Federal Office
for Information Security

A Study on Hardware Attacks against Microcontrollers

Document history

Version	*Date*	*Editor*	*Description*
1.0	17.03.2023		Final Version

Table 1: Document history

Federal Office for Information Security
P.O. Box 20 03 63
53133 Bonn
Internet: https://www.bsi.bund.de

Acknowledgement

This report was authored by the Fraunhofer Institute for Applied and Integrated Security (AISEC) on behalf of the German Federal Office for Information Security (BSI).

Table of Contents

List of Abbreviations

ADC Analog-digital converter

AES Advanced encryption standard

ANSSI Agence Nationale de la Sécurité des Systémes d'Information

COTS Commercial off-the-shelf

CPU Central processing unit

CRC Cyclic redundancy check

ECC Elliptic curve cryptography

ECC Error correction code

EM Electromagnetic

EMFI Electromagnetic fault injection

FCC Federal communications commission

FIB Focused ion beam

GPIO General-purpose input/output

IoT Internet of things

IP Intellectual property

IR Intermediate representation

ISA Instruction set architecture

LFI Laser fault injection

NDA Non-disclosure agreement

OTA Over-the-air

PC Program counter

POS Point-of-sale

ROP Return-oriented programming

RSA Rivest-Shamir-Adleman

SNR Signal noise ratio

SRAM Static random-access memory

TTM Time to market

USB Universal Serial Bus

UV Ultra-violet

XOM Execute-only memory

1 Introduction

Microcontrollers are widely used in numerous applications such as robotics, medical devices, aerospace, and automotive. Concepts such as the internet of things (IoT) further increase their pervasiveness in 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 sensitive applications such as access control systems. 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 are of utmost importance for today's applications.

This security must be carefully evaluated. For one, threats known for other computer systems such as desktops and servers also apply to microcontrollers. Additionally, there is an attack type, which can be ignored in many cases for desktop and server platforms: the hardware attack. Due to their pervasiveness and accessibility, hardware attacks and their countermeasures must be taken into special consideration for microcontrollers. Other computing systems are less easy to access for adversaries, therefore hardware attacks are less relevant but not completely negligible. Despite restrictions, an attacker might gain access to the device or induce hardware attacks remotely. However, this report focuses on hardware attacks on accessible microcontrollers and does not further consider other computing systems. Hardware attacks are often not part of the threat model of microcontroller manufacturers, but nevertheless, these attacks are frequently reported and need to be addressed. Vulnerabilities on the hardware level allow the adversary to mount attacks, regardless of the application software.

There exist security controllers with built-in protections against hardware attacks and satisfy certain assurance levels assessed by independent evaluation laboratories [2]. However, commercial off-the-shelf (COTS) microcontrollers are prevalent in security relevant IoT products. The reasons are manifold but in the end mostly price-driven. COTS devices are mostly cheaper and easier to acquire. In contrast to security controllers where usually a non-disclosure agreement (NDA) is required for detailed product information, COTS microcontrollers benefit from publicly available tools, documentation, and code which accelerates development and time to market (TTM). Finally, microcontroller manufacturers and product designers have limited awareness of potential threats through hardware attacks. Manufacturers are gradually starting to build countermeasures against hardware attacks into some of their product families. However, these devices only slowly find their way into products, while devices which are already deployed in the field remain vulnerable to this kind of attacks. Therefore, effective countermeasures, particularly those that can be retrofitted into existing devices are important for products already in circulation.

This report is meant to create awareness for hardware attacks on critical infrastructure that is built upon COTS microcontrollers. After reading, both technical-proficient and non-proficient readers should be able to judge the threat posed by unprotected implementations . First, an overview of COTS microcontrollers which are used in security relevant IoT products is given. An online market analysis based on publicly available information was conducted to substantiate the relevance of microcontrollers for such products. Further, this report gives an overview of state-of-the-art hardware attacks against microcontrollers. Based on a literature research, the most important attack vectors are elaborated and summarized. Three hardware attacks and suitable countermeasures that are of interest for further investigation are proposed. As mitigation, software-based countermeasures that can be retrofitted for such devices with minimal cost and without a significant degradation of performance or user experience are promoted. As long as hardware countermeasures are only slowly adapted for COTS microcontrollers, software countermeasures are the only way to protect products from hardware attacks. The report concludes with a laboratory analysis where two of the proposals are practically evaluated. The results demonstrate the effectiveness of software countermeasures and are meant to motivate vendors to consider these countermeasures for their products.

2 Market Analysis on the use of Microcontrollers

In order to understand which microcontrollers are used in security-relevant products, a market analysis is conducted. The analysis is based on different product categories to estimate the share of microcontrollers and manufacturers across application domains.

Market Analysis. The starting point for the market analysis is a list of microcontroller manufacturers that was created in the course of the work of Fraunhofer AISEC in the field of hardware security. To avoid a possible bias or forgetting manufacturers, this list was compared and completed with an independent and public list of microcontroller manufacturers [4]. From these manufacturers a list of product categories they propose for their devices was obtained. These categories are boiled down to those where microcontrollers are used in a security-relevant context. For each application, representative products are taken into account and their internals are investigated. Since product manufacturers rarely provide publicly available information about integrated components such as microcontrollers, other sources are used. This includes but is not limited to publicly available teardowns of products by the FCC and researchers in forums, blogs, and journals. Our findings are condensed into overview tables for the respective sections on product categories.

It is important to note that our market analysis does not represent the exact and complete distribution of microcontroller devices in a product category. There are multiple reasons for this: first, the fact that manufacturers hardly ever publish information on integrated components, as already mentioned. Second, the sources that are consulted for these information instead do not fully reflect the market. Pricing, availability, and popularity make certain products more likely to appear in public teardowns. Naturally, this could lead to a potential over- or underrepresentation of certain vendors in our market analysis. But even if the study does not paint the complete picture, it suffices for the reader to accurately judge the threat of hardware attacks in the identified categories. For all categories, products incorporating microcontrollers by four or more different vendors were found. The identified microcontrollers are of similar price ranges and capability levels. None of these include dedicated countermeasures against hardware attacks. Therefore, most products of the respective categories are expected to feature identical or similar microcontrollers and the subsequent listings are considered to be representative within this study.

The following applications were identified across vendors and are deemed most relevant for this study:

- hardware security tokens
- hardware crypto wallets
- smart locks
- point-of-sale (POS) terminals

Adversary Model. It is assumed an adversary that has either the incentive to extract intellectual property (IP) or cryptographic secrets. The consequences of such an attack can be of economic relevance, e.g. stealing from an economic competitor, or security relevance, e.g. compromising critical infrastructure, or both, e.g. compromising critical infrastructure to harm a competitor. However, the exact intentions behind IP theft or extraction of cryptographic secrets are not further explored in this report.

For the extraction of cryptographic secrets the attacks launched can be further classified as supply chain attack, evil maid attack, or straightforward theft:

- A supply chain attack targets the delivery of products. The adversary can either compromise the shipped device or ship own malicious devices which seem to behave like an original device.
- If a product is already in use an attacker may compromise it in an evil maid fashion or try to steal it. The evil maid scenario is defined by the fact that the adversary has only access for a limited time and

tries to avoid changes to the outer appearance or observable behavior of the device. In this case, if the user of a device leaves it out of sight temporarily, i.e. a few minutes up to a day, the attack of the evil maid is not detected.

- If the adversary steals the device, the time span for attacks is longer and damages to hardware or software are unimportant. However, in certain scenarios, the victim can take action to invalidate the cryptographic secrets stored on the device if the theft is detected.

Within these three attack types, an adversary has the possibility to indirectly or to directly extract secrets. The direct approach allows the adversary to immediately extract secrets by circumventing existing countermeasures. Depending on the application, this attack can be used in all three attack types. In contrast, the indirect approach is to tamper the functionality. This can lead to the insecure usage of cryptographic algorithms, which is then exploited by an attacker. Since the attacker has to use the device after the modification to be effective, only evil maid and supply chain attacks are possible. For example, forcing the product to use a low-quality or even static source of randomness can lead to easy-to-guess keys, which may be brute-forced by an attacker in a second step. Further, this allows to influence the overall behavior of the product, which leads to a security compromise. For instance, if the product does no longer verify the authenticity and integrity of a request, which shall be assured cryptographically.

Hardware Security Tokens. Hardware security tokens are used for authentication to web or local services. They can be used as a single or second factor. A single factor authentication is used for the so called password-less authentication. Due to the ease of use this method is becoming increasingly popular. The fact that the security relies on a single secret stored on the device makes the token particularly interesting for an adversary. Compromising the hardware token is sufficient for a successful authentication. In general, hardware security tokens store attestation keys as well as service keys, which are both of interest for an adversary. Besides the keys, the functionality to generate new keys is also of interest. The attestation keys are long-term secrets, which are loaded by the manufacturer or are created by the device itself during production. These are used to attest the authenticity and integrity of a device to a user or a service. If an attacker has access to such attestation keys, they would be able to ship own devices, which generate valid attestation proofs. The service keys are run-time secrets, which are loaded by the user or created by the device itself during the usage, for instance, secrets specifically used to gain access to a service. Having knowledge about service keys would allow to authenticate against a service without being in possession of the respective hardware security token. This would undermine the security guarantees given by hardware security tokens.

Table 2.1: Hardware security tokens and their integrated microcontrollers found in the market analysis.

Vendor	***Device Family***	***Product***
Microchip (former Atmel)	AT32UC3A	Nitrokey Storage 2
STMicroelectronics	STM32F1	Nitrokey Start
		Nitrokey Pro 2
		Nitrokey HSM 2
	STM32L4	Nitrokey FIDO2
		SoloKeys Solo 1 USB-A
		SoloKeys Solo 1 USB-C
		SoloKeys Solo 1 Tap USB-A
		SoloKeys Somu
NXP	LPC55S6	Nitrokey 3
		Solokeys Solo 2A+
	Kinetis	OnlyKey OnlyKey
	LPC11Uxx	Google Titan Security Key
Silicon Labs	EFR32BG13	Certgate AirID FIDO

The results of the market analysis, summarized in Table 2.1, show that mostly microcontrollers from STMicroelectronics or NXP are used within hardware security tokens. Only few devices are based on microcontrollers from Microchip or Silicon Labs.

Hardware Crypto Wallets. While the actual record of crypto currency is stored in a public blockchain, the secret key that provides access to the actual owner must be stored securely in a crypto wallet. Any transaction like transferring the currency requires a user authentication with this key. If needed, the crypto wallet can also generate new keys for future transactions. An adversary who gains illegal access to a user's secret key can use the cryptocurrency as they please, e.g. transfer it to herself. The economical value protected by crypto wallets is not technically limited. This makes crypto wallets interesting targets for an attacker.

Crypto wallets can be implemented in software, but often hardware wallets are used, as attacks on hardware wallets require physical access. An adversary is interested in all attacks that allow them to extract the secret key without being the actual owner of the wallet. In addition, they also profit from gaining the ability to generate new keys. Manufacturers of hardware crypto wallets are aware of attacks and especially supply chain attacks. Hence, to encounter these they modify the packaging, add seals, and harden the firmware of their devices [3].

Table 2.2 shows device families of microcontrollers that are commonly used in hardware crypto wallets. A large market share is taken up by microcontrollers from STMicroelectronics. Otherwise, microcontrollers from Microchip, Mediatek, and Ambiq were found to be parts of hardware crypto wallets.

Table 2.2: Hardware crypto wallets and their integrated microcontrollers found in the market analysis.

Vendor	***Device Family***	***Product***
Microchip (former Atmel)	SAM D51	Shiftcrypto BitBox02
STMicroelectronics	STM32F0	Ledger Nano S
	STM32F2	Trezor One
		Shapeshift keepkey
		ProKey Wallet
		ProKey Optimum
	STM32L4	Coinkite Coldcard Mk3
	STM32L4+	Coinkite Coldcard Mk4
	STM32H7	Foundation Passport Batch 2
	STM32WB	Ledger Nano X
		Keevo Model 1
	STM32MP1	NGRAVE ZERO
Mediatek	MT6850	Keystone Essential
Ambiq	Apollo3	Hashwallet

Smart Locks. Nowadays, modern access control systems are often built with the help of smart locks. Smart locks enable the facility management to dynamically update access rights, for example for new employees. This also implies that secret information used by new employees to authenticate themselves must be frequently integrated into the system. If an adversary is able to integrate arbitrary keys into the system they would gain unauthorized access to the facility. In addition, the successful cloning of existing keys would give them the opportunity to intrude while impersonating an authorized person.

The results of the market analysis summarized in Table 2.3 show that microcontrollers from Microchip, STMicroelectronics, Silicon Labs, Infineon, Texas Instruments, or Telink Semiconductor are used within smart locks. In contrast to the other analyzed product categories, there is no major vendor of microcontrollers in this category.

Table 2.3: Smart locks and their integrated microcontrollers found in the market analysis.

Vendor	*Device Family*	*Product*
Microchip (former Atmel)	PIC18F87K22 PIC16F1938 AT91RM3400	Schlage Sense Smart Deadbolt MiLocks Keyless Entry Simon Voss Entry Transponder
STMicroelectronics	STM32L1	August Smart Lock
Silicon Labs	EFM32 Gecko	Salto XS4 One Salto XS4 Mini
Infineon	PSoC 1 PSoC 6	Unknown Product August Smart Lock
Texas Instruments	TI MSP430	Schlage Connected Keypad Lock
Telink Semiconductor	TLSR825X	OASBIKE Bluetooth Smart Door Lock

POS Terminals. Point-of-sale terminals are the standard way of processing payments in the retail industry. Typically they do not store secret information that is of interest for an adversary. Instead, tampering attacks that alter the original functionality of the terminal are a potential threat for these devices. For example, the customer must trust the information shown on the display integrated into a POS terminal, i.e. the amount to be paid. Therefore, any compromise which gains control over the display would enable fraud. This is the case because the amount shown on the display and the amount charged from the card may not match. An attacker could charge more from the card than the customer is aware of. Thus, the customer loses the opportunity to verify the amount to be charged before the transaction is carried out.

The results of the market analysis in Table 2.4 show that mostly microcontrollers from NXP, STMicroelectronics, Maxim Integrated or Nordic Semiconductor are used within POS terminals.

Table 2.4: POS terminals and their integrated microcontrollers found in the market analysis.

Vendor	*Device Family*	*Product*
NXP	Kinetis K22 i.MX258	Square POS Square Reader Nexgo G3N
STMicroelectronics	STM32F0	Square POS Square Reader
Maxim Integrated	MAX32550 DS5240	SumUp Air Sagem Monetèl ETF930S-GEM Sagem Monetèl ETF930S-GEM
Nordic Semiconductor	nRF52	SumUp Air

3 Hardware Attacks

The following chapter lays out attack classes that were deemed most relevant for hardware attacks on microcontrollers. The attack classes were selected from contemporary research on hardware attacks published in journals or blogs and presented on conferences. As stated in Chapter 2, threats posed by IP theft, evil maid adversaries, supply chain attacks, and device theft are the focus of this report. Therefore, the attack classes were chosen accordingly to these scenarios. For each attack class, a concrete attack proposal and evaluation of countermeasures is proposed.

3.1 Control Flow Manipulation Attacks

The control flow of an application determines the order in which instructions are executed. Control flow attacks aim to modify the flow of execution of a program in a way that benefits the adversary. Manipulating the control flow opens various possibilities to undermine the security of a system. For example, it allows the adversary to execute code which requires higher privileges, or to deliberately prevent code execution. In the latter case, the adversary can, for instance, circumvent password checks and thereby gain higher privileges.

A widely known entry gate to mount control flow attacks are buffer overflows, which can corrupt the return address on the stack. This vulnerability allows the adversary to mount a return-oriented programming (ROP) attack and thereby arbitrary code execution on a target, as the return address determines the instructions to be executed after returning from a function call. ROP and related attacks require a vulnerability in the application code that can be exploited by the adversary. While these vulnerabilities are often reported, they can be mitigated by the use of safe programming languages such as Rust. However, at the moment, the Rust support for microcontrollers is mainly community-driven and far from being as comprehensive as the support for the C programming language.

For embedded devices and microcontrollers where physical attacks are possible, control flow manipulation attacks do not require vulnerabilities in the application code. Instead, hardware attacks such as voltage and clock glitching, electromagnetic fault injection (EMFI) and laser fault injection (LFI) can be used to manipulate the control flow of an application.

Fault attacks are used for causing erroneous behavior such as:

- Bypassing the execution of either a specific instruction (single instruction skip), or multiple contiguous instructions (multiple instruction skips)
- Corrupting data on memory or during a bus transfer
- Replacing an instruction with another one, i.e, instruction modification
- Tampering of the program counter

This erroneous behavior is then exploited by the adversaries in order to gain privileges, bypass security measures, extract information, inject malware and more.

Current literature includes numerous examples of successful fault injection attacks on commercial components and devices. Those include, for example, circumventing the memory protection via fault injection for enabling the extraction and subsequently reverse engineering of automotive firmware [65]. Another example is the extraction of a secret key used in an AES core by manipulating the core configuration in [53],[56]. In [84] the authors manage to escape from a TrustZone with the help of fault injection attacks, and in [67] the authors show how fault attacks can be used to generate buffer overflows and demonstrate a complete exploit with a subsequent software attack. Voltage glitching was used in [14] to skip security checks in the bootloader of various microcontrollers and therewith bypass the memory protection. Similarly, in [70],

instruction skips are used to skip bound checks in USB stacks and exploited to perform an out-of-bounds read and to dump credentials.

3.1.1 Countermeasures

In order to protect microcontrollers against control flow manipulation, several software and hardware countermeasures have been proposed [8],[9],[29],[30],[35],[37],[51],[73],[74],[80],[81],[90],[97]. The aim of those countermeasures is the avoidance, detection and/or correction of control flow attacks.

Software-based countermeasures have the advantage that no special hardware is required and retrofitting may be supported via software updates. Nonetheless, it is not always possible to protect against hardware attacks with software-based countermeasures, as faults are introduced directly into the underlying hardware.

According to their point of integration, control flow protection techniques can be divided into four different categories:

- *Source code countermeasures:* At the source code level, software countermeasures are added through a high-level programming language, e.g. the C language. For example, intra-procedural jump attacks could be detected by inserting routine-calls into protected code. Those routines would then check and increment a counter before the next line of code is executed. The disadvantage of this type of countermeasures is that compiler optimizations may have to be disabled in order for the countermeasure to remain present after compilation. Another drawback is that this technique increases the code base and the complexity developers have to deal with.

- *Compiler-based countermeasures:* Unlike source code countermeasures, the protection added by compiler-based countermeasures is guaranteed to be present in binary form, because of the compiler being explicitly aware of its presence. Compiler-based countermeasures make use of the different representations that a program goes through before having a target-specific representation, i.e. binary form. The most convenient representation for the addition of control flow protections is the so called intermediate representation (IR), which is language and architecture independent. However, the underlying platform architecture still has to be considered for a robust and efficient inclusion of countermeasures.

- *Post-compilation countermeasures:* Post-compilation countermeasures refer to the addition of control flow protections to the binary form at the machine language level, i.e. assembly language. This may, for example, involve the addition of redundant instructions, which have the exact same functionality and guarantee the same computational outcome, even when instructions are skipped. The downside of such techniques is the added overhead regarding code size and execution time with respect to compiler- based countermeasures. This is due to countermeasures being added at a point, at which processor registers and memory addresses have already been allocated, thus requiring the addition of intermediate variables and branches to the existing binary. While source code and compiler-based countermeasures are already market-ready and applied in real-world products, post-compilation countermeasures are more a research topic. Due to the lack of automated tools and the manual effort required of developers, their use is not known outside the research community.

- *Hardware-based countermeasures:* Those are countermeasures that involve the modification of the system hardware, for example addition of hardware peripherals or modification of the instruction set. Compared with their software counterparts, hardware-based countermeasures have less impact on the system performance, as they don't demand execution time from the CPU. However, they are expensive to implement and of course cannot be retrofitted into existing platforms via software updates.

After laying out the four general categories based on the point of inclusion, different techniques for hardening software against hardware attacks tampering with the control flow are described. These

techniques may fall into one or more of the four categories introduced above, depending on the implementation layer they are applied on.

- *Classic Loop Hardening:* This technique relies on duplicating the loop counters and exit conditions, such that the condition has to be checked twice before exiting the loop [78]. By comparing the redundant loop counters, it is then possible to detect fault injection attempts.
- *Instruction Redundancy:* This technique involves the execution of critical instructions at least twice, i.e. instruction duplication, in order to provide the same computational result even if instructions are skipped [9]. The redundant instructions must be idempotent, i.e. instructions which can be executed several times while providing the same result as when executed only once. For non-idempotent instructions, an equivalent has to be found, which greatly increases the amount of instructions and often involves the use of additional registers. However, depending on the threat model, the duplication of instructions may still be insufficient, as some hardware attacks are able to skip multiple instructions at once, as demonstrated in [35], where skips of up to 6 consecutive instructions were achieved.
- *Redundant-Encoded Data:* This technique is based on encoding data in order to prevent memory corruption and protect memory transfers [37]. The encoding projects the value of the data into a broader value range, thus facilitating the detection of manipulations.
- *Global Signature Register:* This technique divides memory in basic blocks and performs signature checks when transitioning between them, thus being able to detect control flow deviations between basic blocks [81].
- *Redundant Computation:* This technique relies on computations being executed twice, using different sets of registers, in order to detect instruction skips and register corruption. It was first proposed in [74] and enhanced in various works presented in recent years [29],[30],[81]. The newest implementation of this technique is presented in [90], where the authors state that control flow corruption inevitably leads to data corruption, thus making data corruption detection a mean for detecting control flow corruption.
- *Variable Duplication:* This technique is based on duplicating all variables and their respective read/write operations [80]. Write operations are performed on the original variable and on its duplicate, while read operations are always followed by a comparison of both variables.
- *Function Duplication:* This technique comprises the duplication of a function. The inputs given to both functions are the same, but the results are stored in different variables and compared after both have been executed [8]. A fault attempt is detected if the results of both functions do not match. An enhancement of this technique is the use of equivalent, non-identical duplicate functions, in order to decrease the probability of an attack being successful on both functions.
- *Statement Counters:* This technique refers to the addition of counters and checks between portions of code, e.g., functions, loops, single lines of code [51]. The value of the counters indicate if the statements are executed in the right order, and fault attempts are identified by the presence of unexpected counter values, i.e. a counter increment was skipped.

3.1.2 Attack Proposal and Countermeasure Evaluation

Due to their potential for bypassing security measures and embedded protections such as secure boot and memory bound checks, a demonstration of fault injection attacks causing instruction skips on microcontrollers is part of this report. This section establishes the practical attack and countermeasure evaluation and its relevance to control flow attacks in general. The laboratory demonstration is described in Section 4.2.

For demonstrating those faults, EMFI [70] on STM32L4 microcontrollers seems a promising approach. This is due to STM32 being one of the most relevant families of microcontrollers for all considered product categories, as shown in the market analysis in Chapter 2.

The proposed countermeasure to evaluate is the software-based redundant computation presented in [90], which targets data flow and control flow corruption. As described above, it builds on the assumption that control flow manipulation leads to data corruption. First, works on redundant execution that laid the foundation for [90] such as [29],[30],[74],[81] and pure control flow protection approaches such as [73],[97] are revisited. This is deemed necessary, as all these works were driven by safety-focused research, and therefore their findings can not be directly applied to security. The results of this evaluation are presented in Section 4.2.2.

For future work, further analysis of techniques to protect memory contents are proposed. The authors of [90] rely on ECC memory protection to ensure instruction and memory integrity. If no ECC hardware is available, as is often the case on COTS devices, an attacker could still tamper memory contents. A possible mitigation of this attack vector is the insertion of encoded data during compile time in order to protect data in memory and bus transfers. For such a compiler extension, different coding schemes should be evaluated regarding their detection capabilities and performance overhead. Ideally, at some point in the future, all these efforts can be combined into a framework that hardens software for arbitrary COTS microcontrollers without having any constraints on the underlying hardware. Therefore, retrofitting firmware for existing products is also possible.

3.2 Side-Channel Attacks

Side-channel attacks are passive attacks that exploit information emitted due to the fundamental way an algorithm is implemented. Among others, cryptographic systems have been broken with timing [50], power [49], and electromagnetic [79] side-channel information. For microcontrollers, power and electromagnetic emission are the most critical side-channel information an attacker can exploit. On the one hand, this is due to the absence of data caches on such systems and the availability of constant-time cryptographic algorithms, both of which avoid leakage through the timing side-channel. On the other hand, an attacker can easily gain physical access to a microcontroller to measure power and electromagnetic emission. Power measurements might require physical modifications such as the integration of measurement elements into the power supply path or the removal of decoupling capacitors. This makes them less applicable to evil maid and supply chain scenarios. However, EM measurements are typically non-invasive and can be carried out in these scenarios.

The target of side-channel attacks is typically to obtain a cryptographic secret. If known to the attacker, the secret allows them further attacks such as impersonation of the device, sending of malicious information, or IP theft. This is extremely critical, as microcontrollers often control physical assets. Even if an attacker has to spend a lot of effort regarding time and equipment, the attack might pay off. This was demonstrated in a resounding fashion by Ronen et al. in [83]. The authors could extract the global AES-CCM key for a Philips Hue smart lamp. This key was responsible for verifying the authenticity of OTA firmware updates. Their attack would have allowed an evil adversary to build authentic but malicious firmware images and compromise the entire wireless network of Philips Hue smart lamps. For example, an adversary could have turned off all bulbs in an area such as Paris, where the number of devices exceeds a critical mass.

A demonstration in [72] showed that side-channel attacks are possible in contexts one might not necessarily expect it. O'Flynn and Dewar use on-chip side-channel analysis to measure cryptographic secrets used by the trusted domain of a device. For this the authors run code in the untrusted domain that utilizes the ADC to obtain side-channel measurements. So while most power and electromagnetic side-channel attacks are conducted in a setting where the adversary has access to the device and lab equipment, this is not

necessarily required. The attack is demonstrated on the AES peripheral used within the ARM TrustZone of a Microchip SAML11 microcontroller.

Straightforward power and electromagnetic side-channel attacks on unprotected software implementations running on microcontrollers are frequently reported. The efforts in [49],[66],[71] are only a few of the papers published on this. All these works demonstrate the rather obvious fact that unprotected software must not be run on simple microcontrollers, if an adversary can gain access to secret information over the power or electromagnetic side-channel.

3.2.1 Countermeasures

Since power side-channel information was first used by Kocher et al. [49] in 1998, researchers have investigated different countermeasures and their effectiveness. Among these countermeasures, masking [24],[36],[42] stands out due to efforts in formalizing attacker models and implementations that can withstand them. This allows designers to judge the security level of their implementation. An effort to standardize masking schemes was recently initiated [15].

Masking schemes apply random masks to secret values in order to create secret shares, which are subsequently used for the cryptographic computations. In d-order masking, $d + 1$ shares are used. All but one share can be sampled from a uniform random distribution. These are the so-called random masks. The final share is chosen such that recombining the shares yields the original secret. A related technique to masking is blinding. In contrast to masking, blinding uses algorithmic approaches to mask sensitive values. For RSA and ECC, scalar [76] or exponentiation blinding [50] can be used to avoid leakage of secret exponents.

In contrast to masking which hides the actual values, shuffling [44] hides the order in which values are processed. Operations are scheduled randomly, which hides correlations of side-channel measurements with secret information.

Another approach to defend against power and electromagnetic side-channel attacks is leakage resilient cryptography [63]. Its underlying rationale is to limit the exposure of the secret key such that an attacker has limited capabilities in accumulating information over multiple observations. It has been applied successfully to cryptographic implementations on microcontrollers in [96]. In contrast to masking and shuffling, no randomness is required. Leakage resilient cryptography is therefore also highly relevant for retrofitting countermeasures to microcontrollers. The work conducted in [96] could be extended by considering the latest leakage resilient schemes such as ISAP [31]. However, this report focuses on the application of masking for the countermeasure evaluation proposed in the next section. The continued analysis of leakage resilient cryptography on COTS microcontrollers is left open for future efforts.

Apart from the masking, shuffling, and leakage resilient countermeasures, which include formalized knowledge on an attacker's capabilities, there are several obfuscation approaches often referred to as hiding. An example for a hiding-based approach is the installation of random stall or dummy operations and additional noise sources to reduce the SNR [93]. Since the resilience of these approaches is hard to put into concrete terms they are mostly used in combination with masking or shuffling. Another approach to hide power and electromagnetic side-channel information are dedicated logic encodings, e.g. dual rail logic. Dual rail implementations balance the power consumption regardless of the processed value [85]. However, implementations are costly and vulnerable to attacks based on different propagation times [46].

3.2.2 Attack Proposal and Countermeasure Evaluation

Due to the clear focus on masking by the research community it is also the focus of protecting microcontrollers against side-channel attacks for this report. Masked software implementations would

allow retrofitting side-channel resilience to COTS microcontrollers. It should be emphasized that leakage resilient cryptography is also an established method in protecting against side-channel attacks. Its suitability for retrofitting side-channel protection to microcontrollers featuring generic unprotected AES hardware accelerators has been explored successfully in [96].

There are two approaches to measure the resilience of a masked implementation. The traditional approach is to rely on standardized laboratory work [33],[34],[89]. This requires massive effort and recent works have shown that standardized tests are not guaranteed to find weaknesses [16].

A newer approach was brought forward in [77]. In this work, Prouff and Rivain present a formal proof of the information leakage in masked implementations. Since then, several formal proofs and works that try to bridge the gap between these theoretic proofs and the actual capabilities of an adversary with more realistic models were published [13],[23],[43]. In [20],[25],[64],[94] various authors show that attacks on naive masking implementations are easier than originally thought. Formal approaches could have shown these vulnerabilities without requiring costly evaluations in the laboratory. However, in [12], Battistello et al. show an attack on an implementation that was deemed secure in a formal model. Wrong assumptions on the model made attacking the implementation a lot easier than originally thought. This shows that the maturity of formal approaches must increase so that they can be put to practical use. Nonetheless, there is massive potential in their application to security evaluations.

Research in both formal approaches to masking and attacks on masked implementations has seen a massive attention gain in recent years. Attacks on masked implementations can be divided into two categories. Each category represents a challenge in implementing masking on microcontrollers correctly. Attacks of the respective category exploit shortcomings in meeting this challenge.

Low physical Noise of Microcontrollers. Microcontrollers suffer from low physical noise. Among others, this is due to their nature of sequentially executing one instruction after the other. Thus, an attacker can easily obtain patterns related to secret information from power or electromagnetic side-channel measurement traces. In [16], Bronchain and Standaert attack a hardened AES implementation proposed by the French ANSSI. They show that by means of advanced statistical post-processing, the full key can be recovered with less than 2,000 measurements. Limited noise in the attacked ARM Cortex-M4 (ATSAM4LC4CA) is the primary cause for this security flaw. The authors propose a significantly higher number of shares to counteract this limitation. Additionally, the authors make an interesting point on closed source security evaluations. A preliminary leakage assessment of the implementation by ANSSI did not detect dependencies between traces and secret data in up to 100,000 traces. With their advanced attacks, the authors were able to recover the key with significantly less. In [60], Masure and Strullu extend the work by [16] with new attacks and a public data set of traces. Similar to [16], Udvarhelyi et al. [95] show an attack on a hardened implementation of the lightweight crypto algorithm ISAP [32]. Their target is an ARM Cortex-M0 (STM32F0) that also suffers from low physical noise. In [17], Bronchain and Standaert attack higher-order masked implementations of AES and the lightweight crypto scheme Spook / Clyde [22]. Their target platforms are an ARM Cortex-M0 (STM32F0) and an ARM Cortex-M3 (STM32F100) microcontroller. A 6-share AES implementation on the Cortex-M0 was demonstrated to be broken with 10 traces and a 5-share AES implementation on the Cortex-M3 with 1000 traces. Only an 8-share implementation of Clyde on the Cortex-M0 could not be broken with less than 50,000 traces. They evaluate their findings in a formal model [43] and demonstrate that the model shows the same results if it is instantiated with realistic noise parameters. Gohr et al. [40] evaluate attacks on the same traces of Spook / Clyde with deep learning and achieve a similar efficiency as [17].

Micro-architectural Leakage. In [7], Balasch et al. show an implementation that provides significantly less side-channel resilience than expected because the actual architecture of the implementation and its leakage through micro-architectural transitions are not considered. An example for such a transition is if a share in an internal register (e.g. in the load-store unit) is overwritten with another share. Balasch et al. compare a masked C implementation and a masked assembly implementation that considers architectural leakage of an Atmega163 and an 8051 microcontroller. They demonstrate that the security order of the C

implementation reduces by factor two if a more realistic model that takes transitions into account is considered. Gao et al. [38] show a similar effect for share sliced implementations on an ARM Cortex-M0 (NXP LPC 1114) and ARM Cortex-M3 (NXP LPC1313). In [75], Papagiannopoulos and Veshchikov study the micro-architectural leakage effects of an Atmega163 in more depth and propose a tool to verify AVR assembly code for such effects.

This is only one of many verification tools for masked implementations that have been proposed by researchers in recent years [10],[11],[26],[39],[48],[62],[92],[98]. The works differ in the theoretic model they rely on, the architecture they provide verification for, the architectural leakage effects they consider and whether they can be used for hardware or software masking. A complete systematization of knowledge on design tools for side-channel aware implementations can be found in [21]. More information on theory and application of masking in general can be found in the survey by Covic et al. [27]. Marshall et al. [59] published an interesting study of different micro-architectural leakage effects on different microcontrollers.

Formal verification of masked implementations can potentially provide a high resilience against side-channel attacks without arduous and possibly erroneous laboratory evaluations. However, as of now, the gap between theoretic proofs, realistic noise levels, and realistic leakage due to micro-architectural effects is too wide. For future evaluation, an attack and countermeasure demonstration is proposed to close this gap.

Practical Evaluation. The attack on higher-order implementations of AES by Bronchain and Standaert in [17] is one of the most fatal attacks as they achieve a complete key recovery with very few traces. They connect their findings to a formal model [43] and identify the low amount of noise on STM32F0 and STM32F100 microcontrollers as the primary weakness. However, they do not consider leakages caused by micro-architectural effects. It remains unclear how much harder the attack gets if these leakage effects are taken into account and are mitigated. Further, the authors assume very strong attacker capabilities like profiling with known randomness. This decision is backed by the reasoning from [6], as making worst case assumptions on the attacker provides a reliable and safe way to evaluate countermeasures. If a countermeasure is effective against the worst case attacker (e.g. one that can profile with known randomness), it is guaranteed to hold up against attackers with less capabilities. However, this implies that countermeasures are more expensive in terms of performance than they need to be in the face of realistic adversaries. This leaves room for further evaluations. Recent work by Masure et al. proposes a threat model for so-called scheme-aware attackers, which makes more realistic assumptions on an adversary's capabilities, while still providing a reliable and safe way to evaluate countermeasures [61]. Within this model, adversaries know the implemented scheme and countermeasures from high level source code all the way to the actual machine code and its execution trace (i.e. what happens when), but they do not have access to internal resources such as the randomness generated by the internal random number generator. Future work that evaluates existing attacks and countermeasures against such a threat model would provide valuable insights for retrofitting countermeasures to COTS microcontrollers.

The attack published by Bronchain and Standaert in [17] is a promising starting point for such an evaluation. Their evaluation could be extended by protecting the AES implementation against micro-architectural faults that cause leakage. The *scVerif* tool [11] can be used to detect such faults in the AES code. In [5], Abromeit et al. propose a compiler built on top of the *scVerif* tool that automatically generates correctly masked software. Unfortunately, the compiler was not made publicly available for further evaluation.

The verified implementation is then again attacked with the proposed attack by Bronchain and Standaert. In addition, other attack approaches from [18],[19],[40],[58] can be adapted for a thorough evaluation. This puts the leakages caused by micro-architectural effects into context with the shortcoming of low physical noise. Since the advanced attacks by Bronchain and Standaert exploit randomness reductions and profile with known randomness, a successful key recovery is still very likely.

In this case, more relaxed adversary models and a higher number of shares can be revisited. These evaluations can be conducted both in theory with the formal model from [43], as already outlined by Bronchain and Standaert in their paper, and with further practical evaluations. This investigation is meant

to either back-up or contradict the pessimistic outlook by Bronchain and Standaert which suggests that the effectiveness of masking on devices with low physical noise is limited in the face of strong adversary assumptions. If this can not be contradicted, parallel hardware architectures with a higher level of noise are, if available, the only hope for such devices. In this case, retrofitting side-channel security with software masking is impossible.

A laboratory evaluation of the proposed attack and countermeasure analysis is not part of this report. Instead, this work is left open for future efforts.

3.3 Read-out Protection Bypass Attacks

Microcontrollers usually 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 its flash memory and SRAM, which often contain valuable IP, cryptographic credentials, or other sensitive data.

The component that restricts access to the internals of a microcontroller via the debug interface is called **read-out protection**. A simple implementation of a read-out protection allows only the irreversible disabling of all debug access. However, depending on the application and product life cycle, this is not always desired. For *failure analysis* or *return management* in general, debug access to certain internals is often necessary and desired by the manufacturer of a product. To enable manufacturers to analyze defective products returned by customers, some microcontrollers support multiple protection levels with different access restrictions. For example, access to the internal SRAM is allowed for failure analysis while the non-volatile flash memory, which usually contains the IP and other sensitive data, remains inaccessible via the debug interface. In case the debug interface is not irreversibly disabled, different methods for reactivation exist. Some implementations allow to reactivate the debug interface only after erasing all data on the microcontroller. Others allow password-based or even more sophisticated challenge-response based approaches based on symmetric or even public-key cryptography. Another example where the debug interface cannot be fully disabled and a more complex read-out protection is necessary is when multiple parties are involved in the firmware development process. Since applications become more and more complex, the firmware is sometimes not developed by one company but the product of multiple software components provided by different companies. To protect the IP of all involved parties during the development process, more sophisticated read-out protection techniques are needed in order to protect the individual IPs on a microcontroller.

Since the debug interface provides access to the internals of a microcontroller, attacking this interface is a valuable target for adversaries. The ultimate goal of an adversary is to gain unlimited access and bypass the read-out protection completely. However, depending on the application a microcontroller is used for, partial access might be sufficient, for example only to the internal SRAM or some CPU registers.

Attacks against read-out protection techniques can be split into the two categories **physical** and **non-physical**, depending on whether physical manipulations, temporary or permanent, are used or not.

Non-physical Attacks. Attacks that do not require physical tampering of a device are categorized as non-physical attacks. They are characterized by exploiting vulnerabilities caused by either the implementation of a read-out protection or even a conceptual weakness.

One example for exploiting an implementation flaw in a read-out protection is a race condition found in the STM32F0 microcontroller series from STMicroelectronics [68]. After the power-up of this microcontroller, there is a short time period in which the debug protection is not yet active. If an adversary is able to connect to the debug interface fast enough and perform a read operation before the protection becomes active, the flash memory can be read out. Since the time span the protection is inactive is very

short, the microcontroller needs to be power-cycled for every read operation. With this approach, the entire flash memory, including the containing IP and other sensitive data, can be read out in just a few minutes.

A conceptual weakness of a read-out protection feature was discovered for the nRF52, CKS32F1, and GD32VF1 microcontroller series [47],[69]. These devices are equipped with a read-out protection feature that, when activated, restricts access to the internal flash memory via the debug interface. However, access to the internal SRAM and the CPU remains possible. This approach has the drawback that it allows an adversary to access temporary credentials stored in SRAM or CPU registers. But even more problematic is that this design can be exploited to bypass the read-out protection and gain access to the flash memory. By placing a load instruction into the SRAM and instruct the CPU to execute it, the access restriction to the flash memory can be circumvented. The trick is to use a load instruction such that the CPU loads data from flash memory into the SRAM, which is accessible to the adversary. Another conceptual weakness was discovered in the STM32F1 series [87]. Similar to the previous approach, only direct access to the flash memory via the debug interface is blocked. However, the partial debug access is sufficient to exploit a vulnerability that leaks content of the flash memory through the PC register. This attack allows to extract large parts of the non-volatile flash memory and thus poses a threat to the containing IP and cryptographic secrets.

A common implementation for read-out protection techniques is that a reactivation of the debug interface first performs a mass erase of all internal memories. This ensures that sensitive information stored on a microcontroller is erased before the debug access is available again. The Atmel SAM7xC and Texas Instruments CC2400 series are only two examples for microcontrollers that implement this feature. One characteristic of these two microcontrollers is that reactivating the debug interface erases the flash memory content but not the SRAM [41],[52]. As a consequence, sensitive data in SRAM is accessible to an adversary after reactivating the debug interface.

Another conceptual weakness has been identified in an IP protection technique based on execute-only memory (XOM) which is integrated in a wide range of microcontrollers. This feature allows that multiple parties can sequentially develop and deploy firmware on a microcontroller without being able to copy the firmware of the other parties but being able to use it. In [86], the authors show that a specific execution of the protected code combined with information on the CPU registers and the SRAM is sufficient to recover the code stored in the XOM.

The presented non-physical attacks show how diverse the different implementations of read-out protections are and that even very restrictive access to the internals of a microcontroller can be exploited to gain access to sensitive information. These attacks are very powerful and require no special equipment, only tools every firmware developer uses for daily work.

Physical Attacks. In contrast to the previous attacks, physical attacks tamper with the behavior of the underlying hardware implementations of a read-out protection. A physical attack can further be categorized into *non-invasive*, *semi- invasive*, and *invasive*:

- Non-invasive attacks do not require modifications of the device. Methods in this category are, for example, voltage or clock glitching, and EMFI.
- Semi-invasive attacks need preparation of the device under attack, such as decapsulation. LFI and other optical attacks are examples for this category.
- Invasive attacks also require chip decapsulation but further need preparations such as removal of the passivation layer of the chip. Invasive attacks range from cheaper attacks like probing of the on-chip interconnects to more sophisticated and expensive attacks like *chip surgery* using a focused ion beam (FIB). Due to the high effort and cost of invasive attacks they are not largely represented in attacks on COTS microcontrollers.

Non-invasive attacks to fully bypass the read-out protection have been shown for the nRF52 and EFM32 microcontroller series [54],[55],[57],[88]. The read-out protection 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, for example, an attack on the read-out protection of the STM32L4 microcontroller series [88]. This attack exploits a design flaw, which leads the hardware to 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 fault via EMFI at a specific 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 how the system reacts to the occurrence of an error can be derived from the public documents.

Another non-invasive attack vector that tampers with the read-out protection was discovered by Schink et al. in [88]. As mentioned earlier, some implementations allow to reactivate the debug interface only after performing a mass erase of all memory blocks of a microcontroller, which are usually flash memory and SRAM.

The reactivation process consists of two phases: first, all the memory of the microcontroller is erased. Second, the debug interface is reactivated. The authors show that by injecting a pulse via EMFI into the first phase, the erase procedure can be interrupted and the flash memory content remains intact. The effect is that the erase procedure is skipped while the second phase, the debug interface reactivation, is still executed. After this attack, the adversary has full debug access to the microcontroller and the internal flash memory. So far, this attack has only been demonstrated on the STM32L4 microcontroller, but other devices could also be vulnerable since this kind of debug reactivation can be found in various COTS microcontrollers.

Semi-invasive attacks against read-out protections have been demonstrated on the STM32F1 and GD32F1 microcontroller series [68],[69]. By applying UV light on a decapsulated STM32F1 microcontroller, the protection level stored in flash memory can be downgraded, which reactivates the debug interface. In [69], the authors demonstrate an active and passive semi-invasive attack on the read-out protection of the GD32F1 microcontroller series from Giga Device. In contrast to other devices, its flash memory is located on a separate chip within the package. This design choice enables passive probing attacks by intercepting the inter-die bonding wires between memory and microcontroller. This allows an adversary to gain access to the content of the flash memory despite the enabled read-out protection. Also, an active attack has been demonstrated that manipulates the data transfer between flash memory and microcontroller. The attacker modifies the data signals between both chips in order to change the read-out protection configuration. This allows the read-out protection level to be downgraded and the debug interface to be re-enabled.

The examples of physical attacks show that even if there are no design or implementation flaws, read-out protections can be circumvented. Even fully deactivated debug interfaces can be reactivated by precisely injecting faults or carefully opening the chip package or tamper with the interconnection between chips. Even though not all effects of the physical attacks are fully understood, these attacks are still very effective and reproducible.

3.3.1 Countermeasures

Attacks on the read-out protection of microcontrollers are severe because, if successful, they provide direct access to the internals of the device. In contrast to attacks on the control flow of an application or exploitation of side-channel leakage, the firmware on the microcontroller is usually not involved. As a result, retrofittable countermeasures are not able to prevent these attacks but, depending on the application a microcontroller is used for, may complicate them or even fully stop their effects and provide a sufficient

level of protection. In order to prevent attacks via the debug interface, well and carefully thought-out read-out protection designs and implementations that take hardware attacks into account are necessary.

Exemplary mitigations that can be retrofitted either purely through software or using existing hardware features of microcontrollers are presented. In general, effective countermeasures against attacks on the read-out protection are very specific to the used microcontroller and its application, so this subject can not be covered completely.

Software-based Mitigation. One exemplary attack where software-based mitigation is possible is the exploitation of data leakage through the PC register, which enables read access to the flash memory content [87]. Exploiting this vulnerability does not allow an adversary to read the entire flash memory, small memory regions remain inaccessible. This peculiarity allows a very specific mitigation approach: the flash memory content can be organized in such a way that small amounts of data, for example, cryptographic credentials are placed in memory regions inaccessible for the adversary. Another example is the attack that exploits load instructions to copy the flash memory content into the SRAM that is accessible to the adversary [54],[55]. Here, the entire flash memory can be extracted and as the firmware on the microcontroller is never executed while the attack is carried out, there is no universal countermeasure that can prevent this attack or its effects. However, depending on the application of the microcontroller and the assets to be protected, different countermeasures are possible. In an application where the firmware and its IP needs to be protected, code obfuscation techniques, as presented in [28],[45], are possible to make reusing and reverse engineering of the firmware more complicated. The authors of [28] propose to use a physical unclonable identifier as unique secret that determines the order of instructions. If the firmware is copied to a different device, the order of instructions would be incorrect. The authors of [28],[45] derive reasonable high brute-force complexities. However, it is unclear how much easier an attack gets if the adversary tries more complex re-combination approaches than simple brute-force. Further, the practicality of such approaches without having dedicated hardware available is questionable.

In an application where protecting cryptographic secrets is the ultimate goal, more effective software-based countermeasures may be possible. For example, in case of hardware security tokens, the secrets used for authentication can be stored encrypted in flash memory. During an authentication, the token owner is asked for a password or passphrase to decrypt the secrets on-demand, as suggested in [88].

Hardware-backed Mitigation. An example for a countermeasure that requires special device features and thus depends on the used microcontroller is suggested in [88]. Here, the authors propose to store credentials in a dedicated SRAM that can be accessed via the debug interface only when the read-out protection is fully disabled. Another example for an application and device specific countermeasure is also presented in [88]. In this use case, it is necessary for the adversary to execute the application code such that the secret credentials, stored in the read-out protected flash memory, are transferred into the SRAM where they are accessible via the debug interface. The authors propose a mitigation approach that checks the state of the read-out protection and refuses further operation if the debug interface is not fully disabled. This does not completely prevent the attack but complicates it. The adversary needs to mount an additional and time consuming attack step to skip this check. Depending on the threat model this countermeasure might already provide sufficient protection.

3.3.2 Attack Proposal and Countermeasure Evaluation

Since read-out protection mechanisms differ from manufacturer to manufacturer or even within individual product families, attacks are usually very tailored to a specific implementation. The identification of a potential vulnerability and the subsequent evaluation in a laboratory can take a long time. With the goal to evaluate the security of as much devices as possible, this report focuses the investigation on vulnerabilities that potentially affect multiple device families of a single or even multiple manufacturers.

The attack on the flash erase operation after unlocking the debug interface presented by Schink et al. in [88] was demonstrated on a single microcontroller family but has the potential to affect a wide range of products. The attack is also very severe since it threatens the confidentiality of the entire flash memory content and thus affects both IP and cryptographic secrets. The underlying effect of this attack is unclear but could affect a variety of devices. The attack was originally demonstrated on an STM32L4 from STMicroelectronics but other manufacturers, such as Silicon Labs and Nordic Semiconductor, have microcontrollers with similar features.

For a practical evaluation, first the results of Schink et al. are reproduced on the STM32L4 microcontroller family. The laboratory setup and results are described in Section 4.1. The original publication [88] provides information on the reliability of the attack working on a single device in their dedicated experimental setup. However, reproducibility on other STM32L4 microcontrollers and the influence of setup parameters, for example the position of the injection coil, remain unclear. Since the adversary deliberately triggers a mass erase of the flash memory and thereby potentially erases all IP and credentials on the device, the assets are irreversibly lost if the disruption is not successful. After the results on the STM32L4 are reproduced, the attack is investigated on other microcontrollers from STMicroelectronics. The idea behind this approach is that the hardware module that is used for the unlocking and erase procedure is the same among the microcontroller families. This report also analyses the effect of the attack on devices from other manufacturers, preference given to vendors identified during the market analysis. In the original publication, EMFI is used to disrupt the erasing procedure. This was adapted for the laboratory evaluation in this report (see Section 4.1). Future work should investigate whether voltage glitching is also suitable for this attack. Voltage glitching would reduce the attack costs and might be more robust and reliable since only precise timing is required. The positioning of the injection coil, which usually has to be done very precisely, would no longer be necessary.

Since software-based countermeasures to prevent attacks on the read-out protection are very application and device dependent, there is no general countermeasure or mitigation technique. Instead, countermeasures must be tailored to each affected device and product built on top of it. For example, there might be affected microcontrollers with special peripherals or features to irrevocably prevent the debug interface from being unlocked. Therefore, the laboratory evaluation in Section 4.1 does not include an evaluation of countermeasures.

4 Laboratory Evaluation

Based on the analysis of hardware attacks in Chapter 3, two of the three proposed attacks were chosen for practical investigation. In the following chapter, the results of this evaluation are presented. In Section 4.1, the evaluation results of an attack that targets a microcontroller's read-out protection are shown. For this attack, the repeatability on different devices is extremely important. Therefore, the laboratory setup and its building blocks, which are used to achieve a high accuracy, are described in detail. In Section 4.2, the results of evaluating a compiler-based countermeasure against control flow manipulation attacks are shown. For this attack, the repeatability is less of an issue. A similar setup as for the read-out protection attack can be used.

4.1 Flash Erase Suppression

In Section 3.3, a non-invasive attack demonstrated by Schink et al. that targets the read-out protection of STM32L4 microcontrollers [88] was described. The attack is presented as part of a comprehensive analysis of open source hardware security tokens. It is only discussed in this context and without a general in-depth analysis. However, the attack itself is also of general interest because, if successful, it provides unrestricted access to all secrets stored on the microcontroller's internal flash memory. Further, other microcontrollers of the STM32 family or even microcontrollers of other manufacturers may be susceptible to this attack vector. For that reason, this attack is reproduced and further details on its practicality are provided in this report.

The STM32L4 microcontroller family features three read-out protection levels. In level 0, the read-out protection is disabled. In level 1, debug access is possible but with the restriction that any read attempt through the debug interface to the flash memory is blocked. Level 2 fully deactivates the debug interface. The attack presented by Schink et al. targets level 1. In the following, it is assumed that the microcontroller is in level 1. This assumption can be made as the authors of [88] also demonstrate how to downgrade from level 2 to 1. Thus a reactivation of the debug interface can be achieved. The attack that is reproduced here leverages that the read-out protection can always be disabled by changing the level to 0. Before the read-out protection is disabled, the device performs a mass erase in order to ensure that all sensitive content stored in the flash memory is erased. As described by the authors, the unlocking process consists of two phases. First, the flash content is erased and afterwards the read-out protection is deactivated. Schink et al. discovered that by injecting an electromagnetic fault into the first phase, the flash erase operation can be suppressed while the second phase is still executed. The attacker obtains a microcontroller with disabled read-out protection (level 0) and intact flash memory contents.

The downside of this attack vector is that, if the attack does not succeed, the adversary loses the flash memory content irretrievably. If the adversary attempts to steal a specific device individual secret, the attack irrevocably failed. If, however, IP or other information that is redundant on all devices of a product series is the target of the attack, an adversary has as many tries as devices in their possession. In both scenarios, the reliability of this attack vector is important. Schink et al. state a success rate of 100 % for this attack. However, they do not provide the exact position of the fault injection coil but only a schematic of the chip package and the coil position. For EMFI, such parameters may have a high impact on the success rate of the attack. Small changes from the actual position can lead to failure of the attack. With the laboratory evaluation in this section, detailed findings on the necessary precision for this attack vector are reported for the first time. Further, it is evaluated whether other microcontrollers are susceptible to this attack vector.

4.1.1 Laboratory setup and Results

In order to reproduce the results of Schink et al., a similar laboratory setup with the same fault injection equipment, namely the ChipSHOUTER from NewAE and the included 4 mm CW injection coil, is used. All

experiments are performed on the STM32L422KBT, the same microcontroller that was used in the original publication. Figure 4.1 depicts a block diagram of the laboratory setup used for the evaluation within this report.

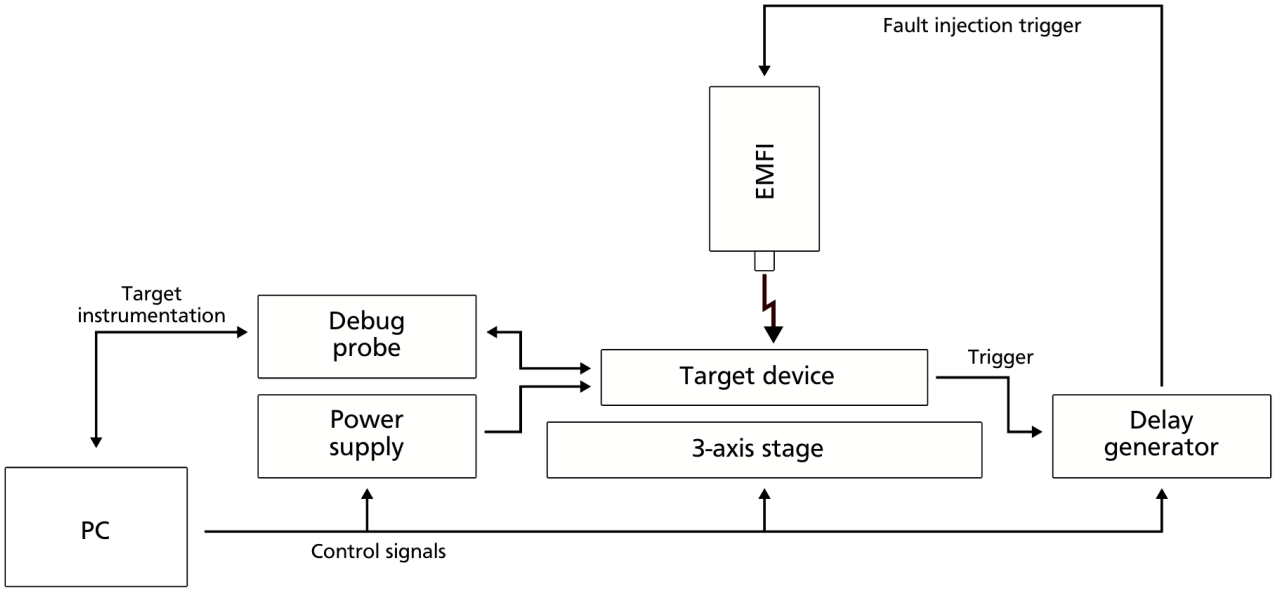


Figure 4.1: Block diagram of the laboratory setup to reproduce the flash erase suppression attack.

The target microcontroller is mounted on a 3-axis position stage. With this, the injection coil can be positioned precisely on the device and an automated grid scan can be performed. To ensure that the target device is always in a defined state, a programmable power supply to power cycle the microcontroller is used. Power-cycling is applied before each measurement or for recovery, if the faulted microcontroller malfunctions. With the delay generator, it is ensured that the fault is injected at the right point in time. The generation of the trigger signal is explained in the next paragraph. A host PC controls the entire setup and instruments the target microcontroller via a debug probe. The laboratory setup is depicted in Figure 4.2.

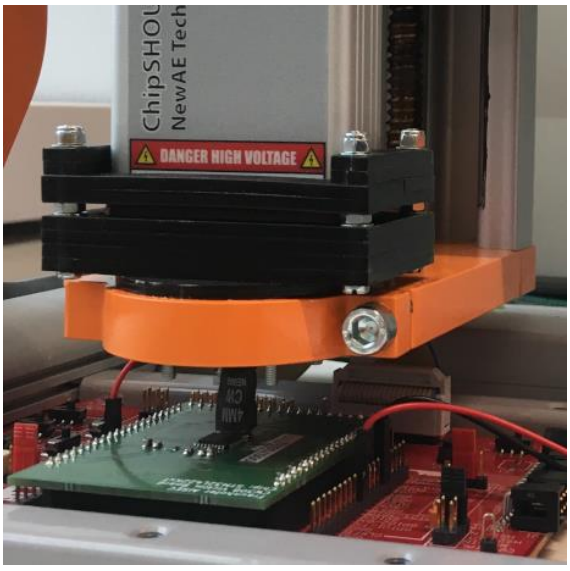


Figure 4.2: Picture of the laboratory setup to reproduce the flash erase suppression attack.

To determine the right point in time for injecting the fault and suppressing the flash erase operation, Schink et al. use a measurement of the EM emanation during the RDP downgrade from level 1 to 0. This measurement is reproduced with the setup depicted in Figure 4.2. In order to generate a suitable trigger signal for the measurement of the EM emanation and the fault injection later on, a special firmware that is executed in the SRAM of the microcontroller is developed. If the device is in RDP level 1, this is within the adversary's capabilities. The firmware raises a trigger signal on a GPIO pin of the microcontroller and subsequently starts the RDP unlock process.

In Figure 4.3, the EM emanation during the RDP unlock operation is depicted. The red line indicates the trigger signal of the target device generated by the firmware. One can clearly identify the two blocks, a small block followed by a larger one, as depicted in the original publication. Even though the timing of the measurement is different, it is sufficient to calibrate the setup to reproduce the fault injection attack. According to Schink et al. the success rate is 100 % when the fault is injected at the beginning of the second block. From Figure 4.3 it can be seen that the fault needs to be injected at around 25 µs. This means that the trigger signal provided by the firmware must be delayed 20 µs (see the duration in Figure 4.3 between the red line and the start of the second distinct block in the signal). As mentioned before, a delay generator was integrated into the setup for this purpose.

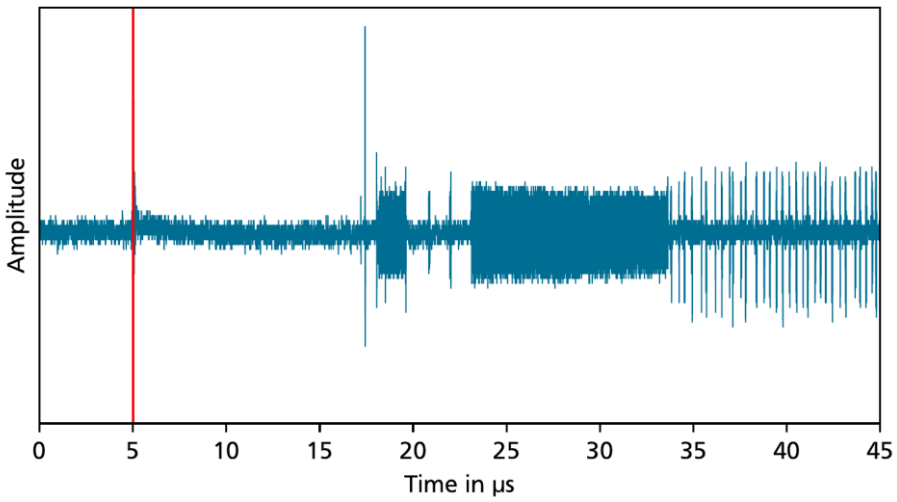


Figure 4.3: EM emanation trace of the STM32L422KBT during an RDP downgrade to level 0.

With the calibrated time for the fault injection, all necessary parameters provided by Schink et al. are present and the evaluation can be started. Schink et al. did not provide information such as the coil voltage and the pulse width. For the laboratory evaluation in this report, a coil voltage of 500 V and a pulse width of 80 ns are used. The coil is placed directly on the chip package. The evaluation is started by manually positioning the injection coil around the area used in the original publication. This approach was supposed to lead to quick results but was not successful. In order to evaluate the susceptibility of the target device, a grid scan of the chip is performed. The chip has an LQFP-32 package with a dimension of 7 x 7 mm. For the grid scan, a step size of 0.1 mm, resulting in 4900 distinct locations for fault injection on the entire chip, was used.

In Figure 4.4 the heatmap of the success rate to suppress the flash erase during an RDP unlock on the STM32L422KBT is depicted. The red dot in the top right corner indicates the package marking and thus the orientation of the chip. The success rate is calculated based on 40 measurements per position for the entire chip. Each measurement was performed individually, meaning that, before each individual fault injection measurement, the 3-axis stage was moved to the home position. This approach includes potential repeatability inaccuracies of the 3-axis stage into the results.

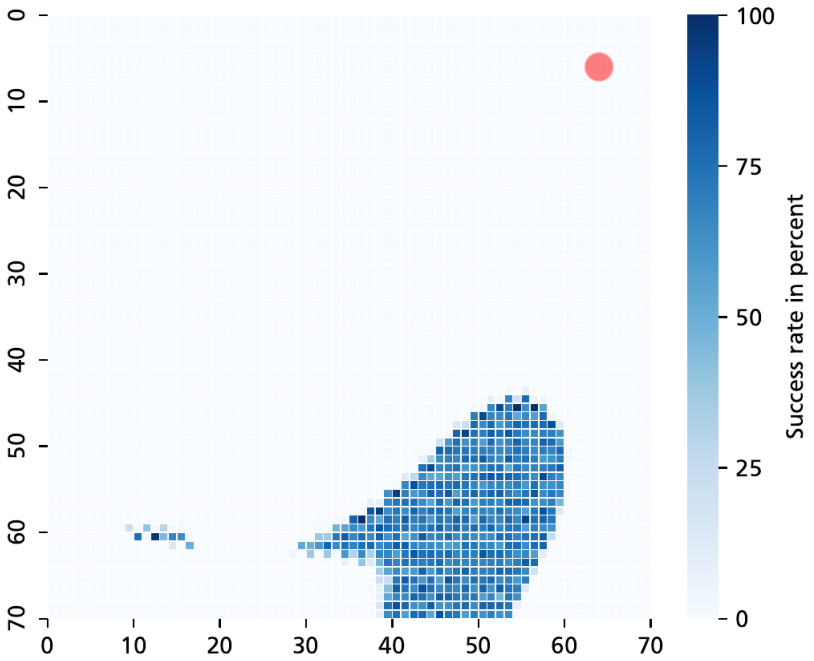


Figure 4.4: Success rate to suppress the flash erase operation on the STM32L422KBT.

With the grid scan over 400 locations with a success rate of 50 %, 12 locations with a success rate of over 90 % and three locations with a success rate of 100 % were identified. One of these three locations is in the small area on the bottom left corner of the chip. This location matches the schematic drawing of the original paper from Schink et al. and may be the position they used. With the same parameters, the attack was evaluated on multiple STM32L422KBT microcontrollers. When neglecting small differences which stem from inaccuracies due to, for example manual soldering of the chips, the same results were achieved.

To conclude the evaluation, the question whether other microcontrollers are susceptible to this attack vector as well, is answered. As for the STM32L422KBT, first, the EM emanation during the RDP downgrade to level 0 is analyzed, to determine the delay for the fault injection. For the investigated microcontrollers, the flash erase operation could be suppressed as well. The preliminary results indicate that other microcontrollers are affected as well. However, more working time in the laboratory is necessary to generate reliable data for these devices. For this reason, we do not list the other affected devices in this document. A separate document with final and detailed information will be published after the affected manufacturers are informed as part of a coordinated vulnerability disclosure process.

4.1.2 Conclusion and Future Work

The laboratory evaluation of the flash erase suppression attack yields the following findings:

- The attack can be reproduced on the STM32L422KBT.
- The results so far show that this attack is very sensitive to the coil position.
- Other microcontrollers are also susceptible to this attack vector.

The laboratory evaluation shows that the attack presented by Schink et al. can be mounted on different microcontrollers. Since a 100 % success rate with the current parameter set is hard to achieve, the attack may not be reliable enough for targets with unique device secrets. Small variations in the position of the injection coil can lead to a failure of the attack and an irretrievable loss of the assets. However, the results are already sufficient for attacks where multiple target devices are available, for example, devices with shared secrets such as common attestation keys or firmware containing IP. As the area with a 50 % success rate is quite large and contiguous, a coil can easily be positioned in this area.

This attack vector needs to be considered and integrated in the threat model of existing and future products using COTS microcontrollers. Further, more investigation needs to be done since the parameter space is large and other parameters, such as the coil distance to the target or the coil type, may lead to higher success rates. Since this attack cannot be prevented completely, application specific mitigation techniques are necessary. For example, sensitive content stored in the flash memory should be encrypted and only decrypted on-the-fly when needed. This countermeasure only makes sense if the cryptographic secret used for decryption is derived from user input. Otherwise, the cryptographic key is as easy to extract as the contents in flash memory it protects. However, such a user interaction cannot be implemented for all applications. The same holds for obfuscation-based mitigation techniques discussed in Section 3.3.

4.2 Compiler-based Fault Injection Protection

In Section 3.1, the threat of physical control flow attacks for microcontrollers was introduced. As a concrete example, the circumvention of the secure boot process on a microcontroller is shown. This attack allows an adversary to execute malicious code on the device. This section demonstrates this on MCUboot, a secure boot implementation for microcontrollers that is partially hardened against physical fault attacks. Compiler-based countermeasures can be used to automatically harden the complete code of such boot frameworks. Here, the adoption of such countermeasures to mitigate the attack is described. The obvious benefit of this countermeasure is that it can easily cover the complete application. This mitigates both straight-forward attacks such as skipping the verification procedure and algorithmic attacks on the cryptographic implementation as exploited for the attack described in this section.

4.2.1 Attacking MCUboot's RSA implementation

MCUboot is a secure boot implementation for 32-bit microcontrollers. It uses an abstraction from the underlying hardware and borrows needed functionalities from the operating system, which is booted by it. In this example, the Zepyhr operating system is used in combination with MCUboot.

To protect against physical hardware attacks, MCUboot applies general hardening against fault injection. Automated testing is available to evaluate the measures against the instruction skip fault model. Within these tests, the compiled binary is checked for the boot process's outcome if several specific instructions are not executed. This ensures hardening against fault attacks, which target a skip of these instructions. Further measures to protect against data corruption can be enabled during compile time. For instance, masking of register values or adding random delays to induce jitter and reduce the likelihood of a precise fault attack. All measures are applied in the source code, and thus have to pass the compiler. Depending on the chosen compiler flags and enabled optimizations it may result in a weaker binary than anticipated in the source code. ARM evaluated this behavior and applied fixes to the compiler which avoid that it removes protective measures [1].

The discussed source code hardenings are not applied over the full implementation of MCUboot and its third-party libraries. The developers' rationale was to harden only the critical control flow, where a single fault would lead to compromise. At the same time they tried to keep the performance and memory overhead as small as possible. In Figure 4.5, the protective regions are symbolically shown. As the cryptographic

implementation was not hardened, its resilience against fault injection is evaluated. More precisely, the implementation of the RSA signature verification, which is used to verify the authenticity of a MCUboot boot image, is analyzed. It is unclear, whether this cryptographic implementation was not hardened as it would be too much effort, too much impact in the performance or, probably most unlikely, whether it was deemed uncritical.

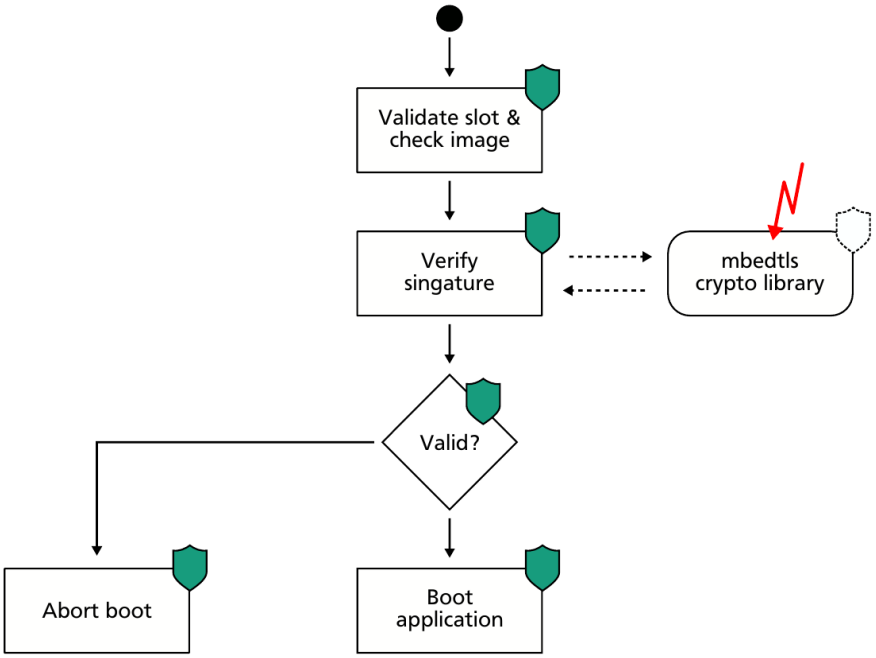


Figure 4.5: Regions of FI hardening in MCUboot.

Recap of RSA algorithm. The RSA algorithm is an asymmetric cryptographic algorithm which can be used to create and verify digital signatures. It is based on the math of modular exponentiation. Its security argument relies on the hardness of decomposing a product into its two large prime factors. An RSA key pair consists of:

- a *private key* with the private exponent d, and public modulus n, and
- a *public key* with the public exponent e, and public modulus n.

The following needs to hold for the private exponent d and the public exponent e:

$$e * d \left(mod\ \varphi(n)\right) \equiv 1 \quad (4.1)$$

The private key is used to generate a digital signature s of a message *msg* with the help of a cryptographic hash function *hash*(·). First, the digest of the message is calculated with $h = hash(msg)$, where $h \in [0, n)$. Second, the signature is generated by:

$$s = h^d (mod\ n) \quad (4.2)$$

If a verifier receives s and *msg*, and is in possession of the corresponding public key, they can verify the signature. First, the digest h is calculated, analogous to signing. Second, the signature is used to calculate a digest candidate h' by:

$$h' = s^e (mod\ n) \quad (4.3)$$

If the signature is correct, h and h' are equal:

$$h' = s^e (mod\ n) = (h^d)^e (mod\ n) = h \quad (4.4)$$

Please note, that for a complete RSA signature verification, padding is required, which is neglected in this example.

Attack on RSA implementation. In MCUboot, the cryptographic operations are provided by mbedTLS. For this work, the implementation of the modular exponentiation *mod n* is most relevant. The mbedTLS library efficiently computes it by applying the *square-and-multiply* algorithm. Within this algorithm each bit of the exponent is evaluated individually starting from the leftmost bit (most significant bit) to the rightmost (least significant bit). On the implementation level, this is a loop, which seems suitable for the evaluation of fault injection. For example, a fault can be applied to force an early return of the algorithm. In Listing 4.1 simplified pseudo-code of the modular exponentiation implementation is listed. Additionally, the targeted effect of the fault is added and highlighted in red.

```
function exponentiation(base, exponent)
  result = 1
  for bit in exponent:
    result *= result
    if bit == 1
      result *= base
      return result
  return result
```

Listing 4.1: Simplified square-and-multiply algorithm. The effect of the fault is included in the pseudo-code and highlighted in red.

Depending on the implementation, an early return would, in the best case, result in a public exponent $e = 1$ during the signature verification. As a consequence, it follows that $d = 1$ is assumed valid as well, as Equation (4.1) holds. This allows an attacker to generate a malicious key pair with $d_a = 1$, $e_a = 1$, and the public exponent n of the targeted key pair. From the malicious key pair, a valid signature s_a for a message msg_a can be created. By applying the fault, the verifier can be tricked into accepting msg_a and s_a as valid, even if they are not valid for the actual key pair with exponents d and e. To attack MCUboot, a malicious application is signed using a malicious key pair with the private exponent $d_a = 1$. The MCUboot image is in possession of a validly generated public key consisting of e and n. As described above, a fault is used to enforce $e = e_a = 1$ and therefore booting the malicious application.

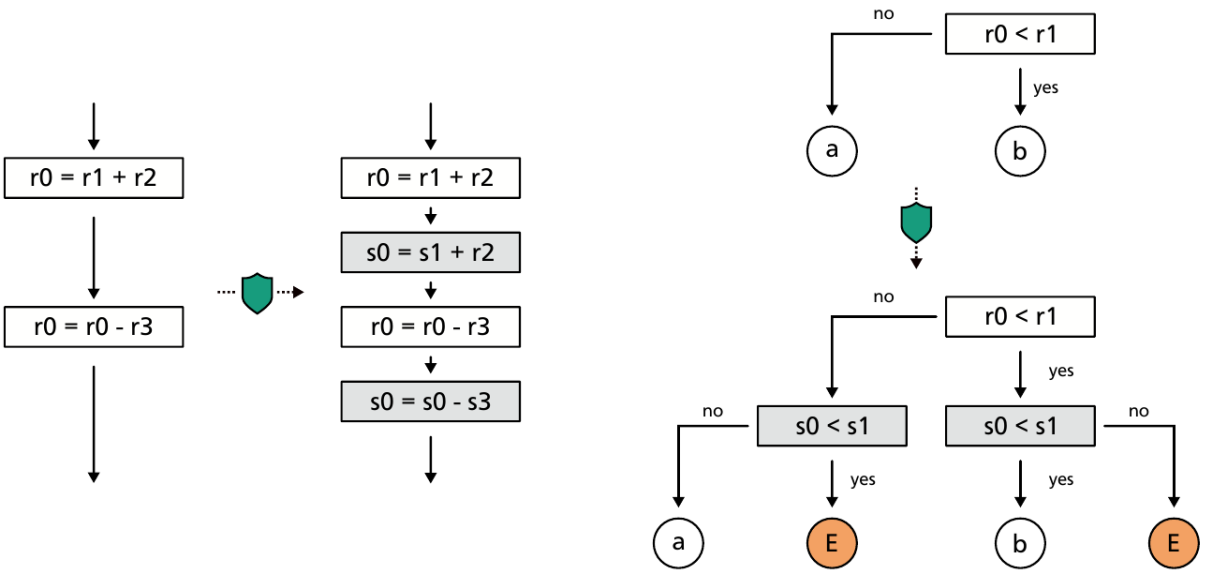
4.2.2 Compiler-based hardening of MCUboot

As stated in Section 3.1, redundant computation as implemented in [90] is deemed a promising approach to mitigate corruption through fault attacks. In [91], the approach from [90] and other works [29],[30],[73],[74],[81],[97] were integrated into the LLVM compiler for RISC-V architectures. Therefore, the countermeasure can be integrated at compile time, arguably the most convenient and safest method for software developers. This work is adapted for the evaluation in this report.

A short overview of software hardening techniques. Over the past years, researchers have tried to achieve good trade-offs between error resilience and performance overhead of software hardening techniques. It is important to note that these works were driven by the safety and not the security domain. In simple words, approaches to counter control flow corruption [73],[97] continually maintain a *signature register* which is updated for each block of instructions. For every transition between blocks, a *signature check* is performed,

which detects illegal transitions. Approaches to counter data flow corruption [29][30],[74],[81],[90] build on redundant execution. The available CPU registers are divided into a set of *primary registers* and a set of *shadow registers*. Every instruction operating on primary registers is duplicated with the respective shadow registers. Any mismatch within a register pair implies illegal tampering or operation under hazardous environmental conditions. The listed works differ in their handling of memory accesses and instructions that influence the program flow, therefore they also differ in their respective performance and code size overhead. Most notably, *NZDC + NEMESIS* [29],[30] is the state-of-the-art in classical data flow protection. In [90], the authors show that a dynamic mapping between primary and shadow registers allows to detect a high percentage of control flow corruptions. All these works assume an ECC memory. As stated in Section 3.1, protecting data with simple encodings is proposed for further evaluations if such a memory is not available.

The program flow in Figure 4.6 demonstrates how the redundant execution works on assembly level. The duplication of basic arithmetic instructions is straightforward. Conditional branches are protected by doing separate branches on the condition applied to the primary registers and the shadow registers. In total, there are four possible outcomes for a condition. However, only two outcomes are legal and allow the program to continue.



(a) Arithmetic (b) Conditional

Figure 4.6: *Examples for redundant execution of arithmetic (a) and conditional (b) code.*

Implementation. Internally, the compiler presented in [91] uses the *NZDC + NEMESIS* approach shown in [29] for duplication and then applies the transformations of [90] on top of it. Therefore, [29] is the focus for laboratory evaluations within this report. In [91], a so called backend function pass is added to the LLVM compiler to harden code. The backend is responsible for translating LLVM's intermediate representation into machine code. Inserting instructions for redundant execution in the last backend function pass ensures that all memory accesses and branches are protected, as the compiler may reorder blocks and insert additional load/stores in early backend function passes. Unfortunately, it also means that the code by Sharif et al. [91] highly depends on the target machine. In their work, only RISC-V architectures are supported. The market analysis in Chapter 2 showed that no RISC-V microcontrollers are present in the relevant product categories. In contrast, ARM Cortex-M microcontrollers are used a lot throughout the different products.

Therefore, the work from [91] was ported for the ARM architecture, more precisely the Thumb ISA, focusing efforts mainly on the implementation of *NZDC + NEMESIS* [29]. Due to differences in the RISC-V and Thumb instruction sets, this is not straightforward. The proof of concept implementation created for this report

managed to automatically protect the relevant methods of mbedTLS (see next paragraph) with the concept of [29]. Due to time constraints, the extension of [90] and other countermeasures could not be realized. To demonstrate the cost of using the compiler extension, a short evaluation of two simple benchmark programs and the relevant mbedTLS method was conducted. Table 4.1 shows that the code size increases by factor 2 to 5 and the cycle count by factor 3 to 4. The exact overhead depends heavily on the application. As shown in Figure 4.6, a sequence of arithmetic instructions can be protected with straightforward duplication, whereas branches require more sophisticated approaches which imply a higher overhead. The same holds for memory accesses and function calls. It is expected that the compiler extension can be optimized to lower the overhead, however, a vendor will always have to accept code size and cycle count overheads somewhere between factor 2 to 3. It should also be emphasized that further modifications are necessary to improve the usability of the proof of concept implementation and to extend it to other ISAs.

Table 4.1: Overhead comparison of the implemented redundant execution countermeasure for different benchmark programs, compiled for the ARM Cortex-M4 (-O2, clang version 13.0.0).

Program	*Protection Level*	*Code Size*	*Cycle Count*
CRC[1]	Without protection	1,344 Byte	370
	With protection	3,448 Byte	860
fibonacci[2]	Without protection	92 Byte	440
	With protection	516 Byte	1,690
mbedTLS modular exponentiation[3]	Without protection	1,962 Byte	11,870,350
	With protection	7,006 Byte	42,646,750

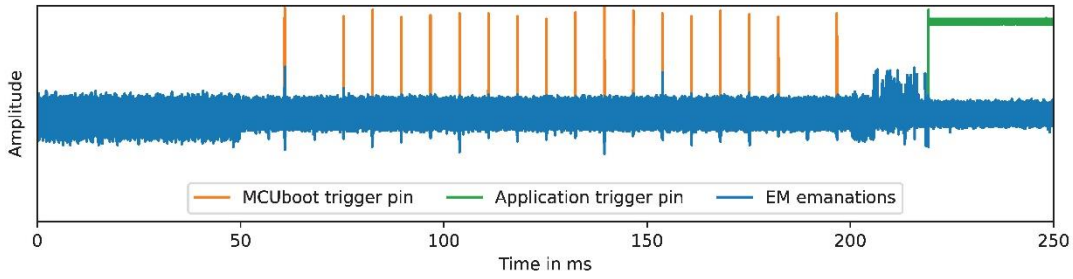
Mitigating the RSA attack. The previous section showed how skipping a single instruction is sufficient to enforce a public exponent $e = e_a = 1$ during signature verification. When analyzing the branch protection in Figure 4.6, it is obvious that one single instruction skip is no longer sufficient for a successful attack. Either two precise skips of single instructions or one skip of at least three instructions are required. If an adversary can realize such attacks, even the extension from [29] to [90] offers no sufficient protection. It only makes a fault detection more likely in the realistic case that the number of skipped instructions cannot be controlled perfectly. However, for perfect protection against such attacks, an integration of control flow corruption countermeasures like [73],[97] would be necessary. Also, software countermeasures are not suitable to fully protect against attacks which introduce faults directly into the PC register.

4.2.3 Laboratory Setup and Results

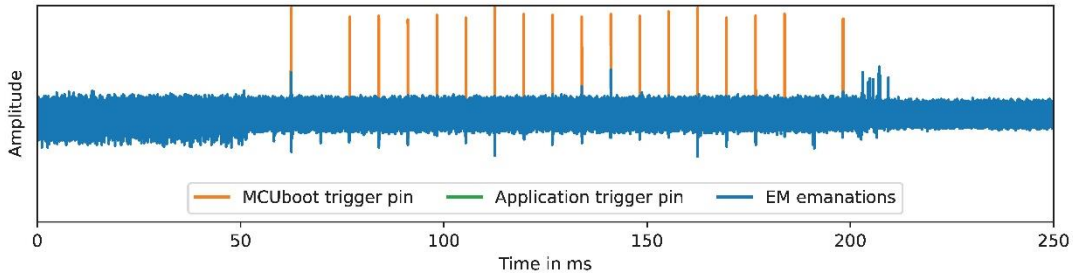
The attack is evaluated on an electromagnetic fault injection setup. A setup similar to the one shown in Figure 4.1 can be used. The only difference is that an oscilloscope is necessary to analyze the electromagnetic emanation and create a trigger dependent on its waveform. The target device is an ARM Cortex-M4 microcontroller running MCUboot with a Zephyr application. Before the actual attack, different parameters to achieve an instruction skip on the chip were investigated. For this, a grid scan similar to the one described in Section 4.1.1 but with a higher resolution of 0.01 mm was performed. An area of 0.14 x 0.04 mm where an instruction skip could be introduced repeatedly was identified. In some subregions of this area, a probability higher than 90 % was achieved. Such a subregion of 9 locations in an area of 0.03 x 0.03 mm was used for the investigation of the MCUboot attack. The used pulse was generated with a delay generator with a fixed pulse width of 100 ns.

[1] https://vhosts.eecs.umich.edu/mibench/
[2] https://github.com/fharookshaik/fibonacci-series
[3] https://github.com/Mbed-TLS/mbedtls

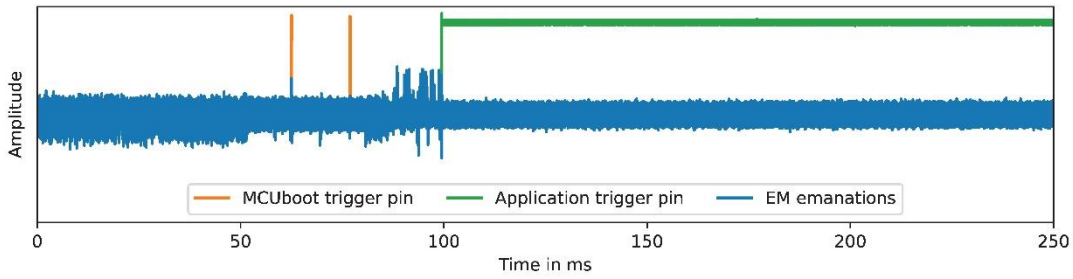
For the MCUboot attack, the malicious application is signed as described above. To identify the current state of MCUboot, the electromagnetic emanation is recorded and evaluated in real-time with the oscilloscope. Figure 4.7 shows the EM traces for the three different scenarios of a valid image in Figure 4.7 a), an invalid image in Figure 4.7 b), and an invalid image with a simulated fault in Figure 4.7 c). The waveform of the modular exponentiation algorithm can be distinguished from other operations. By comparison of Figure 4.7 c) which includes a simulated fault in its implementation the early return and thus the shorted execution time is visible. Further, the handling of each bit is highlighted as the MCUboot implementation was modified to toggle an output. For the attack the side-channel information, without a toggling output, can be used as a trigger to precisely inject the fault into the modular exponentiation. This allows to minimize the jitter of imperfect attacks due to inserting the fault too early or too late. The laboratory evaluation showed that a success rate for the attacker of one third can be achieved with such a setup. Hence, one out of three fault attacks results in MCUboot accepting the malicious signature and booting the malicious application. This demonstrates that the secure boot protection feature can be circumvented, and any application can be booted.



(a) Valid application verified and booted by MCUboot.



(b) Invalid application detected and not booted by MCUboot.



(c) Invalid application image verified because of an injected fault and booted by MCUboot.

Figure 4.7: EM emanation trace of the ARM Cortex-M4 during MCUboot with RSA signature verification.

If MCUboot is hardened with compiler-based redundant execution as described above, the probability of a successful attack reduces drastically. For this evaluation, an artificial trigger is used, i.e. the target pulls up a trigger pin when it is at the critical operation, to model the absolute best case for an attacker. As discussed above, MCUboot was compiled with the data flow protection scheme *NZDC + NEMESIS* [29],[30]. Realizing

advanced approaches for data flow protection [90], control flow protection schemes [73],[97], and the encoded memory transfers proposed in this report are left open for future work. For over 10,000 tries, it was observed that the attacker could boot a malicious software six times, despite the protection by the compiler. The only possible explanation for this behavior is that the introduced fault does not comply with a single instruction skip fault model. Therefore, we fall back on two more general fault models for this analysis.

In the first model, the adversary skips an arbitrary number of instructions, but stays in the method performing the sliding window exponentiation. It was verified that the six experiments for which the adversary could boot malicious software complied with this model, as the sliding window exponentiation went through the expected return path. An investigation of the sliding window exponentiation's assembly code yielded, that the attacker needs to skip at least thirteen instructions. Additionally, the attacker needs to jump to a point, where the primary and shadow registers are in sync, i.e. not jump in between a shadow and a primary instruction. While such fault effects are rare, the laboratory evaluation demonstrates that they could be a realistic threat to implementations. However, control flow schemes such as [73],[97] mitigate this effect completely, as all illegal jumps between basic blocks are detected. As described earlier in this report, the proposal of [90] would provide no perfect protection against this fault model. Therefore, an adoption of control flow protection schemes for compilers such as the one used in this report is highly recommended.

The second model includes all cases where the fault manipulates the program counter such that the sliding window exponentiation method is left directly. This behavior was observed for 50 experiments. It never led to a valid boot, due to the program counter hanging at invalid locations without valid instructions. However, there exists a probability that the program counter points to the application after the fault. The attacker can also design the application's memory region as a slide, where dummy instructions are utilized such that the program counter slides to an entry point. It should be emphasized, that software-based countermeasures cannot offer a complete mitigation of direct program counter manipulations. The conducted experiments show that the probability that a fault leads to a jump into a region with code controlled by the adversary is not negligible. To encounter such attacks distinct hardware countermeasures need to be applied. This is an open task and requires further research. It is left open for future work.

For the majority of observed faults (~ 99.4%), the attack ended in an error handler, either the system's own error handler or one introduced by the compiler for detection of faults.

4.2.4 Conclusion and Future Work

The laboratory evaluation of control-flow attacks against secure boot yields two core findings:

1. If a fault attacker with the ability to skip single instructions is within the threat model, it is not sufficient to only harden the "high-level" control flow.
2. Software-based fault detection countermeasures can be applied conveniently during compile time and raise the bar for a successful attack significantly.

For both points there is potential as well as the necessity for future research.

Generic fault vulnerability analysis. Enforcing a public exponent $e = e_a = 1$ during RSA's signature verification is only one example of exploiting behavior that occurs if a cryptographic algorithm is brought to act outside of its standardized value range. Depending on the use case, other attack points for RSA can be expected to arise. Also, other cryptographic standards such as ECC can be expected to be vulnerable to similar attacks. The safest way to mitigate all possible attacks is to protect every piece of code. However, this implies a large overhead of code size and execution time. If a routine is hardened only partially, the developer needs support of analysis/ verification tools to make a reasonable statement on the security of the implementation.

Improved compiler-based hardening. The developed proof of concept implementation of a data flow protection scheme for ARM microcontrollers in the LLVM compiler demonstrates how valuable such a tool

can be to conveniently retrofit protection against physical attacks. As mentioned earlier in this report, signature checks and encoded memory transfers could further enhance this compiler. Additionally, there is a significant value in making the compiler feature less dependent on the target platform. Otherwise, the compiler needs to be almost completely rewritten for every instruction set architecture, as done for this report. While some protections can not be applied until the end of the compilation process, the foundation for protection as well as necessary data dependencies can be created in a generic intermediate representation. For such an implementation it is necessary to understand the differences of instruction set architectures regarding features that modify the program flow (e.g. architectures with and without conditional flag registers). An extensive study on this is still missing in literature. First observations were collected during the investigations within this report.

5 Conclusion

Microcontrollers constitute the backbone of a smart and connected world. Resilience against hardware attacks is required for numerous applications to protect critical infrastructure, intellectual property, and sensitive information. While this fact is hardly ever challenged, different stakeholders draw different conclusions from it. On the one side, vendors of COTS microcontrollers view hardware attacks as out of scope for their devices. On the other side, product vendors build security-related products from these microcontrollers. The market analysis in this report has shown exactly that for hardware security tokens, hardware crypto wallets, smart locks, and POS terminals. Products that fall into one of those categories are often based on COTS microcontrollers. Only recently, countermeasures against hardware attacks have been considered for integration by some vendors. According to the conducted market analysis, microcontrollers that have dedicated countermeasures in place and were certified to withstand certain attacks identified in this report [2] do not play a role for manufacturers of the listed product categories.

Attacks on COTS microcontrollers are frequently shown in state-of-the-art hardware security research. Within this report, two of three reviewed attack classes were evaluated in practice: control flow manipulation, and read-out protection bypass attacks. To demonstrate these attacks, it was shown how an adversary is able to bypass the read-out protection on different microcontrollers in Section 4.1 and how the control flow of a protected secure boot implementation can be tampered in Section 4.2. For side-channel attacks, this report refers to existing research [82],[88] and proposes valuable evaluations for future work. All three attacks can potentially lead to a theft of cryptographic secrets in an evil maid or theft scenario. The distinguishing factor of those two scenarios is the time and equipment an adversary has at hand, since the theft scenario implies that the device is not returned to the victim. However, it depends on the application whether it makes sense for an adversary to choose a theft attack over an evil maid attack. Supply chain attacks are mostly realized with control flow manipulation and read-out attacks, although side-channel information can be exploited as well. An adversary who wants to steal IP can take advantage of all three attacks. IP theft usually implies more capabilities in relation to time and equipment, similar to the device theft scenario.

The attacks that were demonstrated in this report are expected to be practically feasible on almost all microcontrollers reported in our market analysis. This implies that products built upon these microcontrollers must have dedicated countermeasures in place. For all three attack classes, researchers proposed various countermeasures to mitigate or impede the attacks. This report evaluated efficient and low-cost software countermeasures, as these can be retrofitted to existing products.

To counteract control flow manipulations, a proof-of-concept compiler-based countermeasure built on existing error detection schemes was demonstrated. For attacks on read-out protections, this report established that no universal countermeasure that can be integrated by means of software exists. However, depending on the microcontroller and the application, software-based countermeasures may impede the effect of such an attack and thus provide a sufficient level of protection. The topic of side-channel attacks was not practically evaluated in this report, but an evaluation of masked software implementations in face of microcontrollers with low physical noise was proposed for future work. Alternatively, the approach of retrofitting new leakage resilient cryptography schemes [31] similar to the demonstration in [96] is proposed.

The work conducted in this report demonstrates the difficult situation, developers of products built from COTS microcontrollers find themselves in. For one, being able to judge the threat of hardware attacks to every little implementation detail can not be expected from every embedded developer. Second, vendors of COTS microcontrollers hardly provide any support regarding hardware security, e.g. by listing possible threats or providing hardware security features. Third and last, time and budget constraints often seem to prevent a thorough review process that includes a hardware security perspective. Unfortunately, this report showed that such a review, conducted by professionals in the field of hardware security, is indispensable.

This report showed that there are efforts that automate the application of certain countermeasures. For example, control flow manipulations can be mitigated with compiler features as demonstrated in the

practical evaluation. However, the compiler features are not mature enough for adoption by industry and even if they were, their overhead regarding code size and performance is immense. Developers might be tempted to harden only certain parts of an implementation, but this report established that an expert is needed to identify all critical parts of an implementation. A similar statement can be made on the application of masking to harden a device against power/ EM side-channel attacks. While tooling and code for masked ciphers exist, factors like the noise level of a microcontroller need to be considered. For attacks on the read-out protection of microcontrollers, no straightforward countermeasure which can be applied by means of automatic tooling was identified. Countermeasures rather need to be implemented on application level to achieve sufficient mitigation of such attacks.

This report appeals to three interest groups to change this unsatisfying situation:

The research community needs to do more to improve tooling for software-based countermeasures in terms of practicability and usability for embedded developers.

Microcontroller vendors should provide threat models with respect to hardware attacks for their products to ease product selection for security-related tasks. This gives developers a basis to make decisions on whether a product is suitable to process or store sensitive information in the presence of hardware attacks.

Finally, the third group represents consumers and legislative authorities. This group should create an economic incentive to integrate countermeasures against hardware attacks such that they are considered inalienable for certain use cases. For some areas this incentive may be supported, for example, by regulatory measures providing technical guidelines or recommendations.

Bibliography

[1] Assessing the effectiveness of mcuboot protections against fault injection attacks. URL https://static.linaro.org/connect/lvc21f/ presentations/LVC21F-116.pdf

[2] Aktuelle Zertifikate nach CC. URL https://www.bsi.bund. de/DE/Themen/Unternehmen-und-Organisationen/ Standards-und-Zertifizierung/Zertifizierung-und-Anerkennung/ Zertifizierung-von-Produkten/Zertifizierung-nach-CC/Zertifizierte-Produkte-nach-CC/ zertifizierte-produkte-nach-cc_node.html

[3] Stay safe shopping for hardware wallets. URL https://blog.trezor.io/stay-safe-shopping-for-hardware-wallets-543f144e3d24

[4] List of common microcontrollers. URL https://en.wikipedia.org/wiki/List_of_common_microcontrollers

[5] Arnold Abromeit, Florian Bache, Leon A. Becker, Marc Gourjon, Tim Güneysu, Sabrina Jorn, Amir Moradi, Maximilian Orlt, and Falk Schellenberg. Automated masking of software implementations on industrial microcontrollers. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1006–1011, 2021. doi: 10.23919/DATE51398.2021.9474183

[6] Melissa Azouaoui, Davide Bellizia, Ileana Buhan, Nicolas Debande, Sebastien Duval, Christophe Giraud, Eliane Jaulmes, Francois Koeune, Elisabeth Oswald, Francois-Xavier Standaert, and Carolyn Whitnall. A systematic appraisal of side channel evaluation strategies. Cryptology ePrint Archive, Paper 2020/1347, 2020. URL https://eprint.iacr.org/2020/1347

[7] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. Cryptology ePrint Archive, Paper 2014/413, 2014. URL https://eprint.iacr.org/2014/413

[8] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006

[9] Thierno Barry, Damien Couroussé, and Bruno Robisson. Compilation of a countermeasure against instruction-skip fault attacks. In *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems*, pages 1–6, 2016

[10] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and Francois-Xavier Standaert. maskverif: Automated verification of higher-order masking in presence of physical defaults. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *Computer Security – ESORICS 2019*, pages 300–318, Cham, 2019. Springer International Publishing. ISBN 978-3-030-29959-0

[11] Gilles Barthe, Marc Gourjon, Benjamin Gregoire, Maximilian Orlt, Clara Paglialonga, and Lars Porth. Masking in fine-grained leakage models: Construction, implementation and verification. Cryptology ePrint Archive, Paper 2020/603, 2020. URL https://eprint.iacr.org/2020/603

[12] Alberto Battistello, Jean-Sebastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the isw masking scheme. Cryptology ePrint Archive, Paper 2016/540, 2016. URL https://eprint.iacr.org/2016/540

[13] Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. Random probing security: Verification, composition, expansion and new constructions. Cryptology ePrint Archive, Paper 2020/786, 2020. URL https://eprint.iacr.org/2020/786

[14] 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. doi: 10.13154/tches.v2019.i2.199-224 URL https://tches.iacr.org/index.php/TCHES/article/view/7390

[15] Luís T. A. N. Brandão, Michael Davidson, and Apostol Vassilev. Nist roadmap toward criteria for threshold schemes for cryptographic primitives. NISTIR 8214A, 2020. URL https://doi.org/10.6028/NIST.IR.8214A

[16] Olivier Bronchain and François-Xavier Standaert. Side-channel countermea- sures' dissection and the limits of closed source security evaluations. Cryptology ePrint Archive, Paper 2019/1008, 2019. URL https://eprint. iacr.org/2019/1008

[17] Olivier Bronchain and François-Xavier Standaert. Breaking masked implementations with many shares on 32-bit software platforms: or when the security order does not matter. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):202–234, Jul. 2021. doi: 10.46586/tches.v2021.i3.202-234. URL https://tches.iacr.org/index.php/TCHES/article/view/8973

[18] Olivier Bronchain, Gaëtan Cassiers, and Francois-Xavier Standaert. Give me 5 minutes: Attacking ascad with a single side-channel trace. Cryptology ePrint Archive, Paper 2021/817, 2021. URL https://eprint.iacr.org/2021/817

[19] Olivier Bronchain, Francois Durvaux, Loïc Masure, and Francois-Xavier Standaert. Efficient profiled side-channel analysis of masked implementations, extended. *IEEE Transactions on Information Forensics and Security*, 17: 574–584, 2022. doi: 10.1109/TIFS.2022.3144871

[20] Nicolas Bruneau, Sylvain Guilley, Zakaria Najm, and Yannick Teglia. Multivariate high-order attacks of shuffled tables recomputation. Cryptology ePrint Archive, Paper 2015/837, 2015. URL https://eprint.iacr.org/2015/837

[21] IR Buhan, Lejla Batina, Yuval Yarom, and Patrick Schaumont. Sok: Design tools for side-channel-aware implementations. Cryptology ePrint Archive, Paper 2021/497, 2021. URL https://eprint.iacr.org/2021/497

[22] Gaëtan Cassiers, Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Spookchain: Chaining a sponge-based aead with beyond-birthday security. In Shivam Bhasin, Avi Mendelson, and Mridul Nandi, editors, *Security, Privacy, and Applied Cryptography Engineering*, pages 67– 85, Cham, 2019. Springer International Publishing. ISBN 978-3-

030-35869-3

[23] Gaëtan Cassiers, Sebastian Faust, Maximilian Orlt, and François-Xavier Standaert. Towards tight random probing security. Cryptology ePrint Archive, Paper 2021/880, 2021. URL https://eprint.iacr.org/2021/880

[24] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 398–412, Berlin,
Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-48405-9

[25] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems*
— *CHES 2000*, pages 252–263, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-44499-2

[26] Yann Le Corre, Johann Großschädl, and Daniel Dinu. Micro-architectural power simulator for leakage assessment of cryptographic software on arm cortex-m3 processors. Cryptology ePrint Archive, Paper 2017/1253, 2017 URL https://eprint.iacr.org/2017/1253

[27] Ana Covic, Fatemeh Ganji, and Domenic Forte. Circuit masking: From theory to standardization, a comprehensive survey for hardware security researchers and practitioners, 2021. URL https://arxiv.org/abs/2106.12714

[28] 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. doi: 10.1109/JIOT. 2018.2890277

[29] Moslem Didehban and Aviral Shrivastava. nzdc: A compiler technique for near zero silent data corruption. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2016. doi: 10.1145/2897937. 2898054

[30] Moslem Didehban, Aviral Shrivastava, and Sai Ram Dheeraj Lokam. Nemesis: A software approach for computing in presence of soft errors. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 297–304, 2017. doi: 10.1109/ICCAD.2017.8203792

[31] CE Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, BJM Mennink, Robert Primas, and Thomas Unterluggauer. Isap v2. 0. 2020

[32] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, and Thomas Unterluggauer. Isap – towards side-channel secure authenticated encryption. *IACR Transactions on Symmetric Cryptology*, 2017(1):80–105, Mar. 2017. doi: 10.13154/tosc.v2017.i1.80-105. URL https://tosc.iacr.org/index.php/ToSC/article/view/585

[33] François Durvaux, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. How to certify the leakage of a chip? Cryptology ePrint Archive, Paper 2013/706, 2013. URL https://eprint.iacr.org/2013/706.

[34] François Durvaux, François-Xavier Standaert, and Santos Merino Del Pozo. Towards easy leakage certification. Cryptology ePrint Archive, Paper 2015/537, 2015. URL https://eprint.iacr.org/2015/537

[35] Jean-Max Dutertre, Alexandre Menu, Olivier Potin, Jean-Baptiste Rigaud, and Jean-Luc Danger. Experimental analysis of the electromagnetic instruction skip fault model and consequences for software countermeasures. *Micro- electronics Reliability*, 121:114133, 2021

[36] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In Henri Gilbert, editor, *Advances in Cryptology – EU- ROCRYPT 2010*, pages 135–156, Berlin, Heidelberg, 2010. Springer Berlin
Heidelberg. ISBN 978-3-642-13190-5

[37] Christof Fetzer, Ute Schiffel, and Martin Süßkraut. An-encoding compiler: Building safety-critical systems with commodity hardware. In *International Conference on Computer Safety, Reliability, and Security*, pages 283–296.
Springer, 2009

[38] Si Gao, Ben Marshall, Dan Page, and Elisabeth Oswald. Share-slicing: Friend or foe? *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):152–174, Nov. 2019. doi: 10.13154/tches.v2020.i1.152-174. URL https://tches.iacr.org/index.php/TCHES/article/view/8396

[39] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. Coco: Co-design and co-verification of masked software implementations on cpus. Cryptology ePrint Archive, Paper 2020/1294, 2020. URL https://eprint.iacr.org/2020/1294

[40] Aron Gohr, Friederike Laus, and Werner Schindler. Breaking masked implementations of the clyde-cipher by means of side-channel analysis - a report on the ches challenge side-channel contest 2020. Cryptology ePrint Archive, Pa- per 2022/471, 2022. URL https://eprint.iacr.org/2022/471

[41] Tavis Goodspeed. A 16 Bit Rootkit, and Second Generation Zigbee Chips. 2009. URL https://www. blackhat.com/presentations/bh-usa-09/GOODSPEED/ BHUSA09-Goodspeed-ZigbeeChips-SLIDES.pdf

[42] Louis Goubin and Jacques Patarin. Des and differential power analysis (the "duplication" method). In *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '99, page 158–172,
Berlin, Heidelberg, 1999. Springer-Verlag. ISBN 354066646X

[43] Qian Guo, Vincent Grosso, François-Xavier Standaert, and Olivier Bronchain. Modeling soft analytical side-channel attacks from a coding theory viewpoint. Cryptology ePrint Archive, Paper 2018/498, 2018. URL https://eprint.iacr.org/2018/498

[44] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An aes smart card implementation resistant to power analysis attacks. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *Applied Cryptography and Network Security*, pages 239–252, Berlin, Heidelberg, 2006. Springer Berlin

Heidelberg. ISBN 978-3-540-34704-0

[45] 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. doi: 10.1109/ISVLSI51109.2021.00069

[46] Vincent Immler, Robert Specht, and Florian Unterstein. Your rails cannot hide from localized em: How dual-rail logic fails on fpgas. Cryptology ePrint Archive, Paper 2017/608, 2017. URL https://eprint.iacr.org/2017/608

[47] IncludeSec. Firmware dumping technique for an ARM Cortex-M0 SoC, November 2015. URL https://blog.includesecurity.com/2015/11/firmware-dumping-technique-for-an-arm-cortex-m0-soc/

[48] David Knichel, Pascal Sasdrich, and Amir Moradi. Silver - statistical independence and leakage verification. Cryptology ePrint Archive, Paper 2020/634, 2020. URL https://eprint.iacr.org/2020/634

[49] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-48405-9

[50] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology*
— *CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-68697-2

[51] Jean-François Lalande, Karine Heydemann, and Pascal Berthomé. Software countermeasures for control flow integrity of smart card c codes. In *European Symposium on Research in Computer Security*, pages 200–218. Springer, 2014

[52] Adam Laurie. Atmel SAM7XC Crypto Co-Processor key recovery (with bonus Mifare DESFire hack). URL http://adamsblog.rfidiot.org/2013/02/atmel-sam7xc-crypto-co-processor-key. html

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

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

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

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

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

[58] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):235–274, Jul. 2021. doi: 10.46586/tches.v2021.i3. 235-274. URL https://tches.iacr.org/index.php/TCHES/article/view/8974

[59] Ben Marshall, Dan Page, and James Webb. Miracle: Micro-architectural leakage evaluation. Cryptology ePrint Archive, Paper 2021/261, 2021. URL https://eprint.iacr.org/2021/261

[60] Loïc Masure and Rémi Strullu. Side channel analysis against the anssi's protected aes implementation on arm. Cryptology ePrint Archive, Paper 2021/592, 2021. URL https://eprint.iacr.org/2021/592.

[61] Loïc Masure, Valence Cristiani, Maxime Lecomte, and Francois-Xavier Standaert. Don't learn what you already know: Grey-box modeling for profiling side-channel analysis against masking. Cryptology ePrint Archive, Paper 2022/493, 2022. URL https://eprint.iacr.org/2022/493

[62] David McCann, Elisabeth Oswald, and Carolyn Whitnall. Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. Cryptology ePrint Archive, Paper 2016/517, 2016. URL https://eprint.iacr.org/2016/517

[63] Marcel Medwed, François-Xavier Standaert, and Antoine Joux. Towards super-exponential side-channel security with efficient leakage-resilient prfs. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 193–212, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-33027-8

[64] Thomas S. Messerges. Using second-order power analysis to attack dpa resistant software. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2000*, pages 238–251, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-44499-2

[65] Alyssa Milburn, Niek Timmers, Nils Wiersma, Ramiro Pareja, and Santiago Cordoba. There will be glitches: Extracting and analyzing automotive firmware efficiently. *Black Hat USA*, 2018

[66] Mohamed Saied Emam Mohamed, Stanislav Bulygin, Michael Zohner, Annelie Heuser, Michael Walter, and Johannes Buchmann. Improved algebraic side- channel attack on aes. In *2012 IEEE International Symposium on Hardware- Oriented Security and Trust*, pages 146–151, 2012. doi: 10.1109/HST.2012. 6224335

[67] Shoei Nashimoto, Naofumi Homma, Yu-ichi Hayashi, Junko Takahashi, Hitoshi Fuji, and Takafumi Aoki. Buffer overflow attack with multiple fault injection and a proven countermeasure. *Journal of Cryptographic Engineering*, 7(1):35–46, 7 2017. doi: 10.1007/s13389-016-0136-3

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

https://www.usenix.org/conference/woot17/workshop-program/presentation/obermaier

[69] Johannes Obermaier, Marc Schink, and Kosma Moczek. One exploit to rule them all? on the security of drop-in replacement and counterfeit microcontrollers. In *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association, August 2020. URL https://www.usenix.org/ conference/woot20/presentation/obermaier

[70] Colin O'Flynn. MIN()imum failure: EMFI attacks against USB stacks. In *13th USENIX Workshop on Offensive Technologies (WOOT 19)*, Santa Clara, CA, August 2019. USENIX Association. URL https://www.usenix.org/ conference/woot19/presentation/oflynn

[71] Colin O'Flynn and Zhizhang David Chen. Side channel power analysis of an aes-256 bootloader. In *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 750–755, 2015. doi: 10.1109/CCECE.2015.7129369

[72] Colin O'Flynn and Alex Dewar. On-device power analysis across hardware security domains. Cryptology ePrint Archive, Paper 2019/689, 2019. URL https://eprint.iacr.org/2019/689

[73] N. Oh, P.P. Shirvani, and E.J. McCluskey. Control-flow checking by software signatures. *IEEE Transactions on Reliability*, 51(1):111–122, 2002. doi: 10.1109/24.994926

[74] N. Oh, P.P. Shirvani, and E.J. McCluskey. Error detection by duplicated instructions in super-scalar processors. *IEEE Transactions on Reliability*, 51(1): 63–75, 2002. doi: 10.1109/24.994913

[75] Kostas Papagiannopoulos and Nikita Veshchikov. Mind the gap: Towards secure 1st-order masking in software. Cryptology ePrint Archive, Paper 2017/345, 2017. URL https://eprint.iacr.org/2017/345

[76] Sean F. Parkinson and Eric A. Young. Blinding function in elliptic curve cryp- tography, 2014. URL https://patents.google.com/patent/ US9584320B1/en

[77] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 142–159, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-38348-9

[78] Julien Proy, Karine Heydemann, Alexandre Berzati, and Albert Cohen. Compiler-assisted loop hardening against fault attacks. *ACM Transactions on Architecture and Code Optimization (TACO)*, 14(4):1–25, 2017

[79] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In Isabelle Attali and Thomas Jensen, editors, *Smart Card Programming and Security*, pages 200– 210, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-45418-2

[80] Maurizio Rebaudengo, M Sonza Reorda, and Massimo Violante. A new software-based technique for low-cost fault-tolerant application. In

Annual Reliability and Maintainability Symposium, 2003., pages 25–28. IEEE, 2003

[81] George A Reis, Jonathan Chang, Neil Vachharajani, Ram Rangan, and David I August. Swift: Software implemented fault tolerance. In *International symposium on Code generation and optimization*, pages 243–254. IEEE, 2005

[82] Thomas Roche, Victor Lomné, Camille Mutschler, and Laurent Imbert. A side journey to titan. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 231–248, 2021

[83] Eyal Ronen, Colin O'Flynn, Adi Shamir, and Achi-Or Weingarten. Iot goes nuclear: Creating a zigbee chain reaction. Cryptology ePrint Archive, Paper 2016/1047, 2016. URL https://eprint.iacr.org/2016/1047

[84] Thomas Roth. TrustZone-M(eh): Breaking ARMv8-M's security, 2019. URL https://media.ccc.de/v/36c3-10859-trustzone-m_eh_breaking_armv8-m_s_security

[85] H. Saputra, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, R. Brooks, S. Kim, and W. Zhang. Masking the energy behavior of des encryption [smart cards]. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 84–89, 2003. doi: 10.1109/DATE.2003.1253591

[86] Marc Schink and Johannes Obermaier. Taking a Look into Execute-Only Memory. page 13, 2019

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

[88] 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*, pages 176– 201, July 2021. ISSN 2569-2925. doi: 10.46586/tches.v2021.i3.176-201. URL https://tches.iacr.org/index.php/TCHES/article/view/8972

[89] Tobias Schneider and Amir Moradi. Leakage assessment methodology - a clear roadmap for side-channel evaluations. Cryptology ePrint Archive, Paper 2015/207, 2015. URL https://eprint.iacr.org/2015/207

[90] Uzair Sharif, Daniel Mueller-Gritschneder, and Ulf Schlichtmann. Repair: Control flow protection based on register pairing updates for sw-implemented hw fault tolerance. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(5s):1–22, 2021

[91] Uzair Sharif, Daniel Mueller-Gritschneder, and Ulf Schlichtmann. Compas: Compiler-assisted software-implemented hardware fault tolerance for risc-v. In *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4, 2022. doi: 10.1109/MECO55406.2022.9797144

[92] Madura A Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. Rosita: Towards automatic elimination of power- analysis leakage in ciphers. Cryptology ePrint Archive, Paper 2019/1445, 2019. URL https://eprint.iacr.org/2019/1445

[93] K. Tiri, M. Akmal, and I. Verbauwhede. A dynamic and differential cmos logic with signal independent power consumption to withstand differential power analysis on smart cards. In *Proceedings of the 28th European Solid-State Circuits Conference*, pages 403–406, 2002

[94] Michael Tunstall, Carolyn Whitnall, and Elisabeth Oswald. Masking tables— an underestimated security risk. Cryptology ePrint Archive, Paper 2013/735, 2013. URL https://eprint.iacr.org/2013/735

[95] Balazs Udvarhelyi, Olivier Bronchain, and François-Xavier Standaert. Security analysis of deterministic re-keying with masking and shuffling: Application to isap. In *Constructive Side-Channel Analysis and Secure Design: 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25–27, 2021, Proceedings*, page 168–183, Berlin, Heidelberg, 2021. Springer-Verlag. ISBN 978-3-030-89914-1. doi: 10.1007/978-3-030-89915-8_8.
URL https://doi.org/10.1007/978-3-030-89915-8_8

[96] Florian Unterstein, Marc Schink, Thomas Schamberger, Lars Tebelmann, Manuel Ilg, and Johann Heyszl. Retrofitting leakage resilient authenticated encryption to microcontrollers. Cryptology ePrint Archive, Paper 2020/960, 2020. URL https://eprint.iacr.org/2020/960

[97] Jens Vankeirsbilck, Niels Penneman, Hans Hallez, and Jeroen Boydens. Random additive signature monitoring for control flow error detection. *IEEE Transactions on Reliability*, 66(4):1178–1192, 2017. doi: 10.1109/TR.2017. 2754548

[98] Nikita Veshchikov. Silk: High level of abstraction leakage simulator for side channel analysis. In *Proceedings of the 4th Program Protection and Reverse Engineering Workshop*, PPREW-4, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781605586373. doi: 10.1145/2689702. 2689706. URL https://doi.org/10.1145/2689702.2689706.