IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 2, pp. 88–129. DOI:10.46586/tches.v2024.i2.88-129

Unlock the Door to my Secrets, but don't Forget to Glitch

A comprehensive analysis of flash erase suppression attacks

Marc Schink[1,2], Alexander Wagner[1,2], Felix Oberhansl[1], Stefan Köckeis[1], Emanuele Strieder[1,2], Sven Freud[3], and Dominik Klein[3]

[1] Fraunhofer Institute for Applied and Integrated Security (AISEC), Garching, Germany, firstname.lastname@aisec.fraunhofer.de
[2] Technical University of Munich (TUM), Munich, Germany, firstname.lastname@tum.de
[3] Bundesamt für Sicherheit in der Informationstechnik (BSI), Bonn, Germany, firstname.lastname@bsi.bund.de

Abstract. In this work, we look into an attack vector known as *flash erase suppression*. Many microcontrollers have a feature that allows the debug interface protection to be deactivated after wiping the entire flash memory. The flash erase suppression attack exploits this feature by glitching the mass erase, allowing unlimited access to the data stored in flash memory. This type of attack was presented in a confined context by Schink et al. at CHES 2021. In this paper, we investigate whether this generic attack vector poses a serious threat to real-world products. For this to be true, the success rate of the attack must be sufficiently high, as otherwise, device unique secrets might be erased. Further, the applicability to different devices, different glitching setups, cost, and limitations must be explored. We present the first in-depth analysis of this attack vector. Our study yields that realistic attacks on devices from multiple vendors are possible. As countermeasures can hardly be retrofitted with software, our findings should be considered by users when choosing microcontrollers for security-relevant products or for protection of intellectual property (IP), as well by hardware designers when creating next generation microcontrollers.

Keywords: embedded system security · fault-injection attack · microcontroller · firmware extraction · flash erase suppression attack

1 Introduction

Microcontrollers are the backbone of our modern and connected world and used in numerous applications such as robotics, medical devices, aerospace, and automotive. The internet of things (IoT) further increase the pervasiveness of microcontrollers in more areas within industrial and consumer products. Since microcontrollers are responsible for complex tasks they contain valuable intellectual property (IP) that could be of interest to competitors and are therefore worth protecting. Furthermore, microcontrollers can be found in security-critical applications such as hardware security tokens or crypto wallets, to name just two of many examples [Fed23]. As a consequence, they contain credentials and other cryptographic secrets which make them an attractive target for adversaries. For that reason, the security of microcontrollers and their respective assets is of utmost importance for today's applications. Due to the distribution of microcontrollers and the physical access attackers often have, hardware attacks are a serious threat. Moreover, countermeasures against hardware attacks are difficult to retrofit, and even if software-based countermeasures

Licensed under Creative Commons License CC-BY 4.0.
Received: 2023-10-15 Accepted: 2023-12-15 Published: 2024-03-12

can prevent some attacks, updates for products in the field are difficult to roll out in practice. There are different attack vectors, but targeting the underlying hardware is attractive since such attacks often work independent of the application a microcontroller is used for. One of these attack vectors is to bypass the debug interface protection. The debug interface provides access to the internals of a microcontroller, including the flash memory which is used as non-volatile storage for the firmware, cryptographic credentials, or other sensitive data. After production, this interface is locked, thus preventing access to the flash memory. Attacking the debug interface protection is an attractive target for adversaries since a successful attack usually enables access to all internals of a microcontroller.

1.1 Related Work

In the past, several hardware attacks that target the debug interface protection of microcontrollers have been published. These attacks range from non-physical attacks that exploit implementation flaws to physical attacks that use different fault injection methods to tamper with the debug interface protection. Physical attacks can be grouped in non-invasive, semi-invasive and invasive. In this work, we consider only non-invasive attacks that do not need any modification of the target microcontroller. Methods in this category are, for example, voltage glitching and electromagnetic fault injection (EMFI).

A non-physical attack that allows to bypass the debug interface protection was presented in [Bro15, OSM20]. The flash memory's access restriction is circumvented by performing read operations through load instructions executed in SRAM instead of directly through the debug interface. Another non-physical attack which exploits a vulnerability that leaks information about the flash memory content through the program counter (PC) was presented in [SO20]. This information leakage can be used to extract large parts of the internal flash memory. In both attacks from [Bro15, OSM20] and [SO20], the affected microcontrollers do not allow to fully deactivate the debug interface and thus there is no proper countermeasure against these attacks. In [OT17], a hardware implementation flaw rather than a conceptual weakness is exploited. The authors discovered a race condition in the implementation of the debug interface protection. For a short amount of time after the microcontroller is powered up, the debug interface is accessible and allows to read from the internal flash memory even though the protection is enabled. The authors are able to exploit the race condition to extract the entire flash memory within minutes. A special kind of debug interface protection feature is necessary to enable so-called *multi-party firmware development*. It allows to deploy firmware on a microcontroller such that it can be used by other developers without being able to read or copy the firmware. In [SO19], a *code recovery attack* is presented that circumvents this kind of protection on several devices from different manufacturers. By observing the state of the microcontroller while executing the protected code, the instructions can be recovered.

Non-invasive physical attacks to bypass the debug interface protection have been shown for some devices of the nRF52 and EFM32 microcontroller families [SWUH21, Lim20a, Lim20b, Lim21]. The restrictions on their debug interface can be temporarily disabled by applying a voltage glitch or injecting a pulse via EMFI during the power-up phase of the microcontrollers. The underlying shortcomings that lead to this effect are not fully understood and remain unknown. This is not unusual for physical attacks, especially when the underlying hardware implementation is a black-box. Nevertheless, in some cases the effects are fully understood. In [SWUH21, BFP19], for example, an attack that reactivates a debug interface that was previously deactivated completely, was presented. This attack exploits a design flaw, which makes the microcontroller's hardware downgrade the protection level, if an error during the transfer of configuration data from flash memory into an internal register is detected. By injecting a glitch at a specific point in time during the startup phase of the microcontroller, this protection downgrade can be triggered. It is still not fully understood how the induced fault propagates, but the user manuals specify

how the system reacts to an erroneous transfer.

The attacks listed so far target the debug interface's implementation itself, either by exploiting design flaws or by inducing malfunction through fault injections. In contrast, Skorobogatov [Sko05] was the first to present an attack that exploits an internal feature responsible for disabling the debug protection. This feature allows to unlock the debug interface at any time, for example for failure analysis, and thereby disable all access restrictions. However, in order to ensure the confidentiality of the data stored in flash memory, a mass erase is performed before unlocking the debug interface. During the unlock procedure, he targets the flash memory controller to suppress the erase rather than the debug protection itself. The attack was carried out with voltage glitching on an 8-bit microcontroller. About 15 years later, Schink et al. [SWUH21] present a similar attack, this time on a more recent 32-bit microcontroller and using EMFI. Although a long time has passed since this attack vector was first published, current microcontrollers still seem to be vulnerable. However, neither Skorobogatov nor Schink et al. provide an in-depth analysis and only discuss this attack vector in passing as part of their specific work. This leaves open important questions regarding its feasibility under real-world conditions and its limitations, which we address in this publication.

1.2 Contributions

In this work, we present the first in-depth analysis of the so-called *flash erase suppression* attack vector presented by Schink et al. [SWUH21]. We improve and extend this work with an assessment of the potential risks caused by this attack on real-world products by investigating three open research questions:

- **Generalizability** Investigation of the debug interface protection of different microcontrollers. This includes the STM32L422KxT microcontroller used in [SWUH21], but also other devices of the STM32 microcontroller family as well as three other manufacturers, namely, Geehy, GigaDevice, and Artery.

- **Vulnerability** Dedicated analysis of the devices that are targeted, including a side-channel analysis (SCA) of the flash erase operation and the investigation of die shots for the location of relevant components such as the flash memory. This analysis provides reasoning regarding the chosen parameters for our fault attacks and their results and is supposed to help researchers by describing the process behind carrying out hardware attacks on black-box devices.

- **Transferability** Detailed analysis of the attack vector in different fault injection setups. As the attack tries to suppress the flash memory's erasure procedure, a failed attack might lead to the loss of device unique secrets. For that reason, we evaluate the attack on several microcontrollers of the same type. This answers the question of the overall success rate for this attack and how well it can be *transferred* to other devices of the same type. Further, in certain scenarios, the adversary's time for mounting the attack is limited. Therefore, the threat posed by the attack depends highly on its reliability. To this end, we discuss a realistic adversary model and evaluate the attack for this.

Our investigation shows that the attack affects various microcontrollers from different manufacturers. Without claiming to have developed the best attack possible, we demonstrate success rates that make the attack vector a realistic threat for real-world products. We conclude our analysis with a discussion of the limitations of the integration of countermeasures and the implications of our attack.

2 Flash Erase Suppression

In this section, we outline a common way to implement a debug interface protection and, based on that, describe the idea of the *flash erase suppression* attack. Afterwards, we briefly introduce the necessary basics of flash memory in microcontrollers.

2.1 Debug Interface Protection

Modern microcontrollers have an integrated debug interface which is used to aid development, initial commissioning, and debugging of a system. This interface provides access to the internals of a microcontroller, including the flash memory which, on commercial off-the-shelf (COTS) devices is often used for all non-volatile data, namely firmware, cryptographic credentials, and other sensitive data. Due to the high level of access provided by this interface, it needs to be restrictable to ensure data integrity and confidentiality of products deployed in the field. While there are microcontrollers that only allow to permanently and irreversibly deactivate their debug interface, often more complex protection mechanisms are implemented to meet the requirements of today's applications and development environments. One scenario where such mechanisms come in handy is for example failure analysis, see below. Usually microcontrollers have multiple protection levels with different features implemented, but most of them can be represented by the transition graph depicted in Figure 1.

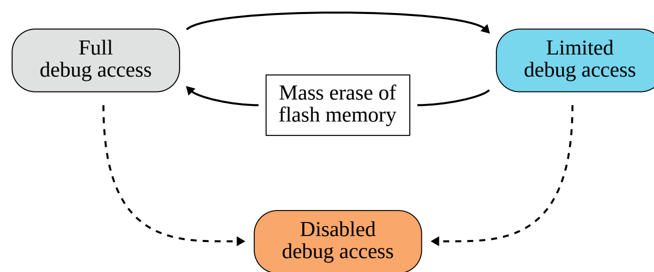


Figure 1: State transition diagram for the debug interface protection levels. The dashed line indicates that not every microcontroller allows to disable debug access.

In its initial state, a microcontroller allows full debug access. This stage is used for the development and testing of products. In order to prevent illegal reads from the internal flash memory once the product is deployed, the protection level can be increased to allow only limited access through the debug interface. The restrictions vary from microcontroller to microcontroller. For example, some devices restrict only access to the flash memory and allow to execute code from static random-access memory (SRAM). Others, however, only grant access to specific peripherals. Their common feature is that the debug interface can be *unlocked* to regain full debug access. During this unlock operation, however, the entire flash memory is erased in order to ensure the confidentiality of its data. Finally, some devices allow to further increase the protection such that no debug access is possible, this state is irreversible. The dashed lines in Figure 1 represents that not all devices support this feature.

Unlike other attacks listed in Section 1.1, the *flash erase suppression* attack does not tamper with the debug interface protection itself, but exploits its integrated unlocking feature. The goal is to disrupt the erase operation in a way that the microcontroller still unlocks the debug interface protection. Instead of targeting the debug interface protection itself, the flash memory controller and related components are targeted. If successful, the adversary has unrestricted access to all secrets stored in the flash memory. We will

demonstrate that this approach is a promising way to gain full and unlimited debug access on a microcontroller and has indisputable advantages over trying to directly targeting the debug interface protection. However, it also comes with two disadvantages. First, in case the attack is not successful, the microcontroller is either bricked due to the injected fault or the flash memory content is irretrievably lost. Second, the adversary needs some kind of interface to trigger the unlock operation. The former disadvantage is a question of the reliability and feasibility of the attack and will be answered through our experimental evaluation in Section 7. The latter disadvantage is more general and will be discussed in the following. There are different ways to interact with a microcontroller and perform an unlock operation, such as application-specific interfaces or integrated bootloaders. In this work, we consider only the microcontroller's integrated debug interface. Ideally, this interface should be deactivated after development and shielded from adversaries. Figure 1 already showed that there is a stage with limited debug access, which suffices to trigger the unlock procedure and gain unlimited access. In the following, we discuss why it is reasonable to assume that products in the field either use this stage or can be tricked into reverting to this stage:

- **Limited Debug Protection** There are microcontrollers that are not capable to fully deactivate the debug interface. In this case, debug access is always possible.
- **Failure Analysis** Even if microcontrollers can completely disable their debug interface, failure analysis sometimes requires limited debug access. For example, a disabled debug interface may prevent that chip manufacturers are not able to analyse defective parts [STM18]. Further, product manufacturers are not able to perform failure analysis of their devices through the debug interface. Some microcontrollers allow access to peripherals to verify the flash memory integrity through the debug interface [Art22b]. Other microcontrollers implement integrity verification even as dedicated feature of the debug interface [Sil17].
- **Multi-party Firmware Development** Firmware is sometimes not developed by a single vendor but consists of software libraries provided by one or more software companies. In order to protect the IP of the different companies, there are microcontrollers that support so-called *multi-party firmware development*. This feature allows to store firmware on a microcontroller that can be used by other developers without being able to read or copy the code. As this feature is intended to enable development on the microcontroller while protecting the contained IP, debug access is always possible. Physical attacks are not in the scope of this feature, nevertheless it is a reasonable attack vector.
- **Debug Interface Reactivation** There are known attacks that show how a deactivated debug access port can be reactivated. Since such attacks come with their own difficulties, they further complicate the flash erase suppression attack and may decrease the overall success rate. For example, the attack presented in [OT17] shows how ultraviolet (UV) light can be used to reactivate a debug access port for the STM32F1 microcontroller series. This attack is semi-invasive and may damage the microcontroller during the execution. In contrast, in [SWUH21, BFP19] two non-invasive glitching attacks are presented that show how the debug access port of different STM32 microcontrollers can be reactivated. For these attacks, the number of attempts is practically unlimited and the risk of damaging the device is low.

Three of the four listed points show that it is a reasonable to assume that devices allow limited debug access, to enable failure analysis or multi-party firmware development, or because there is no possibility to completely disable debug interfaces. Debug interface reactivation attacks demonstrate that, even if the debug interface is deactivated completely, it is not necessarily irreversible for a physical adversary.

2.2 Embedded Flash Memory

The dominant technology to store non-volatile data on microcontrollers is embedded flash memory (eFlash) [Hid17]. Beside the flash memory cells that store the actual data, the eFlash consists of additional components such as high-voltage circuits and control and management logic to ensure operation in spite of physical difficulties that arise during read, erase and program operations [Hid17]. There are multiple types of flash memory which all have different advantages and disadvantages for the use in microcontrollers. The one-transistor (1T)-NOR flash memory cell architecture is widely deployed due to its simplicity and high density [Hid17, Tor17, CGOZ99]. For example, STMicroelectronics uses this cell architecture for different microcontroller types, ranging from security to low power devices which have different requirements for their non-volatile memory [Hid17].

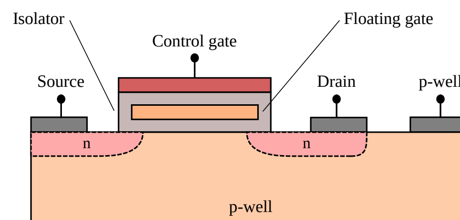


Figure 2: Schematic cross-section of a one-transistor floating gate flash memory cell [App07].

The advantage of simplicity and high density comes with the downside of a more complex erase operation, which will be discussed later. In Figure 2, the schematic of a 1T floating gate cell is depicted. The basic operating principle of a floating gate flash memory cell is that charges on the floating gate influence the threshold voltage V_{th} of the transistor and thereby the current flow between drain and source. In the following, we describe the different operations of a 1T-NOR flash memory cell from a high-level perspective, neglecting most of the physical problems that occur in practice.

Read Operation. In order to determine the state of a cell, a reference threshold voltage is applied to the control gate. If there are no charges on the floating gate, the cell conducts a current between drain and source. In this case, the cell is by definition *erased* and in a logical "1" state. Otherwise, the cell is *programmed* and in a logical "0" state [Ren19].

Program Operation. A flash memory cell is programmed by injecting charges inside its floating gate. By applying a voltage difference between source and drain an electron flow is caused. By additionally applying a high voltage on the control gate, the charges flowing between source and drain are pulled into the floating gate [Ren19].

Erase Operation. While it is possible to program individual cells, the erase operation can be performed only block-wise. The reason is that a high positive voltage must be applied to the p-well in order to attract the charges trapped inside the floating gate. For space reasons, multiple cells share a common p-well and thus can only be erased together [Ren19]. Since the amount of charge inside the floating gate of the individual cells can be different, an erase operation may lead to so-called *over-erased* cells. Such cells lead to incorrect results during read operations and must be avoided. For that reason, the eFlash control logic executes the following procedure to perform an erase operation [Ren19, Hid17]:

1. **Pre-program** All erased cells inside a block are programmed. This ensures that the amount of charges on all floating gates is approximately the same.

2. **Physical Erase** A large electrical field is applied between the common p-well and the control gates to attract charges trapped inside the floating gates. This process is repeated until all cells of a block are erased.

3. **Recovery** In case there are still over-erased cells, a so-called *soft-programming* operation is executed. This is an iterative process where a cell is programmed until it is not over-erased anymore.

Since the *pre-program* and *recovery* phases operate only on certain memory cells, their duration is proportional to the amount of erased data. The duration of these operations is in the range of a few microseconds. The *physical erase* phase is almost independent of the amount of data that is erased and takes usually a couple of milliseconds [Hid17].

A successful flash erase suppression attack must disturb at least phases (1) and (2). Otherwise, the data stored in the flash memory is either lost because all cells are programmed or erased afterwards. The third phase is not critical since only over-erased cells are recovered which should not affect the stored data.

3 Adversarial Model

In this section, we specify the adversarial model on which our investigation is based. We consider the following capabilities of an adversary:

- **Know-How** We assume an attacker with proficient technical know-how and software tooling. Standard tooling for firmware development for microcontrollers suffices.

- **Equipment and Effort** We consider equipment that does not exceed a couple thousand USD in costs, such as off-the-shelf logic analyzers and side-channel measurement or fault injection setups. Therefore, it is a fair investigation to analyse COTS microcontrollers. While these devices are neither marketed nor designed as dedicated high security products that must withstand sophisticated physical attacks, they are used in security-relevant products and store sensitive data such as IP or cryptographic secrets. Security-relevant products should withstand attacks according to our constrained adversarial model.

- **Physical Access and Time** The attacker has physical access to at least one victim device. Multiple devices to gain knowledge and to calibrate the attack setup can be acquired. After calibration, the available time for performing the actual attack depends on the scenario. The authors of [SWUH21] only consider scenarios, for which the time span is constrained to a few hours and no visible physical modifications are allowed. Our study also includes less constrained scenarios for which a device is permanently stolen.

- **Goal** The adversary tries to obtain cryptographic secrets and/or other sensitive data such as IP or personal records. If the flash erase is not suppressed, device unique data is lost. Hence, it is important to achieve the best possible success rate for the time available.

4 Selection of Microcontrollers

In [SWUH21], an STM32L422KxT microcontroller is used as target device. For comparability of our results, we re-analyze this device in our study. We additionally include other STM32 devices to assess if the attack vector generalizes within the device family. The selection of these devices takes different (security) features and flash memory configurations into account which will be described in more detail in Section 6.

In order to evaluate whether the flash erase suppression attack is a general threat for microcontrollers, we include devices from other manufacturers. Foremost, we select microcontrollers that support a debug unlock procedure that promises confidentiality of flash contents as described in Section 2. Further, we focus on general purpose devices which cover a wide range of applications. We include devices of the APM32 and GD32 microcontroller families from Geehy and GigaDevice, respectively. These microcontrollers are designed as drop-in replacements for STM32 microcontrollers and are completely or mostly pin and software compatible to their counterparts. From [OSM20], we know that these drop-in replacements are not copies but own developments. Finally, we include devices of the AT32 microcontroller family from Artery in our research. The feature set, including the debug interface protection, of these devices is similar to that of the STM32 family. However, these microcontrollers are neither pin nor software compatible.

Table 1: Microcontrollers selected for analysis.

Manufacturer	Device series	Target device
Artery	AT32F415	AT32F415CBT7
	AT32F421	AT32F421C8T7
Geehy	APM32F1	APM32F103C4T6
GigaDevice	GD32E103	GD32E103C8T6
STMicroelectronics	STM32L1	32L151CBT6
		STM32L162ZDT6
	STM32L4	STM32L422KBT6

All microcontrollers we selected for evaluation are listed in Table 1. We analyze COTS microcontrollers without dedicated protection against hardware attacks. This device class is widely deployed in numerous areas where security devices are not strictly enforced by regulations. Regardless of the use case — security-related or not — IP theft is a serious threat for enterprises. Firmware extraction does not require a 100 % success rate as an adversary has often access to multiple devices. It should be noted that, for some of the selected devices, other non-physical and physical attacks on the debug interface protection are already known (Section 1.1). While the flash erase suppression attack might not be necessarily the easiest attack for some of these devices, we still consider it worthwhile to include these devices into our analysis to study the attack's effectiveness and aid future developments of secure flash erasure features.

5 Experimental Setup

In this section, we describe the setup we use for device analysis and mounting the flash erase suppression attack. As laid out in Section 1, the success rate and transferability of the attack are critical to the threat it poses for real-world products. Therefore, the experimental setup and its components are of importance. Our setup consists of laboratory equipment according to Section 3, a software framework to orchestrate the measurements as well as target boards for the individual microcontrollers and firmware that is running on the microcontrollers.

Laboratory Setup. As basis for all experiments, we use the CW308 motherboard from NewAE Technology. For each of the selected microcontrollers, we use a dedicated target board that is mounted on the CW308 during the experiments. The target boards are designed such that they can be used for the side-channel based evaluation of the microcontroller as well as for the actual fault attacks later on. We use side-channel analysis (SCA)

as part of our device investigation in Section 6. For that we use a Langer RF-U 2,5-2 near field probe and the WavePro 404HD oscilloscope from Teledyne LeCroy.

The voltage glitching experiments are conducted with the CW1200 ChipWhisperer-Pro from NewAE Technology. Apart from that, the laboratory setup consists of a programmable power supply (PPS) and a debug probe. With the PPS, we automatically power cycle the target microcontroller after each fault attack and apply settings of the debug interface protection that require a power cycle. Further, the PPS is used to power the target microcontroller with different voltage levels during the voltage glitching attacks. To interface the different microcontrollers, we use an off-the-shelf debug probe.

For the experiments with electromagnetic fault injection (EMFI), we use the ChipSHOUTER from NewAE Technology. The laboratory setup consists of an additional delay generator and a positioning table. To place the injection coil automatically and reproducibly on the chip package, the CW308 motherboard is mounted on a 3-axis positioning table. We use the STEPCRAFT D-300, a CNC milling machine for hobbyists, as cheap and affordable positioning table. As delay generator, an FPGA development board is used. The delay generator is responsible to provide a trigger for the ChipSHOUTER to inject the fault at different times relative to a trigger signal provided by the microcontroller.

To carry out the flash erase suppression attack, an adversary requires only part of the listed equipment. At a minimum, only equipment to inject a fault is required. In the most expensive case, a ChipSHOUTER is required for EMFI which costs approximately 4500 USD. If the positioning or timing is unknown to the attacker, the profiling phase requires a setup including a positioning table or oscilloscope. The cost of the positioning table we use is around 1500 USD. To determine the timing of the flash erase operations, an oscilloscope of a lower performance class is sufficient. For example, a RIGOL DS1202Z-E, which costs about 500 USD. All in all, the attack equipment costs at most 6500 USD.

Software Framework. All equipment and the target microcontroller are controlled by a software framework running on a laboratory PC. As mentioned in Section 2, we only use the debug interface to access the microcontroller. In order to interface the microcontrollers, we use the open on-chip debugger (OpenOCD)[1]. A single test run of our evaluation software consists of the following steps. First of all, a power cycle of the target microcontroller is performed to ensure that the device is in a defined state. Afterwards, the flash memory is programmed with a test pattern and the debug interface protection is configured. Next, the software executes a firmware on the SRAM of the microcontroller to trigger the debug unlock operation and the fault attack. Once the trigger signal is detected by our evaluation tool, it checks the status of the microcontroller. Mainly, the tool checks whether the device is unlocked and the test pattern stored in flash memory were erased or not.

Attack Firmware. We craft a simple attack firmware that initializes the device, provides a trigger signal with a GPIO pin of the microcontroller and performs the debug unlock operation. On many microcontrollers, firmware execution from SRAM is possible even though only limited debug access is available. The reason for this is either that it does not directly pose an additional attack surface, or that this functionality is used to accelerate certain functions such as flash memory operations.

We chose to execute an attack firmware on the microcontroller as it provides a stable and deterministic trigger signal with respect to the debug unlock operation. With that we do not need side-channel or software-based triggers. Side-channel based triggers need more attack equipment and software-based triggers usually have more jitter due to the operating system (OS) and other confounding factors on the PC.

[1] https://openocd.org/

6 Device Analysis

In this section, we perform an in-depth analysis of the selected microcontrollers to ensure that we apply the flash erase suppression attack as effectively as possible and to understand the effects of the introduced faults. We start with describing the concrete implementation of the debug interface protection and other related security features of the respective microcontrollers. Based on that information, we perform a side-channel analysis (SCA) to understand how the debug unlock operation is implemented. This insight will help us in narrowing down the point in time for the fault injection attack to suppress the flash erase operation. The times in all the following diagrams refer to the time of the trigger signal of our attack firmware. To reason about what actually happens during the flash erase suppression attack, we take infrared (IR) pictures of the microcontroller dies. This information is helpful to determine whether there is a spatial relation between the location of the injected faults and the location of the eFlash. All information obtained during our device analysis is used subsequently to apply, optimize, and evaluate our attack.

6.1 Debug Interface Protection

The microcontrollers we examine in this work comprise a debug interface protection feature that can be configured to different levels. Their behaviour and transition follows the principle discussed in Section 2.1. All microcontrollers use a similar implementation, only naming convention or protection level encoding differs among the manufacturers. To avoid confusion, we will use the naming introduced in Section 2.1 in the rest of this work.

The protection level resides in a dedicated memory region in the flash memory used to store non-volatile configuration data. Since flash memory is used as storage, an erase operation is necessary before the protection level can be changed. How and when the configuration data is erased during a debug unlock operation will be discussed later.

Table 2: Flash memory features and chip package types of the selected microcontrollers.

Target microcontroller	Flash memory size	Mass erase support	Debug access deactivation	Package
AT32F415CBT7	128KiB	Yes	Yes	LQFP-48
AT32F421C8T7	64KiB	Yes	Yes	LQFP-48
APM32F103C4T6	128KiB	Yes	No	LQFP-48
GD32E103C8T6	64KiB	Yes	No	LQFP-48
STM32L151CBT6	128KiB	No	Yes	LQFP-48
STM32L162ZDT6	2 × 192KiB	No	Yes	LQFP-144
STM32L422KBT6	128KiB	Yes	Yes	LQFP-32

Except for the STM32L1 series, all microcontrollers have a standalone mass erase operation for the flash memory that can be triggered independently from the debug unlock process. We use this feature to draw a comparison between the side-channel emanation of a debug unlock procedure and a standalone flash mass erase. The results of this comparison and details on why it is beneficial for the analysis of the flash erase suppression attack are discussed later in Section 6.2 and Section 7.

All important flash memory features and the device package of the microcontrollers we analyze in this work are listed in Table 2. The device package is relevant for the attack evaluation with EMFI in Section 7.2.

Debug Unlock Operation. The different methods how the debug unlock process and the resulting mass erase are triggered are important for our analysis. An understanding

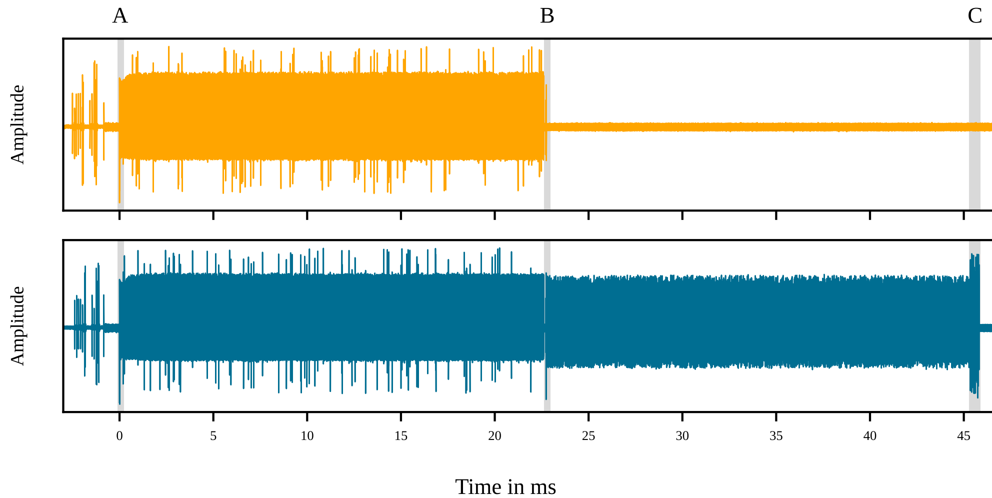


Figure 3: Electromagnetic (EM) trace of the STM32L422KBT6 microcontroller during a mass erase (top) and debug unlock (bottom) operation.

of these methods will be necessary for the subsequent SCA. For the STM32 devices we analyze in this work, the entire debug unlock operation, including a mass erase of the flash memory, is performed once the protection is disabled. In contrast, for the other microcontrollers, the configuration block of the flash memory that holds the protection level must be manually erased first. Afterwards, the protection level can be disabled which leads to the immediate erasure of the flash memory. Note that after erasing the configuration block, the debug interface access remains limited. This is due to the protection level encoding. All microcontrollers remain protected until they are power-cycled or a dedicated reload of the configuration is performed.

Multi-party Firmware Development. STMicroelectronics has a feature called proprietary code-read out protection (PCROP) which enables multi-party firmware development. This feature is implemented in many devices of the STM32 microcontroller family. Among the devices we examine is this work, it is supported by the STM32L422KBT6. This feature is of interest since its goal is protection of IP while the debug interface remains accessible (Section 2.1). PCROP can be configured such that the protected code is affected by the mass erase that is performed when the debug protection is disabled [STM18]. With a successful flash erase suppression attack, firmware that is protected via PCROP could be extracted. For that reason, we include this feature in our evaluation in Section 7. The AT32 family integrates a similar feature which is called security library (sLib) and is supported by both AT32F4 microcontrollers. In contrast to PCROP, this feature remains active even after the debug interface is unlocked [Art22a, Art22b]. The protected flash memory region must be deactivated explicitly by providing the password that was set during the initialization of the sLib feature. Once the password is entered, a mass erase is performed during the next system reset of the microcontroller. Since this behaviour does not fit into the process of how we evaluate the flash erase suppression attack, we leave the susceptibility of this feature to future work.

6.2 Side-Channel Analysis

In order to understand how the debug unlock operation is implemented we perform a side-channel analysis (SCA) on two different microcontrollers. First, we take a look at

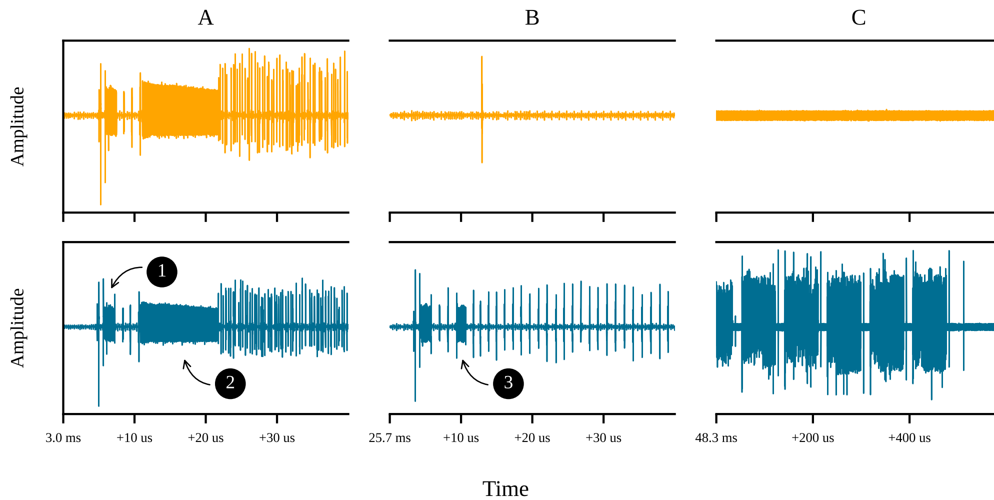


Figure 4: Detail view of the EM traces for the mass erase (top) and debug unlock (bottom) operation depicted in Figure 3.

the STM32L422KBT6 microcontroller. This device is well suited as example because it exhibits a clear EM signal during the debug unlock operation. The different phases of the flash mass erase operation are clearly visible. Afterwards, we take a look at the AT32F415CBT7 microcontroller to show a different debug unlock implementation. The insights we gain from these example devices can be applied to all other devices in this work without exception. The EM traces for all remaining microcontrollers can be found in Appendix A. All measurements are conducted with a supply voltage of 3.3 V and the near field probe placed on one of the supply pins for the digital power domain.

Debug Unlock with Automatic Erasure. The EM traces of a standalone mass erase and debug unlock operation for the STM32L422KBT6 microcontroller are depicted in Figure 3. Due to the comparably high voltages used by the eFlash (Section 2.2), patterns are directly recognizable and a single trace suffices. Since we perform only device initialisation before the trigger, we can ignore the EM signal before $t = 0$ ms. It is recognizable that the unlock operation is almost twice als long as the standalone mass erase operation. The start of both traces looks similar and contains nearly identical patterns. Apart from the duration, a major difference of the two traces is the short block at the end of the debug unlock operation. Also, there is a small gap where the debug unlock procedure's trace has a high activity at around $t = 23$ ms.

To inspect both operations in detail, we zoom into the EM signal directly after the trigger (A), at the small gap in the middle (B), and at the end of the debug unlock operation (C). The resulting plot is depicted in Figure 4. In section (A), the debug unlock operation looks similar to that of the standalone mass erase operation. Therefore, we conclude that the flash is erased at this time. We identify the first block (1) as a kind of initialization phase which always appears and is independent of the amount of data that is erased. Based on the working principle of eFlash in Section 2.2, we identify the second block (2) as the *pre-program* phase of the mass erase operation. It takes just a few microseconds and its length correlates with the amount of data stored in flash memory. To observe this correlation, we performed multiple debug unlock operations with different amount of data stored in flash memory. The subsequent block of high activity represents the *physical erase* phase. It takes several milliseconds and is independent of the amount of erased data. We additionally confirmed this assumption by comparing the EM

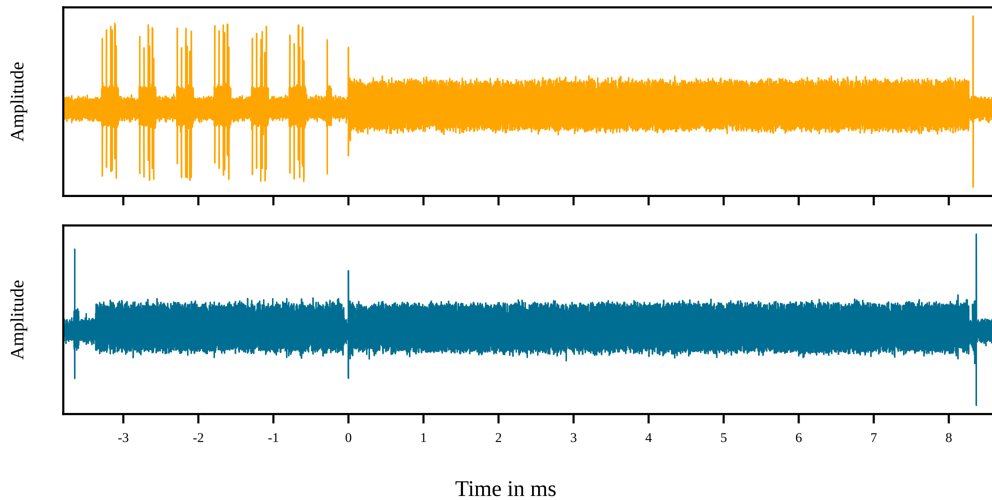


Figure 5: EM trace of the AT32F415CBT7 microcontroller during a mass erase (top) and debug unlock (bottom) operation.

traces of a page and a mass erase. In section (B), the standalone mass erase is finished afterwards, whereas the debug unlock procedure is only halfway done. We assume that this part of the EM signal represents the erase operation of the configuration block, which holds the protection level. The first reason for our assumption is that there must be an erase before the configuration block is programmed. Secondly, the pattern of the signal is very similar to that of the mass erase in section (A). Only the second block (3) is shorter, which fits the significantly smaller size of the configuration block. The block of high activity afterwards, which takes several milliseconds, also fits a subsequent *physical erase* phase. From [SWUH21], we know that, at the end of the debug unlock operation, the configuration block, which among others holds the protection level, is programmed. This is consistent with section (C) of Figure 4. Each block represents one of the five 32-bit configuration data words.

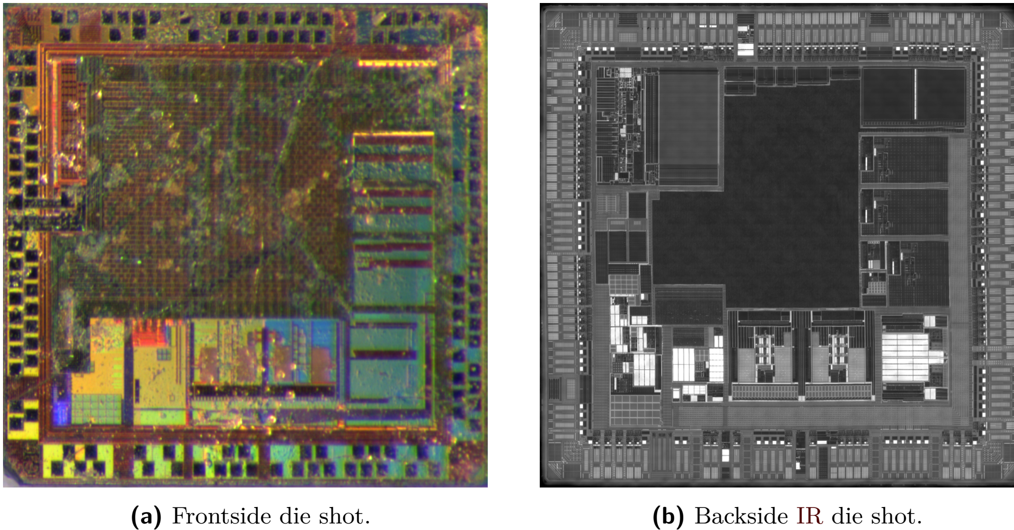
Debug Unlock with Manual Erasure. The previous analysis shows that the configuration block of the STM32 devices is automatically erased and programmed with the lowest protection level once the debug unlock operation is triggered. In contrast, all other microcontrollers require that the configuration block is erased manually before the debug unlock operation can be performed. In Figure 5, the EM emanation for the AT32F415CBT7 microcontroller during a standalone mass erase (top) and debug unlock (bottom) operation is depicted. Like in Figure 3, we can see a similar pattern for the standalone mass erase and the debug unlock operation *after* the trigger. Due to the similarity, we identify this part as the mass erase of the debug unlock operation. There is also a small block of activity at the end of the debug unlock operation that we identify as programming of the protection level. The different phases of the mass erase and programming operation are similar to the STM32L422KBT6 microcontroller but not shown here for the sake of brevity. The EM trace *before* the trigger looks completely different for both traces. This is due to the manual erase of the configuration block that needs to be performed only for the debug unlock operation. The bottom plot shows the erase of the configuration block while the leading six blocks in the top plot stem from the communication between the debug probe and the microcontroller.

Summary. We identified the individual parts of the debug unlock operation for all microcontrollers. Based on the pattern of the EM signal, we assume that the standalone mass erase operation and the mass erase during a debug unlock operation are implemented similarly or are even the very same operation. Hence, if we are able to suppress the standalone mass erase operation we may also be able to suppress the flash erase during the unlock operation. The standalone mass erase operation is beneficial for the evaluation of the flash erase suppression attack in Section 7. The analysis not only makes the individual parts of the debug unlock operation visible but also the details within the mass erase operation. We identified the two phases that are important for the flash erase suppression attack: *pre-program* and *physical erase*. With that we narrowed down the temporal search space for the attack from several milliseconds to a few microseconds.

6.3 Optical Inspection

After the analysis of the EM emanation during the debug unlock operation, we take a look inside the chip package of the microcontrollers. For that, we open them from the backside and acquire IR pictures of their dies with the WIDY SWIR 640V-S camera from New Imaging Technologies. Later, we will use the gathered information to determine whether there is a spatial relation between the location of the injected faults and the location of the flash memory. The die shots are rotated such that they are aligned with the heatmaps presented in Section 7.2.

From [OSM20] we know that the GD32F1 microcontroller series has a dedicated flash memory die inside the chip package. For that reason, we remove the entire chip package of the GD32E103C8T6 to expose all the dies it contains. In contrast to the GD32F1 series, the chip package of the GD32E1 series contains no dedicated flash memory chip. The front- and backside pictures of the microcontroller die are depicted in Figure 6. Based on the backside die shot (Figure 6b) we identify the flash memory in the top left corner of the chip. The regular structure of the flash memory cells right next to the control logic and analog components such as high-voltage circuitry is clearly visible.



(a) Frontside die shot. **(b)** Backside IR die shot.

Figure 6: Die shots of a GD32E103C8T6 microcontroller.

The IR backside picture of the STM32L422KBT6 is shown in Figure 7a. On the bottom left corner we can see the flash memory. It is recognizable from its regular structure and distinguishable from SRAM through its more complex analog and high-voltage circuitry

right next to it. For both devices of the STM32L1 series we are not able to acquire IR backside pictures. We assume that an increased doping level of these devices impedes the transparency in the near-infrared (NIR) range at around 1300 nm wavelength which we used for illumination.

The IR die shot of the APM32F103C4T6 microcontroller is depicted in Figure 7b. On the left side we can see two white, horizontally mirrored and seemingly identical blocks. Based on their size and after all other relevant blocks could be excluded, we identify these blocks as the flash memory.

The backside die shots of the AT32F4 microcontrollers are shown in Figure 7c and Figure 7d. For the AT32F415CBT7, we identify the two white and horizontally mirrored blocks as the flash memory. For the AT32F421C8T7, we assume the white block on the top right corner to be the flash memory. The identification is based on an exclusion procedure of all other relevant blocks and the fact that we see only a single block. This fits with the flash memory size of the AT32F421C8T7 which is only half the size of the AT32F415CBT7. Note that the die depicted in Figure 7d is placed rotated in the chip package.

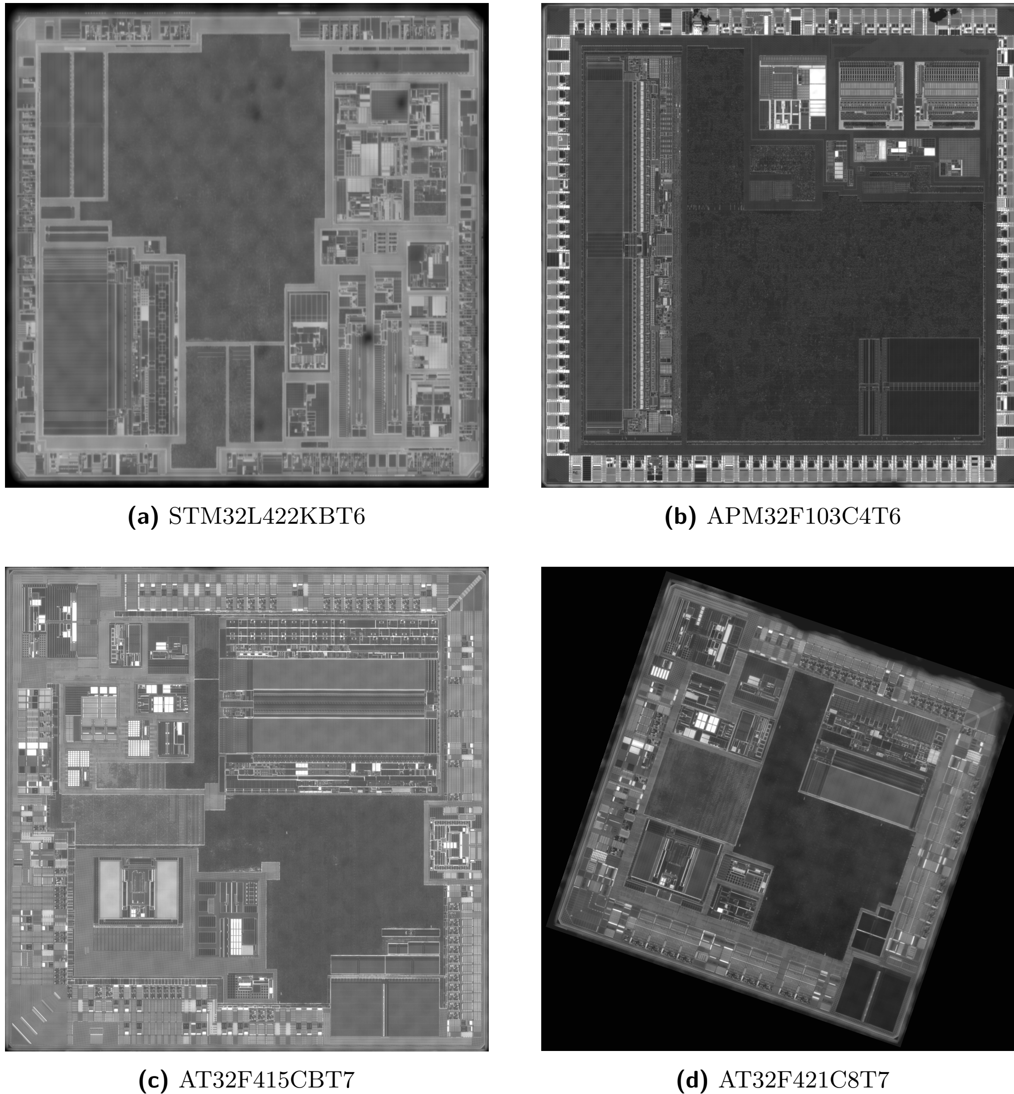
7 Attack Evaluation

We use the information gathered in Section 6 to asses whether the mass erase during a debug unlock operation can be suppressed with voltage glitching and EMFI. Further, we try to identify the root cause of the flash erase suppression attack.

Since the parameter space for both fault injection (FI) techniques is quite large, we use a two-step evaluation to speed up experiments. First, we fill only a single page of the flash memory with a test pattern to verify whether the erase suppression was successful or not. After this step, we have identified glitching parameters, such as the injection time, that lead to a successful attack. To achieve a high test rate, we run only a single experiment for each possible set of parameters. Only in the experiments where the suppression of erasing a single page is successful, we check again by filling the entire flash memory with the test pattern. This approach drastically reduces the time needed per experiment. When available, we also use the standalone mass erase operation to increase the test rate as the complete debug unlock procedure takes about twice the time. For the experiments with EMFI, we also limit the experiments to the susceptible area once identified to further speed up the experiments.

Attack Methodology. From the adversarial model introduced in Section 3, it is obvious that transferability and success rate are extremely important for this attack and therefore crucial parts of the investigation in this section. An attack that tries to extract device unique secrets is only reasonable if the success rate is 100 %, as the data is irreversibly erased if the attempt fails. For the extraction of IP, lower success rates are acceptable, however the economic incentive decreases with the success rate. Our methodology is as follows: for a single device, we performed a parameter search to achieve the highest possible success rate. To determine the parameter transferability, we performed independent measurements on four devices for each analyzed product. We do not run any calibration for these devices prior to any experiments. This allows to evaluate the impact of intrinsic and external variations, such as displacement or manufacturing deviations. Whenever we experienced singular behaviours of individual chips that we could not reproduce for others, we disclose this and describe it in detail. We count such results as failed attacks. The results represent the expected success rate for an attack on an unknown device. This approach is similar to side-channel template attacks as devices are used for "calibration" which differ from the victim device.

In the following, we present the attack methodology for both fault injection techniques using a single device only, namely the STM32L422KBT6 microcontroller which was used



(a) STM32L422KBT6 **(b)** APM32F103C4T6

(c) AT32F415CBT7 **(d)** AT32F421C8T7

Figure 7: Backside IR die shots of the selected microcontrollers.

in [SWUH21]. For the sake of clarity, we only present the results for all other devices where there are device-specific findings.

7.1 Voltage Glitching

For voltage glitching to be effective, usually the buffer capacitors must be removed, or the glitch must be inserted directly into the digital domain of the microcontroller. This is a common preparation step when the CPU of a microcontroller is targeted in order to skip instructions or alter the control flow another way [Lim20c, Lim19, BFP19]. Since flash erase suppression attacks have not been investigated so far, we analyse all microcontrollers with and without the buffer capacitors. We also test the attack with different supply voltages of the target microcontroller. Further, we vary the number of injected pulses and the pulse width of each glitch. The pulse width is a parameter of the ChipWhisperer and expressed as percentage of one clock period of an internal clock source. We also experiment with different injection delays within the time ranges we identified in Section 6. If not

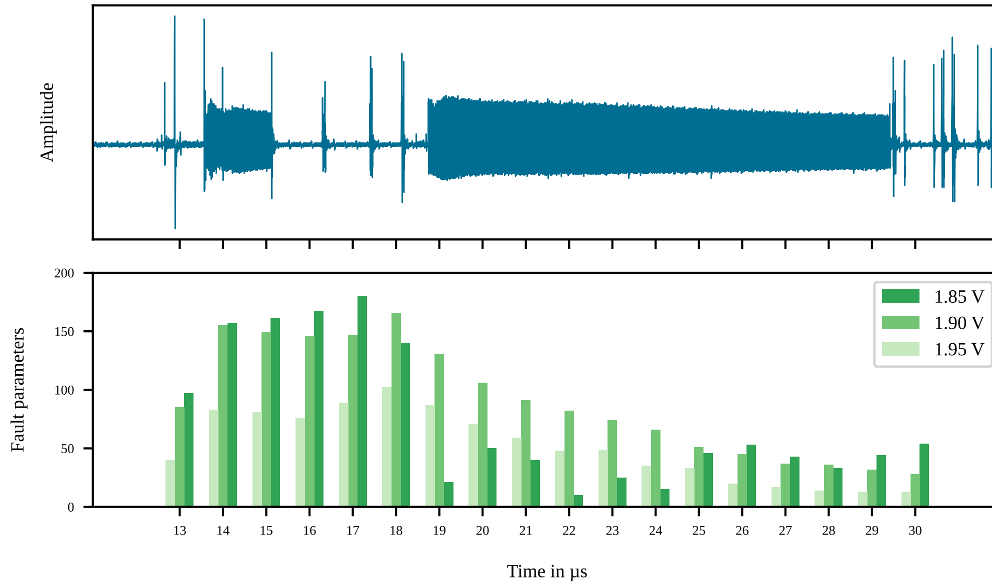


Figure 8: EM trace of the STM32L422KBT6 microcontroller during the debug unlock operation (top) and the number of effective faults parameters (bottom) at different injection times and supply voltages.

stated otherwise, we use the default settings of the ChipWhisperer for all experiments.

STM32L4 Series. According to [SWUH21], the STM32L422KxT microcontroller is not susceptible to voltage glitches. However, the authors state that they use the very same setup as for the EMFI attack with regard to the target supply voltage and the circuitry of the chip. They evaluate the attack in a scenario where physical modifications such as the removal of capacitors are not possible due to time restrictions and the evidence of tampering. For our setup, we can confirm that this microcontroller is not susceptible to voltage glitches when the buffer capacitors of the digital power domain are in place. Once both buffer capacitors are removed, we experience effective faults that suppress the flash erase operation.

Table 3: Glitching parameters and success rate for the STM32L422KBT6 microcontroller.

Supply voltage	Pulses	Pulse width	Injection time	Success rate
1.9 V	18	19 %	$13\mu S$ to $20\mu S$	100 %
	20	18 %	$13\mu S$ to $27\mu S$	

In order to find the best parameters for our setup we sweep the supply voltage of the microcontroller. The STM32L422KBT6 microcontroller has a minimum and maximum supply voltage of 1.71 V and 3.6 V [STM18]. We start our experiments with a minimum voltage of 1.8 V as the device does not work reliably with a lower supply voltage. Note that the minimum supply voltage is specified for certain conditions and with the recommended buffer capacitors in place.

As mentioned above, we start our analysis by only considering a single page of the flash memory. In Figure 8, the number of effective fault parameters at different injection times

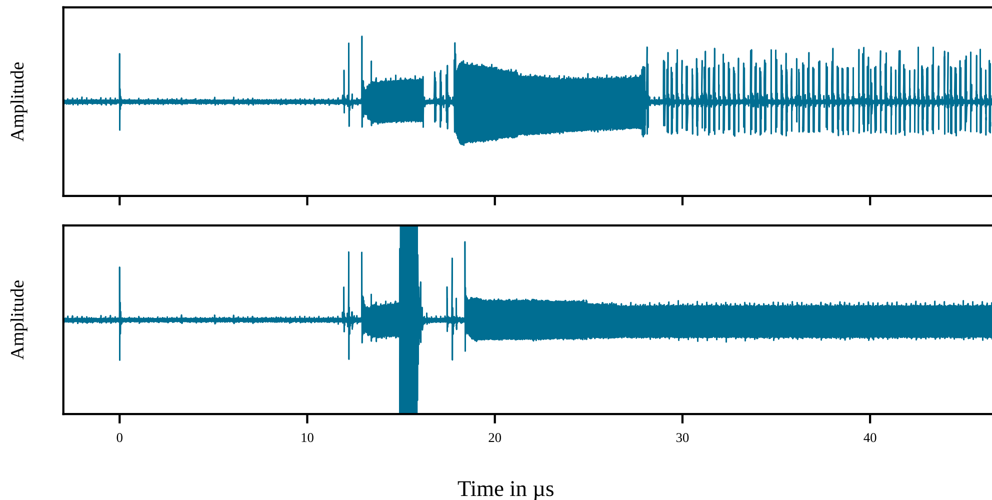


Figure 9: EM trace of the STM32L422KBT6 microcontroller during a debug unlock operation without (top) and with (bottom) injected voltage glitch.

and supply voltages is depicted. The plot shows the three supply voltages with the most effective fault parameters we found. As a time reference, the top plot shows the EM signal of the microcontroller during a debug unlock operation. The results affirm our analysis of Section 6 that the second block is related to the *pre-program* phase since the number of effective faults decreases for later injection times. We also experience an increasing number of defects the later a fault is injected. This also fits with the assumption that after the *pre-program* phase, all data is already lost.

We confirmed the most promising parameters by evaluating that flash contents are preserved across all pages. Further, we chose two of these parameters (Table 3) to evaluate whether the flash erase suppression also works on other STM32L422KBT6 devices. Note that there are many other parameter sets that work, we just list two as an example. The success rate for both parameter sets is 100 % for 1000 tries on all four devices. The time period in which the injection must take place is not critical for either sets and ranges from $7\mu S$ to $14\mu S$. We also evaluated whether the PCROP is affected by the flash erase suppression attack and can confirm its vulnerability. After a successful attack, the flash memory region that previously was secured with PCROP can be read out.

In order to reason about the effect the injected fault generates in the microcontroller, we acquire the EM signal of the microcontroller during the debug unlock operation while injecting a glitch. In Figure 9, the EM trace of the part of the debug unlock operation we identified as *pre-program* and beginning of the *physical erase* phase is depicted. The high activity which exceeds the signal range at around $t = 15\mu S$ arises due to the injected voltage glitch. At the same time, the start of the *pre-program* phase can be observed in the trace without injected glitch. Similar activity can be observed in the trace with injected glitch. The amplitude is smaller and it lasts longer than a normal *pre-program* operation and extends into the area that we identified as the *physical erase* phase. In Figure 10, the EM trace of the entire debug unlock operation is depicted. The faulted and non-faulted operation take exactly the same time. The first part, which corresponds to the mass erase of the flash memory, shows a lower amplitude but lasts exactly the same time. The second part, which starts at about $t = 23$ ms, relates to the erase and subsequent programming of the configuration block to deactivate the debug protection. The traces in the second part look similar which makes sense, considering that in both cases the debug protection is unlocked. Based on this observation, we conclude that the injected voltage glitch does

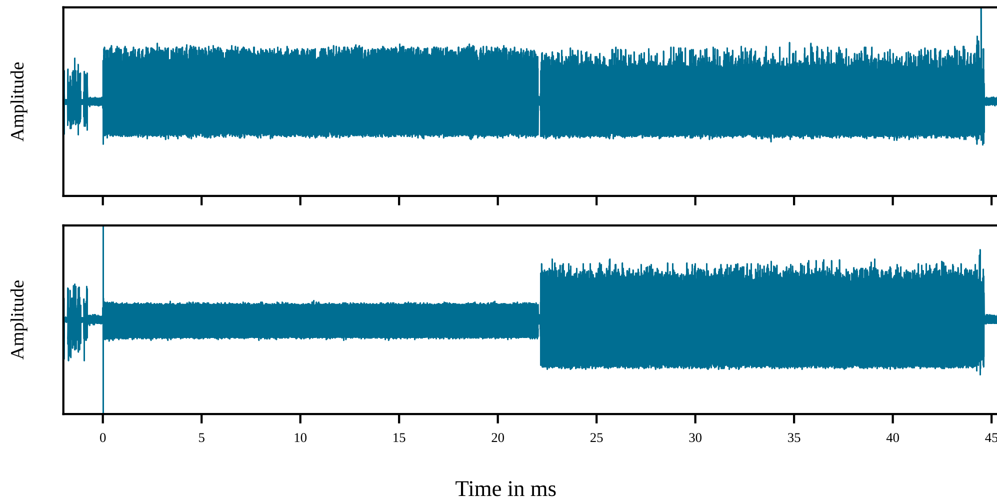


Figure 10: EM trace of the STM32L422KBT6 microcontroller during a debug unlock operation without (top) and with (bottom) injected voltage glitch.

not affect the internal state machine or control logic of the eFlash but rather perturbs the high-voltage circuitry. We base our conclusion on the fact that the EM activity during the mass erase is comparatively low which would fit a malfunction of the high-voltage circuitry. Further, since the duration is *exactly* the same, we come to the conclusion that the eFlash performs the actual erase operation without being effective.

STM32L1 Series. For the STM32L151CBT6 and STM32L162ZDT6 microcontroller, we found no working parameter for our setup to successfully suppress the flash erase operation. We tested both chips with all capacitors in place and nonetheless experience multiple defective devices. Since the STM32L1 series does not support a standalone mass erase operation, all experiments are performed using the debug unlock operation which, according to our experience, leads more often to defective devices during the parameter search. Since devices without capacitors are even more susceptible to defects we have not carried out the experiments without capacitors.

APM32F1 Series. The APM32F103C4T6 microcontroller is very sensitive to voltage glitches and even the parameter search with the standalone mass erase leads to multiple defective devices. We found no working parameter set for this microcontroller. As with the devices of the STM32L1 series, all capacitors were in place during our experiments.

AT32F415 Series. For the AT32F415 microcontroller series, we found working parameters during the parameter search with the standalone mass erase operation. We verified these parameters afterwards with the debug unlock operation for the actual attack. We performed all experiments without capacitors, because, from our experience, we cannot expect a more favorable behavior with capacitors. The numbers of effective fault parameters at different injection times during the mass erase operation are depicted in Figure 23 in Appendix B.

The highest success rate we achieve with our setup is 3.7 % for 1000 tries. Table 4 lists the parameters we use for this fault injection attack. Due to the low success rate, we did not examine this parameter set on multiple microcontrollers. For successful attacks, we observe the same behaviour during the attack as for the STM32L422KBT6. The debug unlock operation takes the same amount of time but the mass erase process is not effective.

Table 4: Glitching parameters and success rate for the AT32F415CBT7 microcontroller.

Supply voltage	Pulses	Pulse width	Injection time	Success rate
2.6 V	20	30 %	$7\mu S$	3.7 %

AT32F421 Series. For the AT32F421C8T7 microcontroller we found no working parameter but also experienced no device defects throughout our experiments. As for the AT32F415 series, we performed all measurements without capacitors installed.

GD32E103 Series. For the GD32E103C8T6 microcontroller, we also found no working parameter set to successfully suppress the flash erase operation. We tested the device without capacitors installed. During our experiments we experienced no defective devices.

Summary. The selected microcontrollers are very sensitive to voltage glitches during flash operations as we experience many defective devices during the parameter search. In most cases, the defective devices work electrically, but the debug interface is no longer accessible. We assume that either the injected faults corrupt parts of the flash configuration or physically damage the debug port. This analysis is outside the scope of this paper, so we did not investigate it further.

We have only found working parameters for a single device that allow an attack according to the adversarial model (Section 3), namely the STM32L422KBT6. For this microcontroller we achieve a success rate of 100 %. This allows an adversary to extract IP and even device unique secrets from the flash memory of such a microcontroller.

7.2 Electromagnetic Fault Injection

The advantage of EMFI in contrast to voltage glitching is that an injected glitch may not affect the entire device but only certain areas. This advantage comes at the cost of an increased parameter space due to the spatial location of the injection coil. For our experiments, we place the coil as close as possible to the package while still being able to move it with the positioning table. In order to further reduce the search space, we use a fix pulse width of 80 ns and use only the ChipSHOUTER's 4 mm coil with clockwise winding for our experiments. We chose this configuration as it has proven to be effective in [SWUH21]. Also, we use a fix supply voltage of 3.3 V for the microcontrollers during all experiments. If not stated otherwise, we perform the measurements with all capacitors in place. This leaves the injection delay and voltage as well as the coil's position as major parameters. For the spatial exploration of the chip package, we use a grid size of 0.5 mm unless otherwise specified. We consider this as a reasonable resolution which allows manual positioning of the probe or positioning using a stencil.

The final results of our experiments are heatmaps that show the success rate of the flash erase suppression in relation to the chip's package. For every position on the heatmap we perform 32 test runs unless otherwise specified. The black circle on the heatmaps indicates the chip orientation. We use a discrete color map in order to make even small changes visible and to ensure that success rates are not overestimated. The heatmaps that we present in the following are conservative in the sense that we round down the success rate in steps of 10 %. Non-evaluated coordinates that did not yield successful faults during the first evaluation step are colored grey.

STM32L4 Series. In Figure 11, the EM signal of the debug unlock operation (top) and the corresponding number of effective fault positions (bottom) is depicted. For this

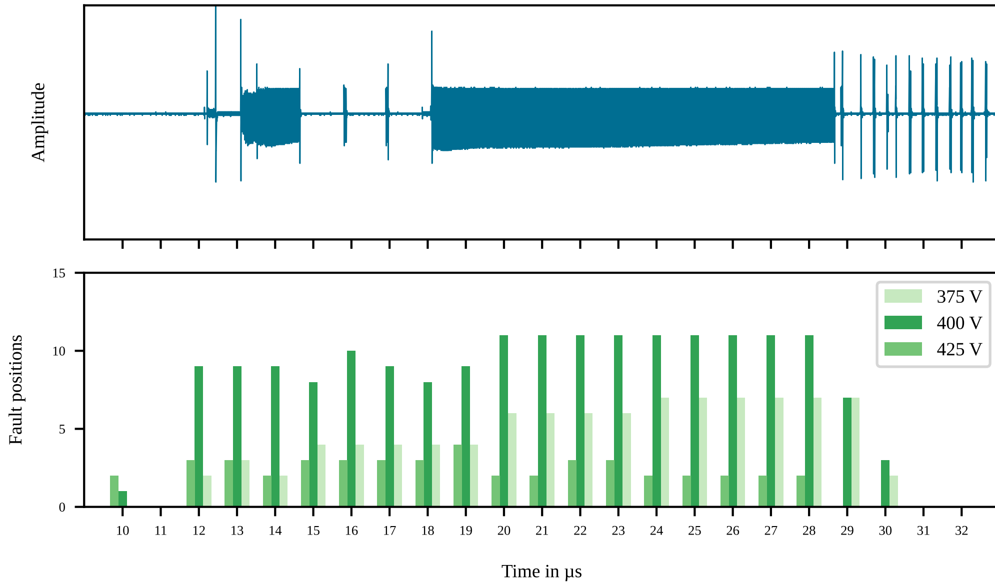
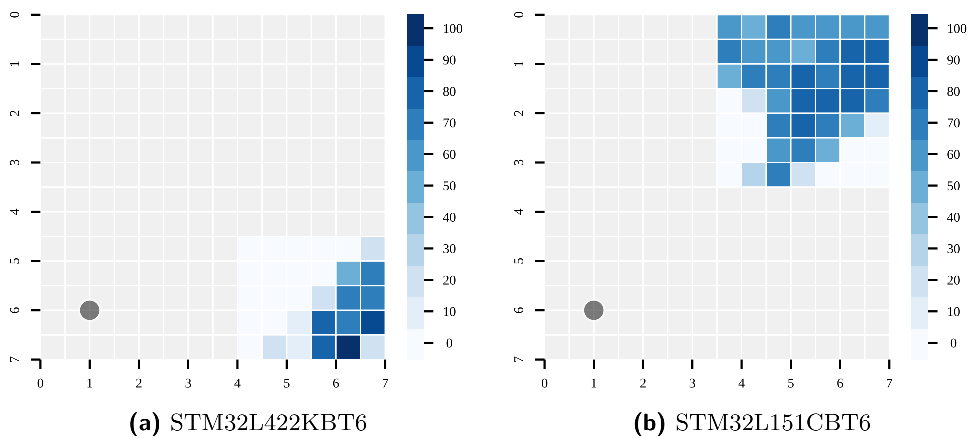


Figure 11: EM trace of the STM32L422KBT6 microcontroller during the debug unlock operation (top) and the number of effective faults positions (bottom) at different injection times and coil voltages.

parameter exploration, we used the first step of experimental evaluation described above. The plot shows the three coil voltages with the highest number of fault locations on the chip package. Similar to voltage glitching, the time span for successful fault injection extends over the two blocks we identified as part of the erase operation. With a coil voltage of 400 V we achieve the highest number of fault positions. Especially at the beginning of the second block, starting at around $20\mu S$, the number of fault positions is increasing. Nevertheless, we chose a coil voltage of 375 V and an injection delay of $18\mu S$ for further analysis. Using this coil voltage, we experience a better transferability, i.e. a higher average success rate on different devices. With a voltage of 400 V, we see more (non-permanent) failures on some devices leading to a lower overall success rate.

For all coil voltages and delays we see only effects on the bottom right corner of the chip package. For that reason, we reduce the area to this part of the chip for further analysis. The superposition of the results from all devices is depicted in Figure 12a. The heatmaps for the individual devices are depicted in Figure 30. There is an area of 1 mm^2 in the bottom right corner with a success rate between 70 % and 100 %. Our investigation yields a different susceptible area on the chip package in comparison to the original publication [SWUH21]. Most likely, the distance between chip package and injection coil is the reason for the different location. In both cases, the susceptible area does not match with the location of the flash memory on the chip. We assume that the injected EM pulses affect the power supply of the chip or the high-voltage circuitry of the eFlash which results in a suppression of the flash erase operation. As for the voltage glitching analysis, we can confirm that firmware protected by PCROP is vulnerable as well. A successful attack allows an adversary to read out flash memory that previously was secured with PCROP. For successful erase suppression attacks, the devices exhibit the same behaviour as for voltage glitching. The debug unlock operation still includes the mass erase operation but is not effective (Section 7.1).

STM32L1 Series. The results of the parameter exploration for the STM32L151CBT6 are depicted in Figure 24 in Appendix C. The time span for successful fault injection extends over the two blocks we identified as part of the erase operation. In contrast to the STM32L422KBT6 microcontroller, there is no such a sharp drop in the number of effective faults after the second block. This does not fit with the analysis in Section 6.2, we assume that the implementation has not such a strict separation between *pre-program* and *physical erase* phase and both are combined. Another reason could be that the EM signal does not allow a proper separation between both phases. With a coil voltage of 300 V we achieve the highest number of fault positions. Especially at the end of the second block, starting at $68\mu S$, the number of fault positions is increasing. We chose a coil voltage of 300 V and an injection delay of $73\mu S$ for further analysis. The decision is based on the number of faults but just as important is that for these parameters, there is a contiguous area that is susceptible, as the following results will show.

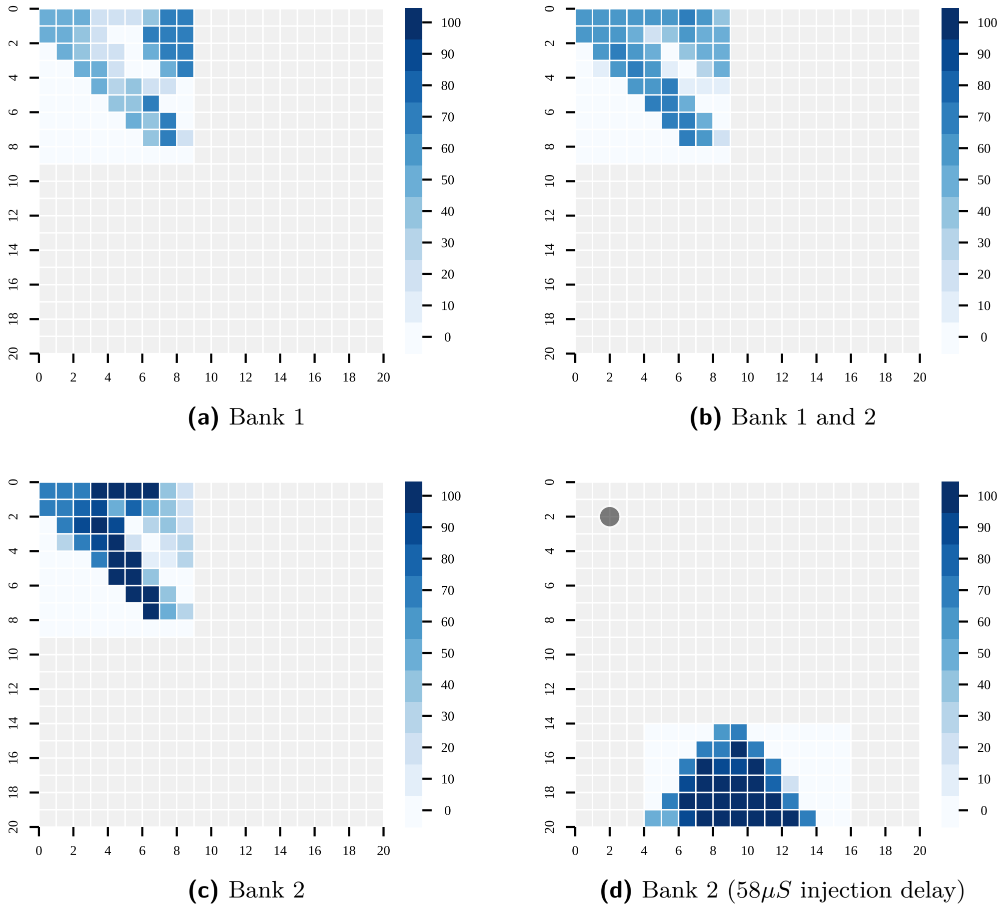


(a) STM32L422KBT6 **(b)** STM32L151CBT6

Figure 12: Heatmaps showing the overall success rate of the flash erase suppression attack from four (a) and eight (b) devices for two STM32 microcontrollers.

For a single device we made the observation that the suppression is only partially successful. Every second block of 256B of the flash memory is erased while the other half remains preserved. This means that at least 64KiB of this device can successfully be recovered. The heatmap for this device is depicted in Figure 31a. On another device we experience less susceptibility in the upper right corner of the package and observe more failures of the executed firmware. Even though the execution of our attack firmware is disrupted by the injected fault, the attack was successful and the flash memory remains intact. The heatmap of this device is depicted in Figure 31f. Since we did not experience similar behaviour on other devices, we repeat the experiment on a second, identical target board. As the second device does not exhibit such a behaviour, we assume that this behaviour is device intrinsic or stems from small deviations caused by hand soldering the devices. In total, we ran the complete attack evaluation on eight devices. Figure 12b shows a heatmap where the success rate for all eight devices is combined into a superposition. The resulting success rates are calculated conservatively in the sense that we consider the device where only half of the flash memory remains intact as completely failed. The eight individual heatmaps per device are depicted in Figure 31 in Appendix D. Nevertheless, with our setup we achieve a success rate of 70% to 80% in the top right corner.

The STM32L162ZDT6 microcontroller differs from the others in two respects. First, it comes in an LQFP-144 package and is eight times larger in terms of area. Second, this microcontroller has two instead of a single flash memory bank. To keep the evaluation time within reasonable limits we increase the grid size for our measurements to 1 mm.



(a) Bank 1 **(b)** Bank 1 and 2

(c) Bank 2 **(d)** Bank 2 ($58\mu S$ injection delay)

Figure 13: Heatmaps showing the overall success rate of the flash erase suppression attack from four individual STM32L162ZDT6 devices.

We perform the parameter exploration for both flash banks separately. The results are shown in Figure 25 in Appendix C. The parameter exploration shows that the number of fault positions decrease and increase over time, for the first and second bank, respectively. The reason may be that the banks are erased consecutively. One interesting observation on this device is that the high number of effective fault positions for bank 1 are $15\mu S$ to $25\mu S$ *before* the large block of activity in the EM signal appears that we identified as beginning of the flash erase operation. The reason could be that on this device it is not the erase operation that is directly suppressed but already the preparation phase. For further analysis, we chose a coil voltage of 200 V because we experience a more stable success rate across multiple devices in comparison to higher voltages. Since we want to use the same injection parameters for both flash banks, we conducted the parameter exploration for the second bank only with a coil voltage of 200 V. We chose an injection delay of $34\mu S$ for further analysis. We additionally performed a measurement for the second bank with an injection delay of $58\mu S$ because we experienced a large and contiguous area at the bottom of the chip package. There are multiple vulnerable regions on the chip. Since they depend very much on the injection time and flash bank, we chose three areas in order to keep the measurement time within a reasonable range.

In Figure 13, the superposition of four devices for both flash banks and at different injection delays are depicted. The individual heatmaps are depicted in Figure 32 in

Appendix D. For bank 1 we achieve a maximum success rate of 70 % in the top left corner. This is because on a single device we experience no effect at all. In contrast, bank 2 has multiple regions on the chip package where a success rate of 100 % can be achieved. Especially at the bottom of the chip package, we experience a large and consecutive area of $3 \times 4mm^2$ that is susceptible on all four devices. In this area, the injection coil can be placed by hand and without further aids. The highest overall success rate for both flash banks can be achieved in the top left corner of the chip package with a success rate of up to 60 %. Note that the heatmap with both banks is a superposition of bank 1 and 2 and not measured separately. However, we performed spot checks to verify that both flash banks remain intact when injecting faults at the resulting positions on the chip package. For our setup, we experience that bank 2 is more susceptible and stable than bank 1. Further experiments are needed to find out the root cause for this behaviour.

As we were unable to acquire die shots for STM32L1 chips (Section 6.3), we cannot draw any further conclusions about the fault positions.

APM32F1 Series. For the APM32F103C4T6 microcontroller we experienced an increased sensitivity to higher coil voltages compared to other devices. On one device we made the observation that it is no longer susceptible to the flash erase suppression attacks once we injected a fault with a coil voltage above 300 V. The general functionality of the device remains intact. We limit the coil voltage for our experiments on this device series to 300 V. With this limitation, no more defective devices occurred and we did not investigate this behavior further. The plot for the parameter exploration is depicted in Figure 26 in

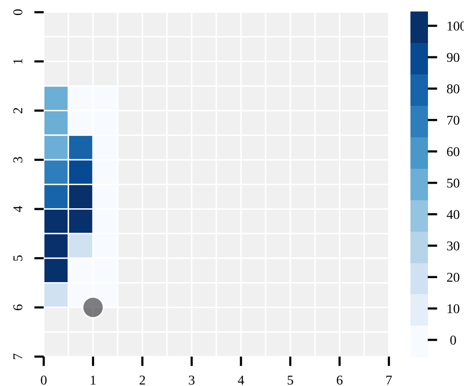


Figure 14: Heatmaps showing the overall success rate of the flash erase suppression attack from four devices for the APM32F103C4T6 microcontroller.

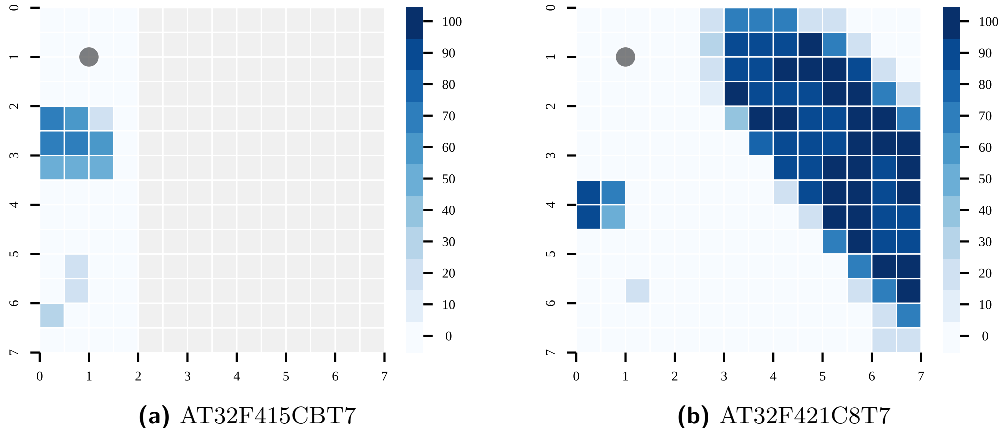
Appendix C. The general behaviour is similar to that of the STM32 devices, however, lower coil voltages seem to be more effective and the overall number of fault positions is lower. The timing plot is limited to $20\mu S$ for the sake of clearance, although we still experience successful fault injection later. However, the later the fault is injected, the higher chance that the attack is not successful or that parts of the flash memory have already been erased. Due to the low number of fault positions, we also investigated whether removing the capacitors influences the number of effective faults. For our setup and the parameters we evaluated, we achieve the best results when all capacitors are removed.

Based on the results of Figure 26, we chose a coil voltage of 275 V and an injection time of $8\mu S$ for all further experiments. In Figure 14, the superposition heatmap with the success rates in relation to the chip package is depicted. The heatmap shows that we achieve the highest success rates of 70% to 100% on a large area on the left side. For individual devices, we manage to successfully suppress flash erase for all 32 test runs (Figure 33). The

locations of the effective faults show a clear relation to the location of the flash memory we identified in Figure 7b. Since we experience no effective faults in other locations on the chip, we conclude that on this device, the high-voltage circuitry of the eFlash is disrupted which results in an ineffective mass erase.

AT32F415 Series. Our measurements for the AT32F415CBT7 microcontroller with different coil voltages of 300 V to 500 V, in steps of 100 V, show that this device is not susceptible when all capacitors are installed. For that reason, we remove all capacitors from the target board and evaluate the device again. Without capacitors and with a coil voltage of at least 450 V, we were able to suppress the flash erase operation. The results of the parameter exploration can be seen in Figure 27 in Appendix C.

For further experiments, we chose a coil voltage of 500 V and increase the number of delays and repetitions per delay. In the second step, we again analysed the complete attack with this reduced parameter set. Due to the initial evaluation with a single flash page we already know that the susceptible area is to the left 2 mm of the chip package. We achieve the best results for an injection delay of $15.1\mu S$. In Figure 15a, the superposition of all four heatmaps is depicted. The individual heatmaps of all devices can be found in Figure 34. The success rate for the superposition is only between 20% to 70% in the middle on the left edge of the chip package. Nevertheless, the AT32F415CBT7 microcontroller is still vulnerable to the flash erase suppression attack. Especially in scenarios where an attacker has multiple tries, for example when the IP stored on the device is of interest, this attack vector still poses a threat. When we compare the area of the effective faults with the location of the flash memory identified in the die shot (Figure 7c), no spatial relation is recognizable. We assume that on this device, we targeted other components such as the internal power supply of the microcontroller.

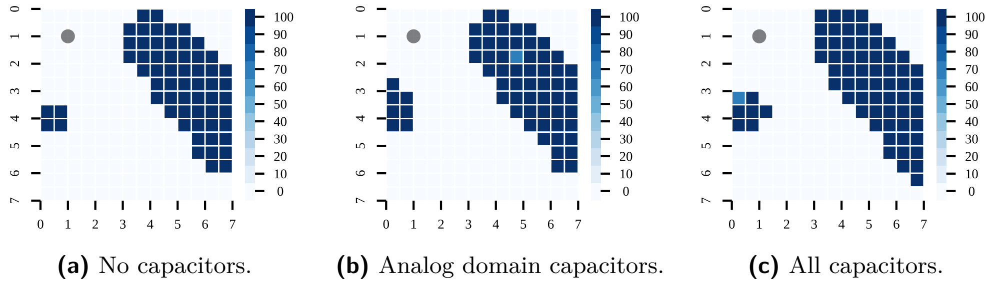


(a) AT32F415CBT7 **(b)** AT32F421C8T7

Figure 15: Heatmaps showing the overall success rate of the flash erase suppression attack from four devices for two AT32 microcontrollers.

AT32F421 Series. Based on the results of the AT32F415CBT7 microcontroller, we start our measurements with no capacitors equipped and similar coil voltages. The results of the parameter exploration can be found in Figure 28. Note that the fault positions after $40\mu S$ do not disappear but we finished the measurement from that point on. We observe no increase in the number of fault positions or an increase of contiguous area. For some configurations we observe single defects caused by the flash erase operation after only $27\mu S$. At around $20\mu S$, the number of fault positions drastically increases for 400 V and 500 V and for 300 V a small number of fault positions appears. In contrast to the

AT32F415CBT7, we observe for a coil voltage of 500 V a fives times higher number of fault positions. Remember that these measurements are performed for a single page only and we have no information about the condition of the remaining flash pages. Therefore, we use an injection delay of $21\mu S$ for evaluating the complete attack. As mentioned above, faulting the erase operation at its beginning increases the chance that all pages are still intact. Further, we used a coil voltage of 500 V subsequently.



(a) No capacitors. **(b)** Analog domain capacitors. **(c)** All capacitors.

Figure 16: Heatmaps showing the influence of the capacitors to the success rate of the flash erase suppression attack on the AT32F421C8T7 microcontroller.

In Figure 15b the resulting superposition heatmap is depicted. The heatmaps of all devices can be found in Figure 35 in Appendix D. It can be seen that the area for which the flash erase operation can be successfully suppressed is relatively wide and in the middle of this area success rates of 90% to 100% are achieved on all devices. When we compare the susceptible area with the die shot (Figure 7d) of the microcontroller, we can see that the area reflects the orientation of the die in the package. Also the location of the flash memory and the susceptible area matches.

In order to evaluate the influence of capacitors on the flash erase suppression attack, we use an additional device with three configurations: no capacitors, only capacitors of the analog domain, and all capacitors equipped. The resulting heatmaps of the three configurations are depicted in Figure 16. For each point on the heatmap, we executed four test runs. Based on the resulting heatmaps we conclude that the capacitors do not have a negative impact on the success rate. On the contrary, the area on the right side is even slightly larger when all capacitors are in place.

GD32E103 Series. The GD32E103C8T6 microcontroller exhibits a very high sensitivity against EMFI. We bricked one device by injecting a pulse with a coil voltage of 400 V and one device with a coil voltage of 200 V. The two bricked devices cause a short circuit as soon as they are powered and can no longer be operated.

After multiple measurements on small areas across the chip, we narrowed down the sensitive region where injecting a fault causes a permanent defect. We discovered that only at the very bottom edge of the package, the chip is susceptible to the flash erase suppression attack. To analyze, whether this region expands beyond the package, we extend the scan region by additional 1.5 mm at the bottom. This means that the injection coil is placed, to some extend, outside the chip package. The number of fault positions for different coil voltages and injection times are depicted in Figure 29 in Appendix C. The overall number of effective faults is low in comparison to the other microcontrollers we evaluated in this work. Based on these results we chose a coil voltage of 500 V and an injection delay of $16\mu S$ for all further evaluations.

The superposition heatmap is depicted in Figure 17. The heatmaps for all devices can be found in Figure 36 in Appendix D. For the reason we mentioned before, the heatmaps are larger than the actual package of the microcontroller. The red colored region on the heatmap marks the *keep out zone* so the device is not permanently damaged. Keeping

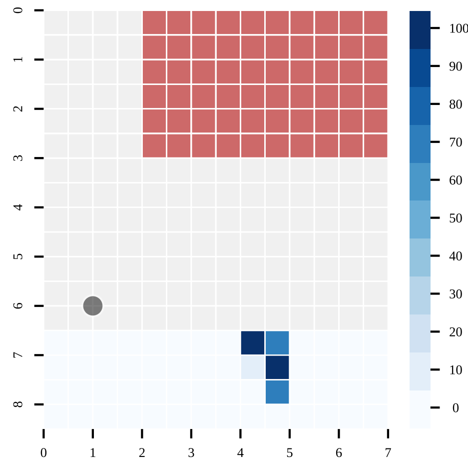


Figure 17: Heatmaps showing the overall success rate of the flash erase suppression attack from four devices for the GD32E103C8T6 microcontroller. The red colored area is the area that may lead to a permanent device defect if a fault is injected.

clear of this zone, we experienced no further device defects. There are two locations with a success rate of 70 % and 100 %. However, the shape of the area makes the correct positioning of the injection coil difficult. We also evaluate whether the success rate or the susceptible area can be improved by removing capacitors. For our setup and parameters, the bulk capacitor and the analog capacitors are necessary to achieve effective faults. Only the capacitors of the digital domain can be removed.

The fault location does not match with the eFlash we identified on the die shot in Figure 6. For that reason we assume that we inject the fault in the power supply of the microcontroller. The location of the faults matches with the location of the analog power supply pin of the chip, however, this can also be just a coincidence. Based on the optical inspection we cannot identify any component at the *keep out zone* which allows to infer the cause of the permanent defects. Further evaluation is necessary to identify the root cause of the faulty behaviour on this microcontroller.

Summary. In contrast to voltage glitching, we were able to successfully carry out the attack on all microcontrollers with EMFI. The results show that the attack can be transferred to different devices, once suitable parameters for a setup are found.

For all devices, the success rate is sufficiently high to extract the IP from flash memory if only a few target devices are available. Except for two microcontrollers, namely the STM32L151CBT6 and STM32L162ZDT6 (bank 1), we also can achieve a success rate of 100 %. However, the size and shape of the vulnerable area is very different for the other devices. For example, the AT32F421C8T7 and STM32L162ZDT6 (bank 2) both form a large contiguous area which means that the attack's repeatability and transferability make it a realistic threat for many scenarios. The injection coil can be placed by hand and without further aids. In an evil maid scenario where time and equipment is limited, an attack on these device seems feasible. In contrast, the small vulnerable area for the APM32F103C4T6 and GD32E103C8T6 make a stencil or positioning table necessary to reliably achieve a 100 % success rate. The shape of the vulnerable area also makes it very difficult to place the coil by hand.

For both microcontrollers of the STM32L1 series, we experience very different behaviour among the individual devices. On some devices, the vulnerable area is different or we observe no effect at all. Further experiments are necessary here to determine the cause.

8 Countermeasures

As the flash erase suppression attack targets the underlying hardware of the microcontroller and fully reactivates the debug interface, any general countermeasure must be implemented in hardware. A possible countermeasure could be to check the memory content after the erase operation. The debug interface is only unlocked when the entire flash memory content is erased. For a successful attack, an adversary would also have to fault this check. Another possible countermeasure would be to reduce the success probability by varying the time of the erase operation. This could be achieved, for example, by using a jittery clock with a variation of several microseconds. In case a static trigger is used like in this work, this approach would reduce the success probability. However, the high activity of the erase operation visible in the EM signal could be used as trigger. In this case, the countermeasure would make the attack more complicated in terms of required hardware but not necessarily reduce its success rate.

Without hardware support, only application-specific mitigations implemented in software are possible. One example for such a countermeasure is to store credentials encrypted in flash memory. This is possible, for example, in applications like hardware security tokens or crypto wallets. For these applications, the user provides the key to decrypt the credentials on-demand before an authentication or to sign a transaction.

In case the IP contained in the firmware of the microcontroller needs to be protected, countermeasures are more difficult to implement. One mitigation is code obfuscation in order to complicate reverse engineering. Obfuscation can also be used to impede the transfer of the firmware to other devices. For example, the authors in [CMG19, HMV+21] propose to use unique device secrets that determine the order of instructions. Even though reasonably high computational complexities for reverse engineering are derived in those works, it remains unclear how to realize the identifier the obfuscation is based on. If only chip IDs, as discussed in [HMV+21], are used, an adversary can obtain the value the same way as the obfuscated firmware in our setting. The same holds for physical unclonable identifiers, as long as no dedicated hardware that shields it from adversaries is in place. Without a secret that is inaccessible via the reactivated debug interface, obfuscation only complicates reverse engineering but cannot be considered a proper countermeasure.

There are other mitigations that can be seen as a trade-off between reliability and security. For example, using a protection level for which the debug interface is permanently disabled, if available. This comes with the downside that failure analysis may not be possible anymore. For certain use cases, however, this might be expedient. For others, at least the trade-off between potential failure analysis and the device security should be reevaluated. Note that for some devices, even deactivating the debug interface does not provide sufficient security when further hardware attacks are considered.

9 Conclusion and Outlook

In this work, we evaluate the susceptibility of microcontrollers to the so-called *flash erase suppression* attack. Our results show that this attack vector affects multiple devices across different manufacturers. In comparison to state-of-the-art literature, our evaluation takes difficulties such as reproducibility and transferability into account which arise under real-world conditions. For all microcontrollers, we achieve success rates that are sufficiently high that with a small number of available target devices, the containing IP can be extracted. On some microcontrollers, we even reach a success rate of 100 % which allows an adversary to extract device unique secrets. The device intrinsic variations we experience in this work confirm that experiments on single devices as done so far [SWUH21, Sko05] are not sufficient to assess the threat of this attack vector. Despite the fact that our analysis had to be carried out in a black-box setting, we provide insights into the root cause of this

attack vector. Based on optical inspection we could not find a spatial relation between the introduced faults and the embedded flash memory (eFlash) on the chip. In combination with the insights of our side-channel analysis (SCA), we come to the conclusion that an injected fault disturbs the power supply or high-voltage circuitry of the eFlash which leads to a suppressed flash erase operation.

Implications. Our results show that the flash erase suppression attack vector needs to be considered by manufacturers of microcontrollers and the companies who build products based on those microcontrollers. Due to the attack's high success rate and transferability even on simple lab setups, microcontroller vendors should acknowledge the attack vector by integrating it into their threat model such that product manufacturers can judge its threat correctly for their use case. These product vendors, in turn, should try to mitigate the attack vector in software, if possible. We want to emphasize that our results should not be understood as the best attack that can be mounted on the evaluated devices but as a lower bound of what is possible. Due to the high number of parameters, better results in terms of affected area and success rate are likely possible. An attacker targeting a specific microcontroller will most likely extend the parameter exploration to find favorable parameters for that device. Rather, our results should raise awareness for developers and manufacturers alike.

Future Work. Further investigation of possible countermeasures is needed to protect future products. A complete mitigation of the attack can only be achieved by hardening the microcontroller's hardware. As discussed in the previous section, the retrofitting of countermeasures to existing devices is highly dependent on the use case. Further research in relation to the attack could evaluate whether microcontrollers using other non-volatile memory technologies such as EEPROM, MRAM, and FRAM are also susceptible to erase suppression.

References

[App07] Appaloosa. Aufbau einer flash-ram-speicherzelle, January 2007. https://commons.wikimedia.org/wiki/File:FLASH_RAM-Cell.svg.

[Art22a] Artery Technology. *AT32F415 Series Reference Manual*, June 2022. Rev. 2.00.

[Art22b] Artery Technology. *AT32F421 Series Reference Manual*, June 2022. Rev. 2.01.

[BFP19] Claudio Bozzato, Riccardo Focardi, and Francesco Palmarini. Shaping the glitch: Optimizing voltage fault injection attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):199–224, Feb. 2019.

[Bro15] Kris Brosch. Firmware dumping technique for an ARM Cortex-M0 SoC. https://blog.includesecurity.com/2015/11/NordicSemi-ARM-SoC-Firmware-dumping-technique.html, November 2015.

[CGOZ99] Paulo Cappelletti, Carla Golla, Piero Olivo, and Enrico Zanoni. *Flash Memories*. Springer New York, first edition, 1999.

[CMG19] Benjamin Cyr, Jubayer Mahmod, and Ujjwal Guin. Low-cost and secure firmware obfuscation method for protecting electronic systems from cloning. *IEEE Internet of Things Journal*, 6(2):3700–3711, 2019.

[Fed23] Federal Office for Information Security. A study on hardware attacks against microcontrollers, March 2023. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Hardware-Attacks/Hardware-Attacks-Microcontroller.html.

[Hid17] Hideto Hidaka. *Embedded Flash Memory for Embedded Systems: Technology, Design for Sub-Systems, and Innovations.* Springer Publishing Company, Incorporated, first edition, 2017.

[HMV+21] Muhammad Monir Hossain, Sajeed Mohammad, Jason Vosatka, Jeffery Allen, Monica Allen, Farimah Farahmandi, Fahim Rahman, and Mark Tehranipoor. Hexon: Protecting firmware using hardware-assisted execution-level obfuscation. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 343–349, 2021.

[Lim19] LimitedResults. Pwn the ESP32 crypto-core, June 2019. https://limitedresults.com/2019/08/pwn-the-esp32-crypto-core/.

[Lim20a] LimitedResults. nRF52 Debug Resurrection (APPROTECT Bypass) Part 1, June 2020. https://limitedresults.com/2020/06/nrf52-debug-resurrection-approtect-bypass/.

[Lim20b] LimitedResults. nRF52 Debug Resurrection (APPROTECT Bypass) Part 2, June 2020. https://limitedresults.com/2020/06/nrf52-debug-resurrection-approtect-bypass-part-2/.

[Lim20c] LimitedResults. Nuvoton M2351 MKROM, January 2020. https://limitedresults.com/2020/01/nuvoton-m2351-mkrom-armv8-m-trustzone/.

[Lim21] LimitedResults. Enter the EFM32 Gecko, June 2021. https://limitedresults.com/2021/06/enter-the-efm32-gecko/.

[OSM20] Johannes Obermaier, Marc Schink, and Kosma Moczek. One exploit to rule them all? on the security of drop-in replacement and counterfeit microcontrollers. In *Proceedings of the 14th USENIX Conference on Offensive Technologies*, WOOT'20, USA, 2020. USENIX Association.

[OT17] Johannes Obermaier and Stefan Tatschner. Shedding too much light on a microcontroller's firmware protection. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC, August 2017. USENIX Association.

[Ren19] Renesas Electronics Corporation. *NOR Flash Memory Erase Operation*, November 2019. Rev. A.

[Sil17] Silicon Labs. *EFM32PG12 Pearl Gecko Family Reference Manual*, January 2017. Rev. 0.5.

[Sko05] Sergei Skorobogatov. Semi-invasive attacks – a new approach to hardware security analysis. April 2005. https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-630.pdf.

[SO19] Marc Schink and Johannes Obermaier. Taking a look into execute-only memory. In *13th USENIX Workshop on Offensive Technologies (WOOT 19)*, Santa Clara, CA, August 2019. USENIX Association.

[SO20] Marc Schink and Johannes Obermaier. Exception(al) Failure – Breaking the STM32F1 Read-Out Protection. https://blog.zapb.de/stm32f1-exceptional-failure/, March 2020.

[STM17] STMicroelectronics. *STM32L100xx, STM32L151xx, STM32L152xx and STM32L162xx advanced ARM®-based 32-bit MCUs*, September 2017. Rev. 15.

[STM18] STMicroelectronics. *STM32L41xxx/42xxx/43xxx/44xxx/45xxx/46xxx advanced ARM ®-based 32-bit MCUs*, October 2018. Rev. 4.

[SWUH21] Marc Schink, Alexander Wagner, Florian Unterstein, and Johann Heyszl. Security and trust in open source security tokens. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):176–201, Jul. 2021.

[Tor17] Giulio Torrente. *Investigation of degradation mechanisms and related performance concerns in 40nm NOR Flash memories*. Phd thesis, Community Université Grenoble Alpes, July 2017.

A Device Analysis

The EM side-channel traces of the STM32L1 microcontrollers differ slightly from the other devices. For that reason, we take a closer look at them.

The EM side-channel trace during the debug unlock operation of the STM32L151CBT6 microcontroller is depicted at the top of Figure 18. In contrast to the other devices, we can identify three instead of two distinct blocks. We assume that the additional block is related to the integrated data EEPROM. Like the flash memory, the EEPROM is erased during the debug unlock operation [STM17]. To inspect the unlock operation in detail, we zoom into the signal directly after the trigger (A), after the first block (B), and after the second block (C). The resulting plot is depicted at the bottom of Figure 18.

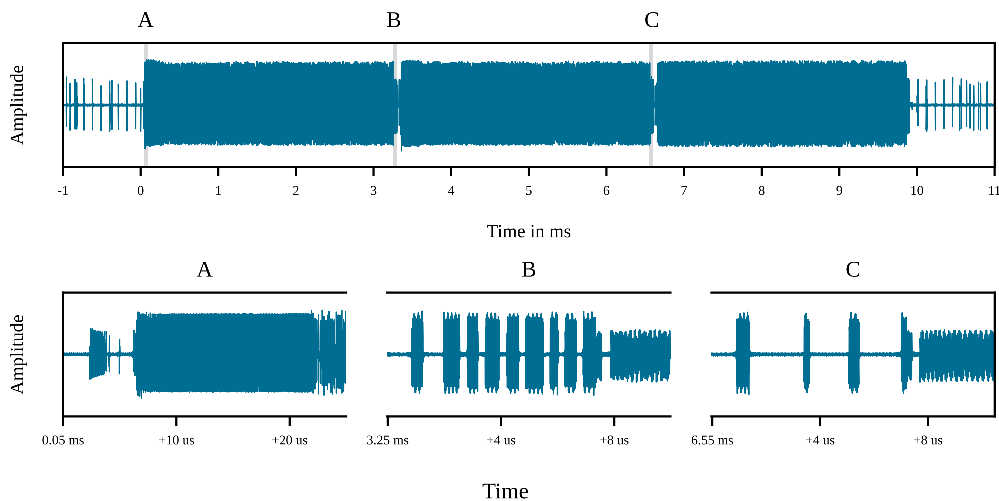


Figure 18: EM trace of the STM32L151CBT6 microcontroller during a debug unlock operation (top) and a detailed view (bottom).

In Section (A), we observe a similar pattern as for the STM32L422KBT6 microcontroller: a short block followed by a longer block of EM activity. We conclude that this is the mass erase of the flash memory. In section (B), there are nine short blocks of high EM activity. We assume that these blocks corresponds to write operations of the flash memory or EEPROM but can not conclusively clarify its function. In section (C), we observe four additional blocks of activity. Since the microcontroller is a so-called *Cat.1* device [STM17], the number of blocks matches with the amount of 32-bit words of the configuration block stored in flash memory. The assumption that the four blocks represent the write operation of the configuration block is strengthened by the fact that we do not see any activity at the end of the debug unlock operation.

The STM32L162ZDT6 microcontroller is a so-called *Cat.4* device with a dual-bank flash memory and additional EEPROM as data storage [STM17]. The EM signal of the entire debug unlock operation is depicted at the top of Figure 19. The signal has a structure similar to that of the STM32L151CBT6 microcontroller: three large blocks of activity with a duration of about 4 ms each. To inspect the unlock operation in detail, we zoom into the signal directly after the trigger (A), after the first block (B), and after the second block (C). The resulting plot is depicted at the bottom of Figure 19. Section (A) and (C) are similar to the STM32L151CBT6 microcontroller while section (B) shows a different pattern: we observe only three or four blocks of activity instead of nine. For this device we cannot match the blocks of EM activity neither in section (B) nor (C) with the number of 32-bit configuration data words. We would expect 10 blocks of activity, one for each of

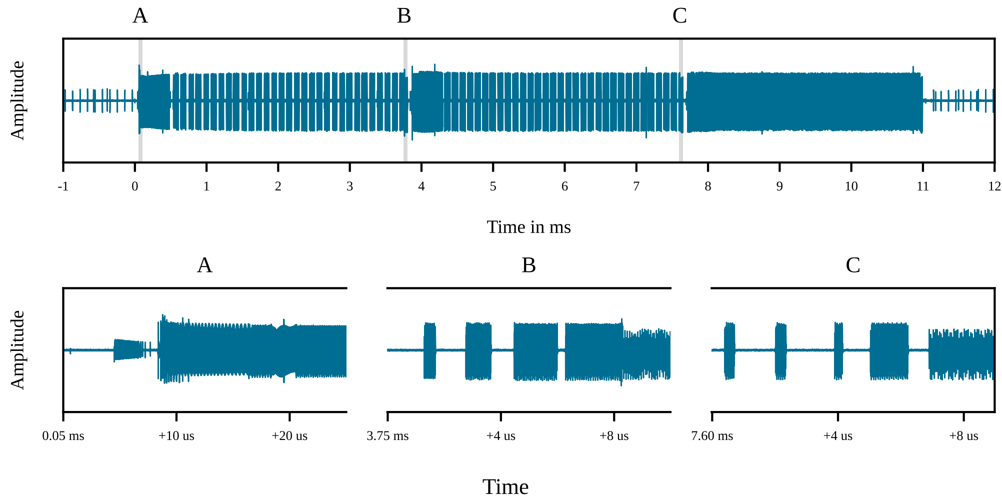


Figure 19: EM trace of the STM32L162ZDT6 microcontroller during a debug unlock operation (top) and a detailed view (bottom).

the 32-bit configuration data words of a *Cat.4* microcontroller.

In contrast to the STM32L4 series, we observe no dedicated erase operation for the configuration data on both devices of the STM32L1 series. Based on the EM side-channel and the information provided by the data sheet, we assume that the first large two blocks correspond to the erase operation of the flash memory and EEPROM. However, it remains unclear what function the last block has. Further investigation is necessary to determine a relation between the EM activity and the individual operations of the debug unlock operation.

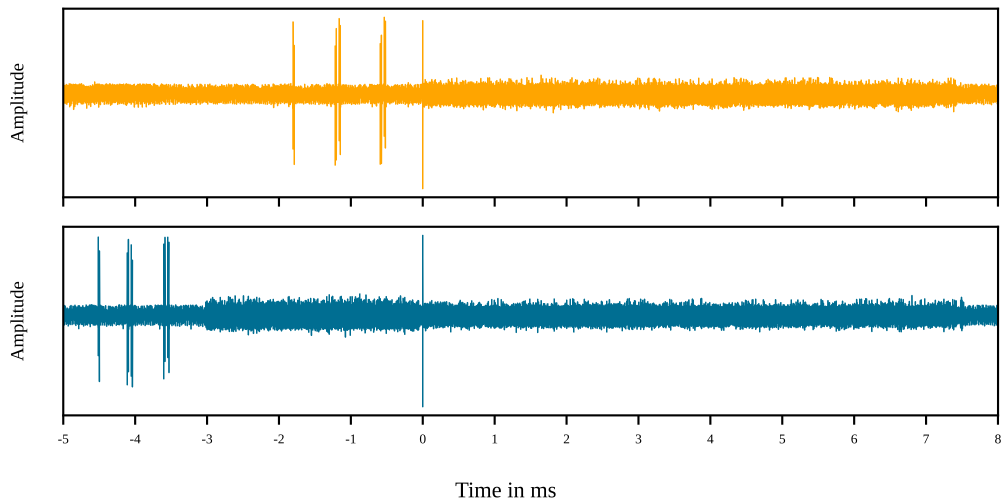


Figure 20: EM trace of the AT32F421C8T7 microcontroller during a mass erase (top) and debug unlock (bottom) operation.

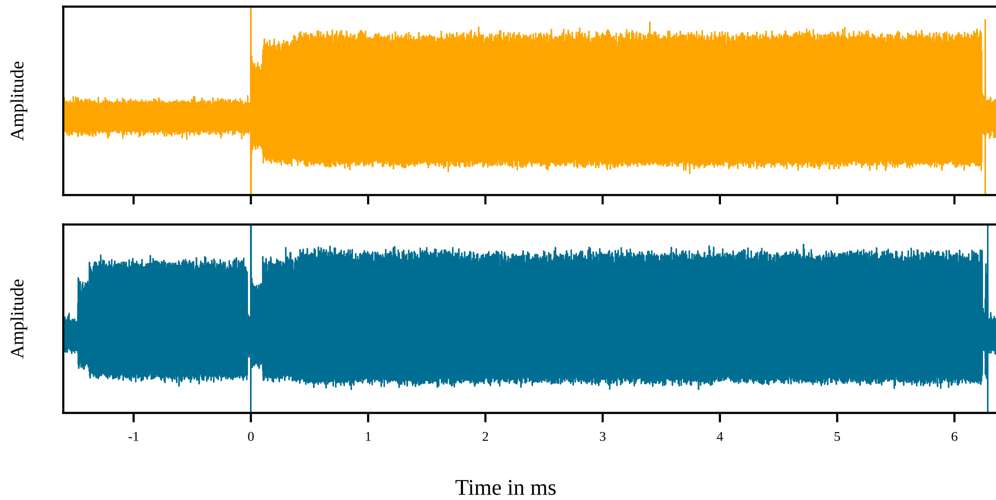


Figure 21: EM trace of the APM32F103C4T6 microcontroller during a mass erase (top) and debug unlock (bottom) operation.

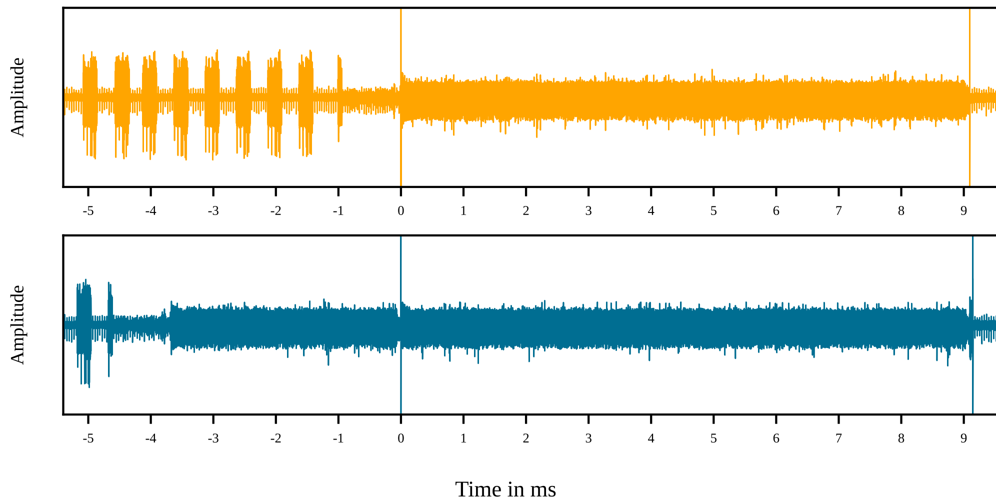


Figure 22: EM trace of the GD32E103C8T6 microcontroller during a mass erase (top) and debug unlock (bottom) operation.

B Voltage Glitching: Parameter Exploration

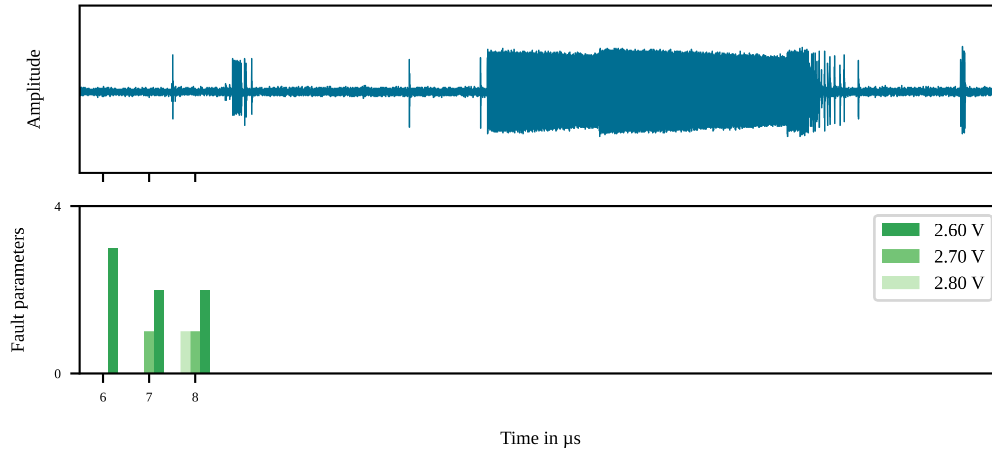


Figure 23: EM trace of the AT32F415CBT7 microcontroller during the mass erase operation (top) and the number of effective faults parameters (bottom) at different injection times and supply voltages.

C EMFI: Parameter Exploration

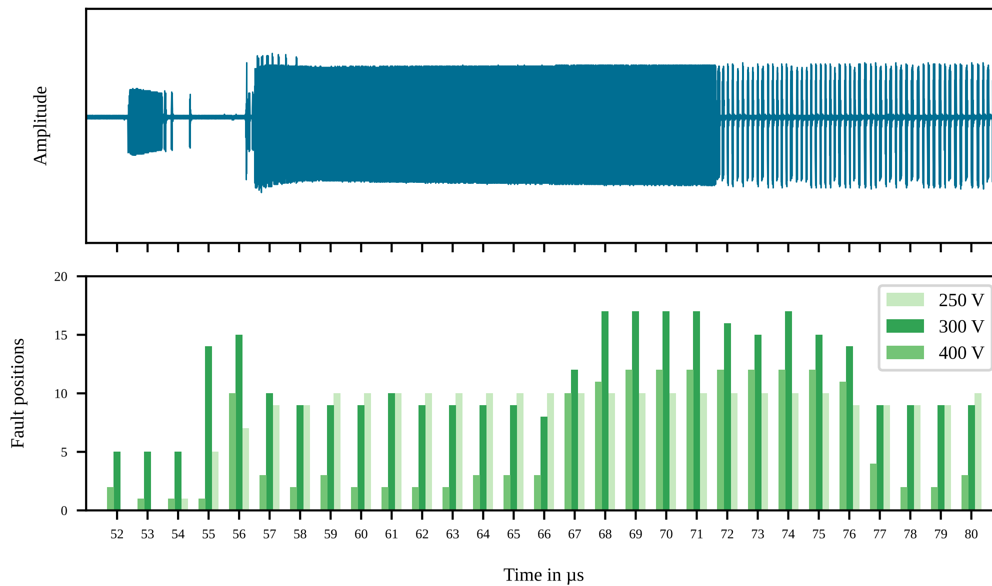


Figure 24: EM trace of the STM32L151CBT6 microcontroller during the debug unlock operation (top) and the number of effective faults positions (bottom) at different injection times and coil voltages.

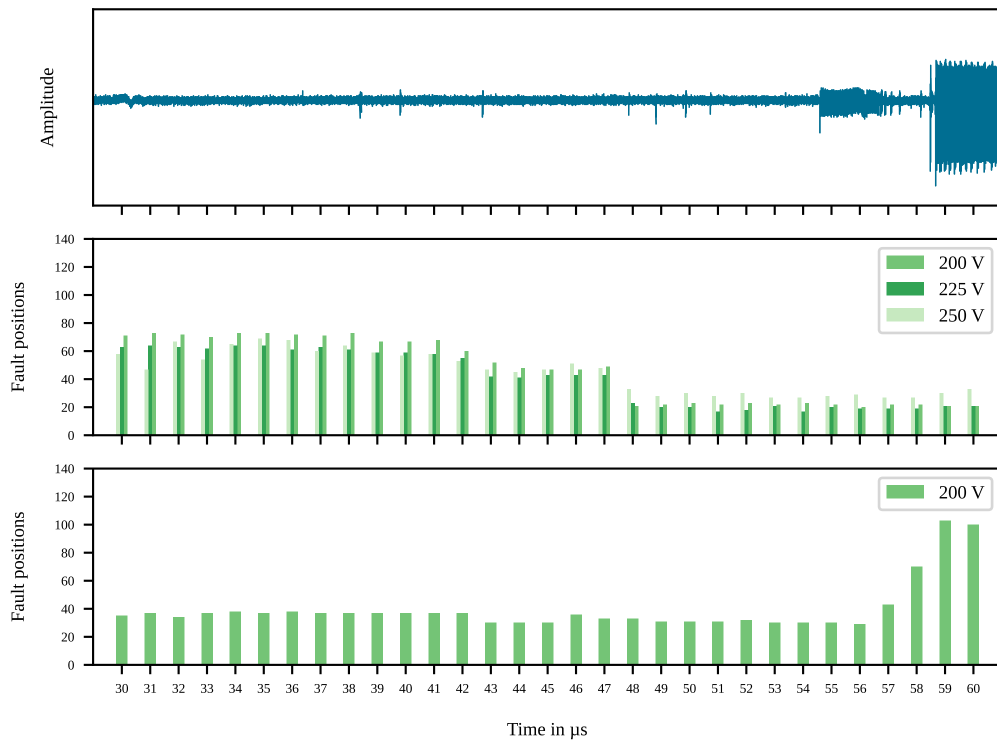


Figure 25: EM trace of the STM32L162ZDT6 microcontroller during the debug unlock operation (top) and the number of effective faults positions for flash bank 1 (middle) and bank 2 (bottom) at different injection times and coil voltages.

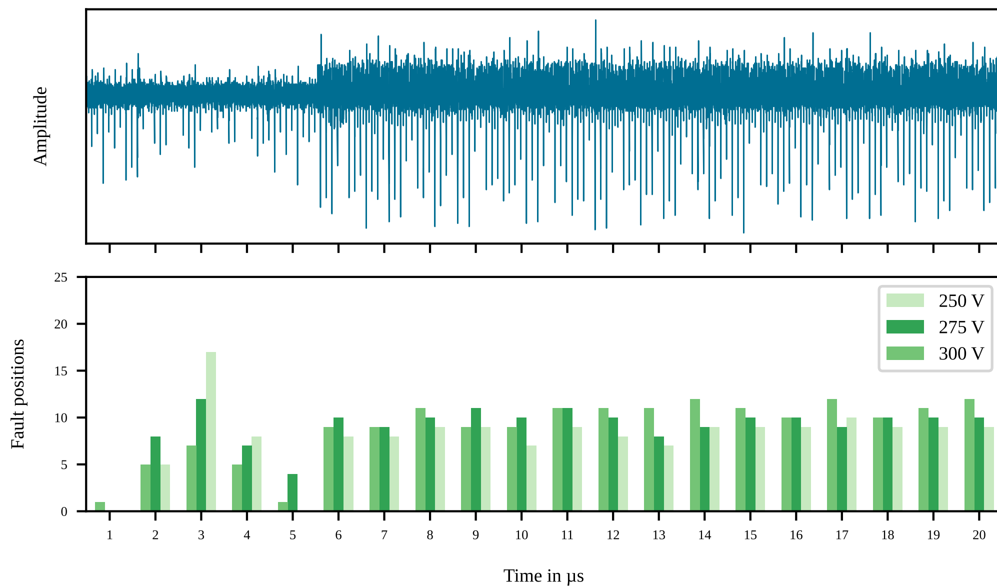


Figure 26: EM trace of the APM32F103C4T6 microcontroller during the debug unlock operation (top) and the number of effective faults positions (bottom) at different injection times and coil voltages.

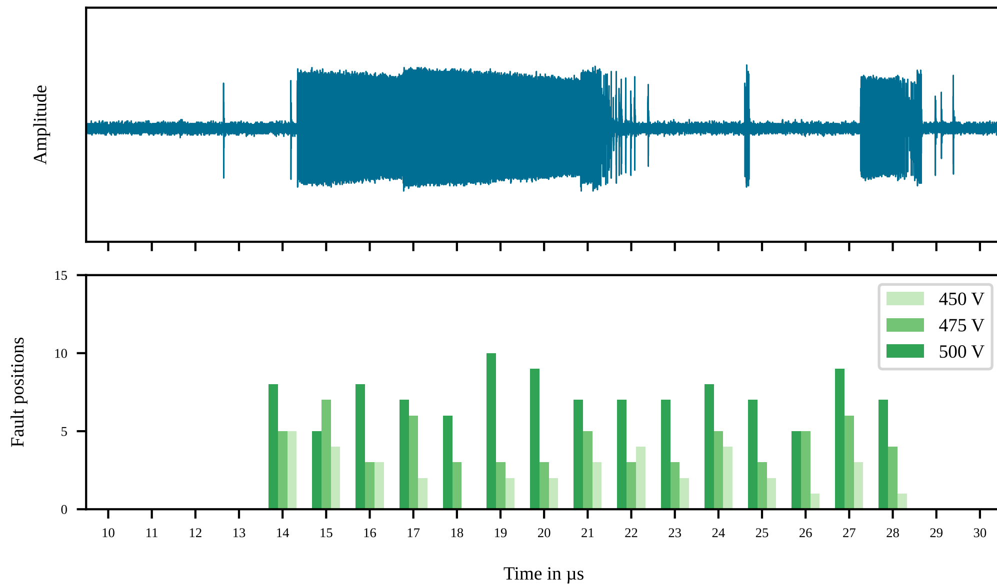


Figure 27: EM trace of the AT32F415CBT microcontroller during the debug unlock operation (top) and the number of effective faults positions (bottom) at different injection times and coil voltages.

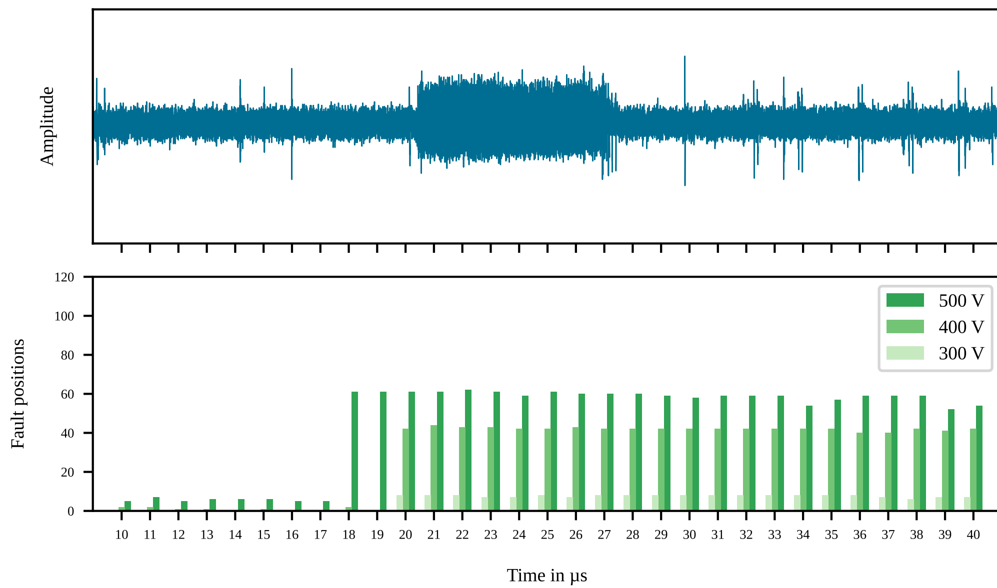


Figure 28: EM trace of the AT32F421C8T7 microcontroller during the debug unlock operation (top) and the number of effective faults positions (bottom) at different injection times and coil voltages.

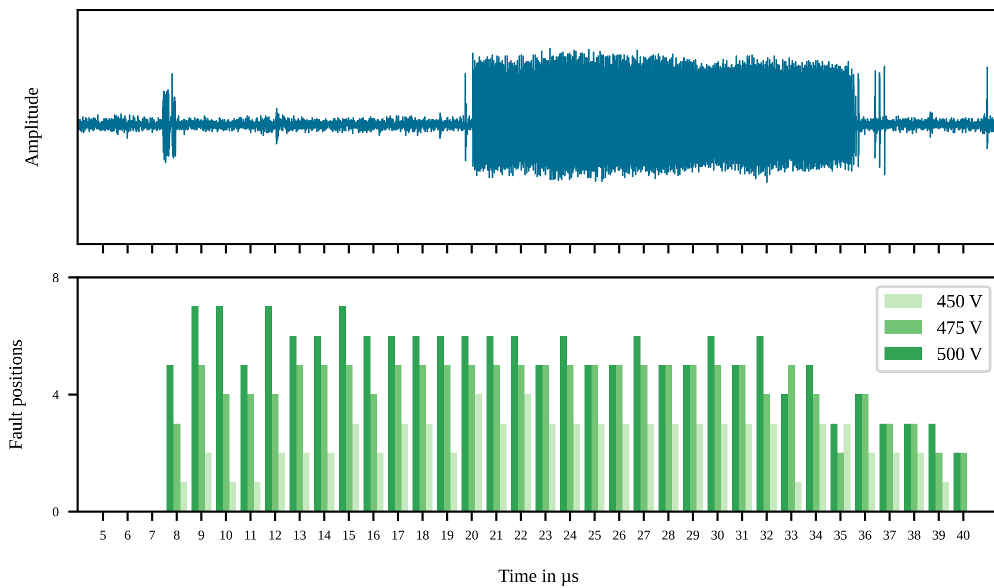
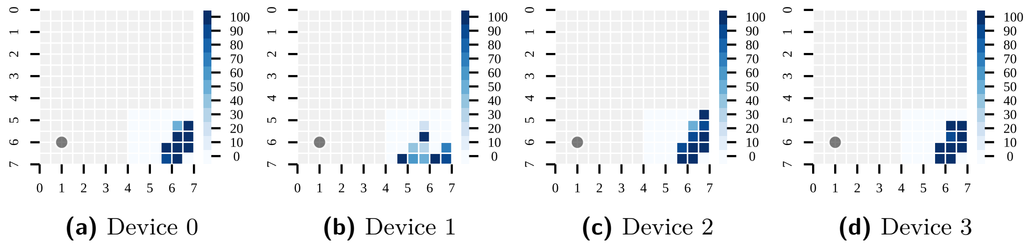


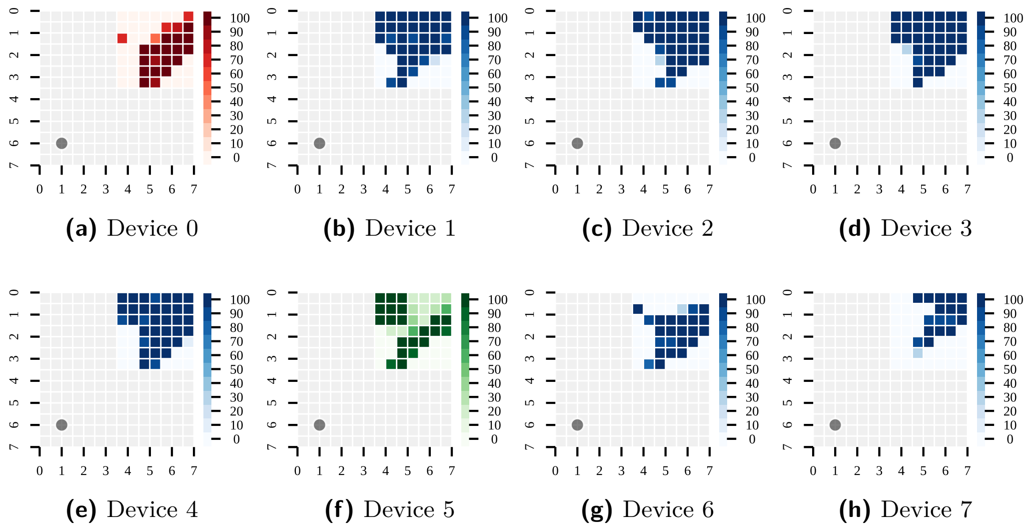
Figure 29: EM trace of the GD32E103C8T6 microcontroller during the debug unlock operation (top) and the number of effective faults positions (bottom) at different injection times and coil voltages.

D EMFI: Heatmaps



(a) Device 0 **(b)** Device 1 **(c)** Device 2 **(d)** Device 3

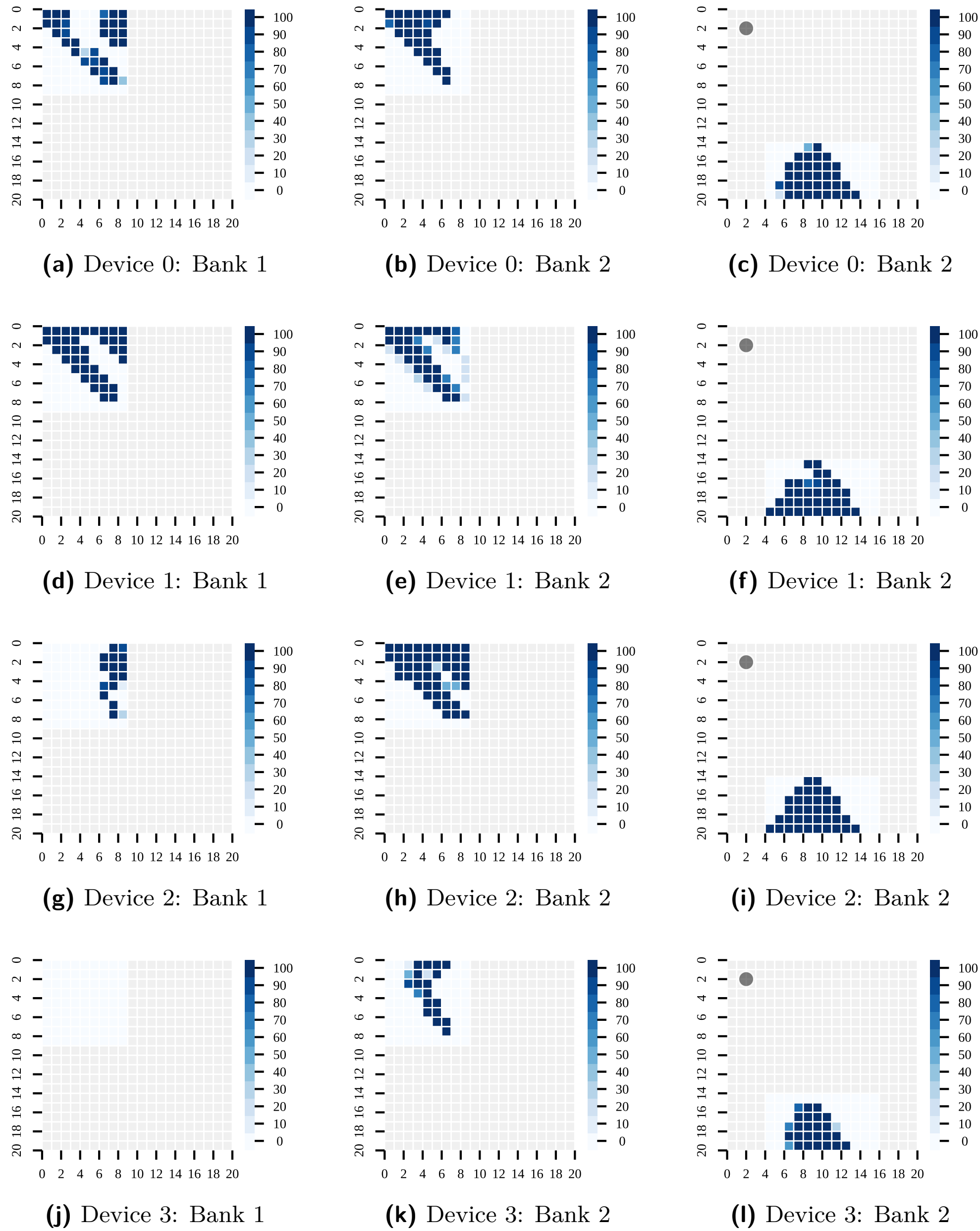
Figure 30: Heatmaps showing the success rate of the flash erase suppression attack from four individual STM32L422KBT6 devices.



(a) Device 0 **(b)** Device 1 **(c)** Device 2 **(d)** Device 3

(e) Device 4 **(f)** Device 5 **(g)** Device 6 **(h)** Device 7

Figure 31: Heatmaps showing the success rate of the flash erase suppression attack from eight individual STM32L151CBT6 devices. The colored heatmaps indicate that only parts of the flash memory are retained (red) and that the firmware execution failed but the entire flash memory is retained (green).



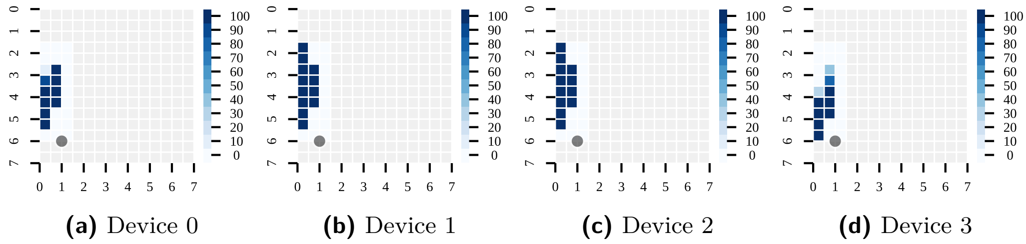
(a) Device 0: Bank 1 **(b)** Device 0: Bank 2 **(c)** Device 0: Bank 2

(d) Device 1: Bank 1 **(e)** Device 1: Bank 2 **(f)** Device 1: Bank 2

(g) Device 2: Bank 1 **(h)** Device 2: Bank 2 **(i)** Device 2: Bank 2

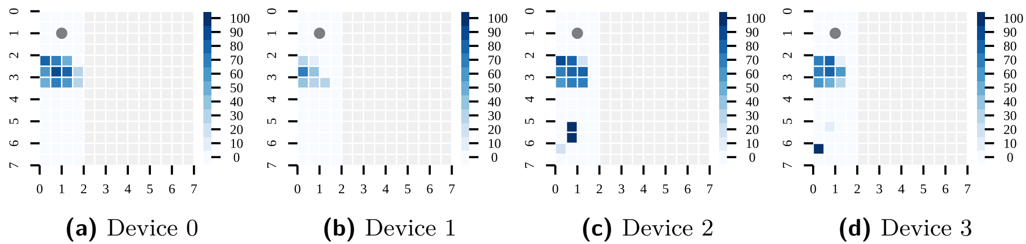
(j) Device 3: Bank 1 **(k)** Device 3: Bank 2 **(l)** Device 3: Bank 2

Figure 32: Heatmaps showing the success rate of the flash erase suppression attack from four individual STM32L162ZDT6 devices. For each device, results are shown for both flash banks at different locations and injection times.



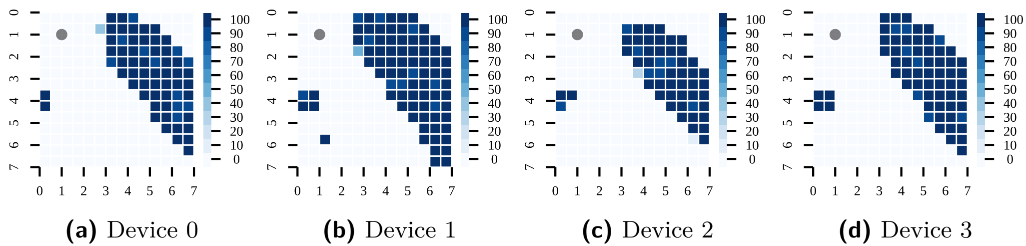
(a) Device 0 **(b)** Device 1 **(c)** Device 2 **(d)** Device 3

Figure 33: Heatmaps showing the success rate of the flash erase suppression attack from four individual APM32F103C4T6 devices.



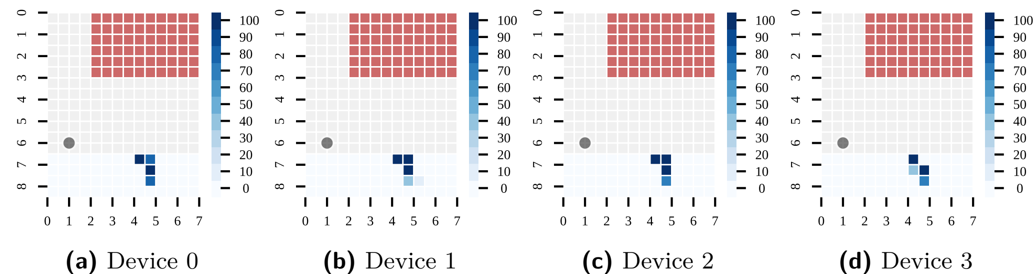
(a) Device 0 **(b)** Device 1 **(c)** Device 2 **(d)** Device 3

Figure 34: Heatmaps showing the success rate of the flash erase suppression attack from four individual AT32F415CBT devices.



(a) Device 0 **(b)** Device 1 **(c)** Device 2 **(d)** Device 3

Figure 35: Heatmaps showing the success rate of the flash erase suppression attack from four individual AT32F421C8T7 devices.



(a) Device 0 **(b)** Device 1 **(c)** Device 2 **(d)** Device 3

Figure 36: Heatmaps showing the success rate of the flash erase suppression attack from four individual GD32E103C8T6 devices. The red colored area represents an area in which there is a risk of permanent destruction of the device due to EMFI.

E Coordinated Disclosure

As part of a coordinated disclosure process, we informed the security teams of all affected products about our findings. Technical and detailed information were provided more than 90 days prior to the publication of this paper. Table 5 lists the affected microcontrollers identified in this work together with the corresponding device series, manufacturer, and the assigned CVE numbers.

Table 5: Assigned CVE numbers for each affected microcontroller identified in this work.

Manufacturer	Device series	Target device	CVE number
Artery	AT32F415	AT32F415CBT7	CVE-2024-21740
	AT32F421	AT32F421C8T7	
Geehy	APM32F1	APM32F103C4T6	CVE-2024-21739
GigaDevice	GD32E103	GD32E103C8T6	CVE-2024-21741
STMicroelectronics	STM32L1	STM32L151CBT6	CVE-2021-29414
		STM32L162ZDT6	
	STM32L4	STM32L422KBT6	