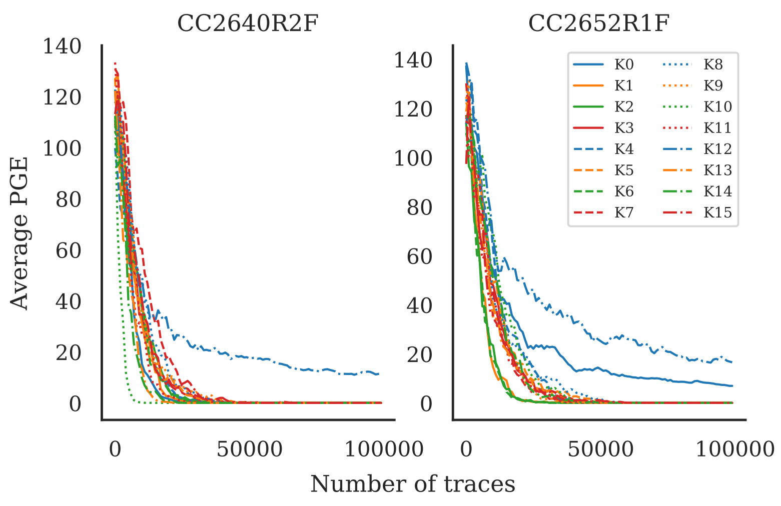

Fig. 5. Partial guessing entropy for all subkey bytes when targeting the hardware AES implementation of the CC2640R2F (left) and for the CC2652R1F (right). In both plots the partial guessing entropy is averaged over 50 experiments. Note that some key bytes are consistently easy to recover with a small number of traces (e.g. key byte 10 for the CC2640R2F). Other key bytes can not be consistently recovered (e.g. key byte 12 in both cases).

We perform voltage fault injection to determine the susceptibility of the hadware AES implementation to such an attack. While injecting faults we can record one valid ciphertext output and all faulted outputs; these faulted outputs were then split based on the number of faulted bytes and their positions within the ciphertext. We used the open source phoenixAES implementation that is part of the SideChannelMarvels project to recover the key from these faulted ciphertexts [51]. Open-source tools such as these demonstrate that an attacker does not necessarily need to understand the underlying key recovery mechanisms. The attack was determined to be successful on both targets and a demonstration Python notebook is provided in the repository.

We note that injecting a glitch before or after the AES operation would often result in the implementation outputting 96 bytes of data, even though our implementation is meant to output only 16 ciphertext bytes. This longer output often contained the full key that was being used for the AES operation. We did not investigate this behaviour further, as we were targeting a rather artificial implementation in which the key was hardcoded within the same function.

6 Conclusion

Physical attacks can be a realistic threat for embedded systems. In many cases these physical attacks can be carried out using commercially available and low-cost equipment. In this work we investigate the susceptibility of Texas Instruments SimpleLink microcontrollers to low-cost non-invasive physical attacks. We extracted the ROM bootloader of these microcontrollers and then analysed it using a combination of static analysis and emulation. Our analysis reveals two potential avenues for a physical attacker to circumvent debug security features.

To demonstrate our findings we perform two voltage fault injection attacks that disrupt the normal execution flow of the ROM bootloader. First, we show that code parsing the debug security settings is susceptible to fault injection. Secondly, we identified an execution path that, according to TI, is only used during the integrated circuit manufacturing process under normal circumstances. We demonstrate that this code can be reached using voltage fault injection, enabling all JTAG test access ports and debug access ports.

In summary, we demonstrate practical voltage fault injection attacks that allow a physical attacker to gain full debugging access on a previously locked down microcontroller. The attacks have a relatively high success rate and can in some cases be executed in a few seconds. They allow to extract the device's firmware, to bypass secure boot features and any other security features building on those. We apply our attacks to a Tesla Model 3 key fob to demonstrate the practical applicability.

Additionally, we investigate the susceptibility of the included hardware AES accelerator to physical attacks. First, we successfully mount a correlation power analysis based side-channel attack on the hardware AES accelerator. Secondly, we demonstrate key recovery by using voltage fault injection to introduce faults that are exploitable using differential fault analysis.

Basic non-invasive physical attacks have become more accessible in recent years, in no small part due to the availability of low cost open-source tooling and training material [41]. Unfortunately the large majority of general purpose microcontrollers lack countermeasures against such physical attacks. The classical threat model in which an adversary who has physical access is able to connect a JTAG debugger, but is not able to perform basic non-invasive physical attacks, may no longer be suitable today.

Similarly, it is important to remember that an adversary will attempt to identify the weakest link when attacking a device. For example, software countermeasures against side-channel attacks can be implemented on a device. A real world adversary may decide to circumvent code readout protection features, allowing to recover the cryptographic key from the firmware. This allows to extract the secret key without having to defeat side-channel countermeasures.

6.1 Responsible disclosure

We first contacted the Texas Instruments Product Security Incident Response Team (PSIRT) on November 6th 2021[2]. The PSIRT indicated that they would be unable to resolve the issues that bypass debug security features as the ROM

[2] Instructions to report security vulnerabilities can be found at https://www.ti.com/security

is immutable, and noted that these products were not designed to withstand physical attacks. As a result of our reports Texas Instruments released a general security advisory to inform customers on the possibility of physical attacks[3].

We also contacted Tesla on November 6th 2021[4]. We recovered an AES key from the extracted firmware, but this by itself was not deemed a security issue. The recovered key may be used to protect the confidentiality (but not the authenticity) of firmware updates. The key fob also uses an additional secure element that is likely used when unlocking and starting the car.

Acknowledgements We want to thank the Texas Instruments and Tesla product security incident response teams for their responsiveness. This work was supported in part by CyberSecurity Research Flanders with reference number VR20192203. In part by the Research Council KU Leuven C1 on Security and Privacy for Cyber-Physical Systems and the Internet of Things with contract number C16/15/058. In addition, this work was supported by the European Commission through the Horizon 2020 research and innovation programme under grant agreement Cathedral ERC Advanced Grant 695305, under grant agreement H2020-FETFLAG-2018-03-820405 QRANGE and under grant agreement H2020-DS-LEIT-2017-780108 FENTEC.

References

1. Agoyan, M., Dutertre, J., Naccache, D., Robisson, B., Tria, A.: When clocks fail: On critical paths and clock faults. In: Gollmann, D., Lanet, J., Iguchi-Cartigny, J. (eds.) Smart Card Research and Advanced Application, 9th IFIP WG 8.8/11.2 International Conference, CARDIS 2010, Passau, Germany, April 14-16, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6035, pp. 182–193. Springer (2010). https://doi.org/10.1007/978-3-642-12510-2_13, https://doi.org/10.1007/978-3-642-12510-2_13
2. Anderson, R., Kuhn, M.: Tamper resistance-a cautionary note. In: Proceedings of the second USENIX workshop on electronic commerce. vol. 2, pp. 1–11 (1996)
3. Anderson, R.J., Kuhn, M.G.: Low cost attacks on tamper resistant devices. In: Christianson, B., Crispo, B., Lomas, T.M.A., Roe, M. (eds.) Security Protocols, 5th International Workshop, Paris, France, April 7-9, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1361, pp. 125–136. Springer (1997). https://doi.org/10.1007/BFb0028165, https://doi.org/10.1007/BFb0028165
4. Balasch, J., Gierlichs, B., Verdult, R., Batina, L., Verbauwhede, I.: Power Analysis of Atmel CryptoMemory - Recovering Keys from Secure EEPROMs. In: Dunkelman, O. (ed.) Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7178, pp. 19–34. Springer (2012). https://doi.org/10.1007/978-3-642-27954-6_2, https://doi.org/10.1007/978-3-642-27954-6_2

[3] The advisory can be found online at https://www.ti.com/lit/pdf/swra739

[4] Instructions to report security vulnerabilities can be found at https://www.tesla.com/legal/security?redirect=no

5. Bozzato, C., Focardi, R., Palmarini, F.: Shaping the glitch: Optimizing voltage fault injection attacks. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(2), 199–224 (2019). https://doi.org/10.13154/tches.v2019.i2.199-224, https://doi.org/10.13154/tches.v2019.i2.199-224
6. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings. Lecture Notes in Computer Science, vol. 3156, pp. 16–29. Springer (2004). https://doi.org/10.1007/978-3-540-28632-5_2, https://doi.org/10.1007/978-3-540-28632-5_2
7. Carpi, R.B., Picek, S., Batina, L., Menarini, F., Jakobovic, D., Golub, M.: Glitch it if you can: Parameter search strategies for successful fault injection. In: Francillon, A., Rohatgi, P. (eds.) Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8419, pp. 236–252. Springer (2013). https://doi.org/10.1007/978-3-319-08302-5_16, https://doi.org/10.1007/978-3-319-08302-5_16
8. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2523, pp. 13–28. Springer (2002). https://doi.org/10.1007/3-540-36400-5_3, https://doi.org/10.1007/3-540-36400-5_3
9. Cui, A., Housley, R.: BADFET: defeating modern secure boot using second-order pulsed electromagnetic fault injection. In: Enck, W., Mulliner, C. (eds.) 11th USENIX Workshop on Offensive Technologies, WOOT 2017, Vancouver, BC, Canada, August 14-15, 2017. USENIX Association (2017), https://www.usenix.org/conference/woot17/workshop-program/presentation/cui
10. Dehbaoui, A., Dutertre, J., Robisson, B., Tria, A.: Electromagnetic transient faults injection on a hardware and a software implementations of AES. In: Bertoni, G., Gierlichs, B. (eds.) 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, September 9, 2012. pp. 7–15. IEEE Computer Society (2012). https://doi.org/10.1109/FDTC.2012.15, https://doi.org/10.1109/FDTC.2012.15
11. Doget, J., Prouff, E., Rivain, M., Standaert, F.: Univariate side channel attacks and leakage modeling. J. Cryptogr. Eng. **1**(2), 123–144 (2011). https://doi.org/10.1007/s13389-011-0010-2, https://doi.org/10.1007/s13389-011-0010-2
12. Dusart, P., Letourneux, G., Vivolo, O.: Differential fault analysis on A.E.S. In: Zhou, J., Yung, M., Han, Y. (eds.) Applied Cryptography and Network Security, First International Conference, ACNS 2003. Kunming, China, October 16-19, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2846, pp. 293–306. Springer (2003). https://doi.org/10.1007/978-3-540-45203-4_23, https://doi.org/10.1007/978-3-540-45203-4_23
13. Eisenbarth, T., Kasper, T., Moradi, A., Paar, C., Salmasizadeh, M., Shalmani, M.T.M.: On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoq Code Hopping Scheme. In: Wagner, D.A. (ed.) Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings. Lecture Notes in Computer

Science, vol. 5157, pp. 203–220. Springer (2008). https://doi.org/10.1007/978-3-540-85174-5_12, https://doi.org/10.1007/978-3-540-85174-5_12

14. Ferrigno, J., Hlavác, M.: When AES blinks: introducing optical side channel. IET Inf. Secur. **2**(3), 94–98 (2008). https://doi.org/10.1049/iet-ifs:20080038, https://doi.org/10.1049/iet-ifs:20080038
15. Fioraldi, A., Maier, D., Eißfeldt, H., Heuse, M.: AFL++ : Combining incremental steps of fuzzing research. In: Yarom, Y., Zennou, S. (eds.) 14th USENIX Workshop on Offensive Technologies, WOOT 2020, August 11, 2020. USENIX Association (2020), https://www.usenix.org/conference/woot20/presentation/fioraldi
16. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2162, pp. 251–261. Springer (2001). https://doi.org/10.1007/3-540-44709-1_21, https://doi.org/10.1007/3-540-44709-1_21
17. Garbelini, M.E., Wang, C., Chattopadhyay, S., Sun, S., Kurniawan, E.: SweynTooth: Unleashing Mayhem over Bluetooth Low Energy. In: Gavrilovska, A., Zadok, E. (eds.) 2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020. pp. 911–925. USENIX Association (2020), https://www.usenix.org/conference/atc20/presentation/garbelini
18. Gerlinksy, C.: Breaking Code Read Protection on the NXP LPC-family Microcontrollers. In: RECON. Brussels, Belgium (2017)
19. Giraud, C.: DFA on AES. In: Dobbertin, H., Rijmen, V., Sowa, A. (eds.) Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers. Lecture Notes in Computer Science, vol. 3373, pp. 27–41. Springer (2004). https://doi.org/10.1007/11506447_4, https://doi.org/10.1007/11506447_4
20. Goodspeed, T.: Practical attacks against the MSP430 BSL. In: Twenty-Fifth Chaos Communications Congress. Berlin, Germany (2008)
21. Goodspeed, T.: A side-channel timing attack of the MSP430 BSL. Black Hat USA (2008)
22. den Herrewegen, J.V., Oswald, D.F., Garcia, F.D., Temeiza, Q.: Fill your boots: Enhanced embedded bootloader exploits via fault injection and binary analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(1), 56–81 (2021). https://doi.org/10.46586/tches.v2021.i1.56-81, https://doi.org/10.46586/tches.v2021.i1.56-81
23. Hospodar, G., Gierlichs, B., Mulder, E.D., Verbauwhede, I., Vandewalle, J.: Machine learning in side-channel analysis: a first study. J. Cryptogr. Eng. **1**(4), 293–302 (2011). https://doi.org/10.1007/s13389-011-0023-x, https://doi.org/10.1007/s13389-011-0023-x
24. Kartal, O.: Dragon Dance. https://github.com/0ffffffffh/dragondance (2020)
25. Kasper, M., Kasper, T., Moradi, A., Paar, C.: Breaking KeeLoq in a Flash: On Extracting Keys at Lightning Speed. In: Preneel, B. (ed.) Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5580, pp. 403–420. Springer (2009). https://doi.org/10.1007/978-3-642-02384-2_25, https://doi.org/10.1007/978-3-642-02384-2_25
26. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings. Lecture Notes in Computer Science,

vol. 1109, pp. 104–113. Springer (1996). https://doi.org/10.1007/3-540-68697-5_9, https://doi.org/10.1007/3-540-68697-5_9
27. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999). https://doi.org/10.1007/3-540-48405-1_25, https://doi.org/10.1007/3-540-48405-1_25
28. Kocher, P.C., Jaffe, J., Jun, B., Rohatgi, P.: Introduction to differential power analysis. J. Cryptogr. Eng. **1**(1), 5–27 (2011). https://doi.org/10.1007/s13389-011-0006-y, https://doi.org/10.1007/s13389-011-0006-y
29. Kömmerling, O., Kuhn, M.G.: Design principles for tamper-resistant smartcard processors. In: Guthery, S.B., Honeyman, P. (eds.) Proceedings of the 1st Workshop on Smartcard Technology, Smartcard 1999, Chicago, Illinois, USA, May 10-11, 1999. USENIX Association (1999), https://www.usenix.org/conference/usenix-workshop-smartcard-technology/design-principles-tamper-resistant-smartcard
30. Ledger-Donjon: Rainbow. https://github.com/Ledger-Donjon/rainbow (2021)
31. LimitedResults: nRF52 Debug Resurrection (APPROTECT Bypass). https://limitedresults.com/2020/06/nrf52-debug-resurrection-approtect-bypass/ (2020), [Online; accessed 9-Dec-2021]
32. Lu, Y.: Attacking Hardware AES with DFA. https://yifan.lu/2019/02/22/attacking-hardware-aes-with-dfa/ (2019), [Online; accessed 9-Dec-2021]
33. Maurine, P.: Techniques for EM fault injection: Equipments and experimental results. In: Bertoni, G., Gierlichs, B. (eds.) 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, September 9, 2012. pp. 3–4. IEEE Computer Society (2012). https://doi.org/10.1109/FDTC.2012.21, https://doi.org/10.1109/FDTC.2012.21
34. Meriac, M.: Heart of darkness-exploring the uncharted backwaters of hid iclass (tm) security. In: 24th Chaos Communication Congress (2010)
35. Moradi, A., Schneider, T.: Improved side-channel analysis attacks on xilinx bitstream encryption of 5, 6, and 7 series. In: Standaert, F., Oswald, E. (eds.) Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9689, pp. 71–87. Springer (2016). https://doi.org/10.1007/978-3-319-43283-0_5, https://doi.org/10.1007/978-3-319-43283-0_5
36. Moustafa, M.: Emerald. https://github.com/reb311ion/emerald (2021)
37. Obermaier, J., Schink, M., Moczek, K.: One exploit to rule them all? on the security of drop-in replacement and counterfeit microcontrollers. In: Yarom, Y., Zennou, S. (eds.) 14th USENIX Workshop on Offensive Technologies, WOOT 2020, August 11, 2020. USENIX Association (2020), https://www.usenix.org/conference/woot20/presentation/obermaier
38. Obermaier, J., Tatschner, S.: Shedding too much light on a microcontroller's firmware protection. In: 11th USENIX Workshop on Offensive Technologies (WOOT 17). USENIX Association, Vancouver, BC (Aug 2017), https://www.usenix.org/conference/woot17/workshop-program/presentation/obermaier
39. O'Flynn, C.: Fault injection using crowbars on embedded systems. IACR Cryptol. ePrint Arch. p. 810 (2016), http://eprint.iacr.org/2016/810

40. O'Flynn, C.: Low-cost body biasing injection (BBI) attacks on WLCSP devices. In: Liardet, P., Mentens, N. (eds.) Smart Card Research and Advanced Applications - 19th International Conference, CARDIS 2020, Virtual Event, November 18-19, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12609, pp. 166–180. Springer (2020). https://doi.org/10.1007/978-3-030-68487-7_11, https://doi.org/10.1007/978-3-030-68487-7_11
41. O'Flynn, C., Chen, Z.D.: 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. pp. 243–260 (2014). https://doi.org/10.1007/978-3-319-10175-0_17, https://doi.org/10.1007/978-3-319-10175-0_17
42. O'Flynn, C., d'Eon Greg: I, for One, Welcome Our New Power Analysis Overlords - An Introduction to ChipWhisperer-Lint. Black Hat USA (2018)
43. Quisquater, J., Samyde, D.: Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In: Attali, I., Jensen, T.P. (eds.) Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2140, pp. 200–210. Springer (2001). https://doi.org/10.1007/3-540-45418-7_17, https://doi.org/10.1007/3-540-45418-7_17
44. Roche, T., Lomné, V., Mutschler, C., Imbert, L.: A Side Journey To Titan. In: Bailey, M., Greenstadt, R. (eds.) 30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021. pp. 231–248. USENIX Association (2021), https://www.usenix.org/conference/usenixsecurity21/presentation/roche
45. Ronen, E., Shamir, A., Weingarten, A., O'Flynn, C.: IoT Goes Nuclear: Creating a ZigBee Chain Reaction. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. pp. 195–212. IEEE Computer Society (2017). https://doi.org/10.1109/SP.2017.14, https://doi.org/10.1109/SP.2017.14
46. Roth, T.: SVD-Loader for Ghidra. https://github.com/leveldown-security/SVD-Loader-Ghidra (2019)
47. Roth, T., Nedospasov, D., Josh, D.: wallet.fail - hacking the most popular cryptocurrency hardware wallets. In: Thirty-Fifth Chaos Communications Congress. Berlin, Germany (2018)
48. Seri, B., Vishnepolsky, G., Zusman, D.: BLEEDINGBIT: The Hidden Attack Surface Within BLE Chips. https://info.armis.com/rs/645-PDC-047/images/Armis-BLEEDINGBIT-Technical-White-Paper-WP.pdf (2018), [Online; accessed 12-April-2021]
49. Shepherd, C., Markantonakis, K., van Heijningen, N., Aboulkassimi, D., Gaine, C., Heckmann, T., Naccache, D.: Physical fault injection and side-channel attacks on mobile devices: A comprehensive survey. CoRR **abs/2105.04454** (2021), https://arxiv.org/abs/2105.04454
50. Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2523, pp. 2–12. Springer (2002). https://doi.org/10.1007/3-540-36400-5_2, https://doi.org/10.1007/3-540-36400-5_2
51. Teuwen, P.: SideChannelMarvels - PhoenixAES. https://github.com/SideChannelMarvels/JeanGrey (2021)

52. Texas Instruments: Understanding security features for SimpleLink™ Bluetooth® low energy CC2640R2F MCUs. https://www.ti.com/lit/ml/swpb016a/swpb016a.pdf (2017), [Online; accessed 9-Dec-2021]
53. Texas Instruments: Secure Boot in SimpleLink™ CC13x2/CC26x2 Wireless MCUs. https://www.ti.com/lit/an/swra651/swra651.pdf (2019), [Online; accessed 9-Dec-2021]
54. Texas Instruments: CC13x0, CC26x0 SimpleLink™ Wireless MCU Technical Reference Manual. https://www.ti.com/lit/ug/swcu117i/swcu117i.pdf (2020), [Online; accessed 9-Dec-2021]
55. Texas Instruments: CC13xx/CC26xx Hardware Configuration and PCB Design Considerations. https://www.ti.com/lit/an/swra640e/swra640e.pdf (2020), [Online; accessed 9-Dec-2021]
56. Texas Instruments: Applications for the SimpleLink™ platform. https://www.ti.com/wireless-connectivity/applications.html (2021), [Online; accessed 9-Dec-2021]
57. Timon, B.: Non-profiled deep learning-based side-channel attacks with sensitivity analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(2), 107–131 (2019). https://doi.org/10.13154/tches.v2019.i2.107-131, https://doi.org/10.13154/tches.v2019.i2.107-131
58. Wouters, L., Gierlichs, B., Preneel, B.: My other car is your car: compromising the Tesla Model X keyless entry system. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 149–172 (2021). https://doi.org/10.46586/tches.v2021.i4.149-172, https://doi.org/10.46586/tches.v2021.i4.149-172
59. Wouters, L., den Herrewegen, J.V., Garcia, F.D., Oswald, D.F., Gierlichs, B., Preneel, B.: Dismantling DST80-based Immobiliser Systems. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(2), 99–127 (2020). https://doi.org/10.13154/tches.v2020.i2.99-127, https://doi.org/10.13154/tches.v2020.i2.99-127