TUM School of Computation, Information and Technology
Technische Universität München

# Tamper-Sensitive Design of PUF-Based Security Enclosures

**Kathrin A. Garb**

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

**Doktors der Ingenieurwissenschaften (Dr.-Ing.)**

genehmigten Dissertation.

Die Dissertation wurde am 19.06.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 02.10.2023 angenommen.

# Abstract

In the field of IT security, protection against hardware manipulations is of particular importance. The hardware, the system's root of trust, is the cornerstone of system integrity. Any manipulations at the hardware level will inevitably affect software security at the OS or application level. With physical access to the system, an attacker can extract secret keys and intellectual property, perform hardware modifications or even exchange fundamental components. Therefore, preventing access to critical hardware components is essential to any high assurance security module.

To prevent and recognize physical manipulation of Hardware Security Modules (HSMs), creating a barrier that hampers accessibility to critical components is crucial. Many designs for security enclosures have been proposed over the years that provide only partial protection of the critical areas or require a battery for continuous monitoring.

Novel technology based on Physical Unclonable Functions (PUFs) provides large-scale protection without needing a battery [ION$^+$19, IOK$^+$18] by surrounding the device with a capacitive mesh of electrodes. These PUF-based security enclosures differ in their measured capacitances, which are influenced by manufacturing variations.

Previous work on PUF-based enclosures demonstrated the reliable measurement of the small capacitances in the femtofarad range [OIHS18] and confirmed the Gaussian distribution of capacitances [ION$^+$19, IOK$^+$18]; thereby laying the groundwork for this PUF-based technology.

This work focuses on improving the tamper-sensitivity of PUF-based capacitive security enclosures and takes the next steps towards their operative deployment.

As a first step, I describe FORTRESS, the *FORtified Tamper-Resistant Envelope with an Embedded Security Sensor*. FORTRESS is a prototype HSM with integrated PUF-based security, a hardened operating system, and extended with complete post-processing and supply chain capabilities.

This is followed by a security analysis of FORTRESS, where I discuss the three most relevant hardware attacks. This includes micro-drilling of the capacitive enclosure, surface probing of electrodes, and magnetic probing of critical connections. Through analysis of these attacks, adequate countermeasures are proposed that strengthen the security and design of FORTRESS.

To enhance the tamper-sensitivity, I discuss an extension of FORTRESS through a tamper-sensitive error correction scheme based on the wiretap channel. It is crucial for the scheme's design to recognize an attack while still providing sufficient reliability against environmental changes. This is achieved by modeling both effects as a wiretap channel, implemented through a polar code construction. The proposed scheme achieves a physical layer security of 100 bits for a PUF-secret length of 306 bits.

Finally, I describe potential threats to FORTRESS through fault injection attacks. I first demonstrate how cryptographic algorithms are affected by fault injection, followed by a vulnerability analysis through ARCHIE, an architecture-independent framework for fault emulation. With ARCHIE, I target the critical sections of the FORTRESS software and discuss risk mitigation through different countermeasures.

# Zusammenfassung

Im IT-Sicherheitsbereich hat der Schutz vor Hardware Manipulationen eine besondere Stellung. Die Hardware als der Vertrauensanker des Systems, ist der Eckpfeiler der Systemintegrität. Manipulationen auf der Hardwareebene beeinflussen unweigerlich die Softwaresicherheit der Betriebssystem- oder Anwendungsschicht. Durch physischen Zugriff zum System, kann ein Angreifer geheime Schlüssel extrahieren, geistiges Eigentum abgreifen, Hardwareveränderungen durchführen, oder sogar Komponenten austauschen. Deshalb ist eine der grundlegenden Anforderungen an ein Hochsicherheitsmodul, kritische Komponenten vor physischem Zugriff schützen.

Um die physische Manipulation von Hardware-Sicherheitsmodulen (HSMs) zu verhindern und zu erkennen, ist eine Barriere, die den Zugang zu kritischen Komponenten verhindert, maßgeblich. Viele Designs für Sicherheitsfolien wurden über die Jahre vorgeschlagen. Diese bieten entweder nur einen Teilschutz für kritische Bereiche, oder erfordern eine Batterie für die kontinuierliche Überwachung.

Neue Technologien basierend auf Physical Unclonable Functions (PUFs), die das System mit einem kapazitiven Elektrodengitter umhüllen, ermöglichen einen großflächigen Schutz, ohne die Notwendigkeit einer Batterie [ION+19, IOK+18]. Die Kapazitäten der PUF-basierten Sicherheitsfolien unterscheiden sich voneinander, da sie durch Schwankungen im Herstellungsprozess beeinflusst werden.

Voherige Arbeiten zu PUF-basierten Sicherheitsfolien zeigten die zuverlässige Messung der Kapazitäten im Femtofarad-Bereich [OIHS18], und bestätigten die Gaußverteilung der gemessenen Kapazitäten [IOK+18, ION+19]. Diese Arbeiten bilden die Grundlage für PUF-basierte Folientechnologien.

Diese Arbeit schließt an die Vorarbeiten an, und fokussiert sich darauf die Sensitivität der PUF-basierten Sicherheitsfolien gegenüber physischen Manipulationsversuchen zu verbessern. Hierdurch werden die nächsten Schritte in Richtung des operativen Einsatzes dieser Technologie unternommen.

Zunächst, beschreibe ich FORTRESS, einen Hardwaresicherheitsmodul-Prototyp mit integrierter PUF-basierter Sicherheit, der durch ein gehärtetes Betriebssystem, eine vollständige Datenverarbeitung, und Lieferkettenaspekte erweitert wurde.

Dem folgt eine Sicherheitsanalyse von FORTRESS, in der ich die drei relevantesten Hardwareangriffe diskutiere. Dies beinhaltet Bohrangriffe auf die Sicherheitsfolie, Oberflächensondierung der Gitterelektroden, und magnetische Sondierung der kritischen Busverbindungen. Die Angriffsanalyse ermöglicht die Bewertung geeigneter Gegenmaßnahmen, die die Sicherheit und das Design von FORTRESS verbessern.

Um die Sensitivität gegenüber Manipulationen zu steigern, wird FORTRESS durch einen manipulationssensitiven Fehlerkorrekturcode, basierend auf dem Wiretap Channel, erweitert. Für das Codedesign ist es entscheidend einen Angriff zu erkennen, und zugleich eine ausreichende Zuverlässigkeit gegenüber Umweltveränderungen sicherzustellen. Dies wird durch die Modellierung als Wiretap Channel erreicht, der durch eine Polar Code Konstruktion implementiert wird. Der vorgeschlagene Code erreicht eine physische Sicherheit von 100 Bits für ein PUF-Geheimnis von 306 Bits Länge.

Zuletzt beschreibe ich die potentielle Bedrohung von FORTRESS durch gezielte Fehlereinbringung. Ich zeige, zunächst, wie kryptographische Algorithmen durch Fehlereinbringung beeinflusst werden können. Dem folgt eine Vulnerabilitätsanalyse durch ARCHIE, ein architekturunabhängiges Framework zur Emulation von Fehlern. Mit ARCHIE untersuche ich die kritischen Bereiche der FORTRESS Software, und diskutiere die Risikominderung durch unterschiedliche Gegenmaßnahmen.

# Acknowledgment

Thanks to all co-authors for your critical eyes and minds that helped find flaws and improve the quality of the publications. Thanks for the good collaboration, dedication, respectful atmosphere, and everything I learned from you.

First, I want to thank Dr.-Ing. Johannes Obermaier and Ludwig Kürzinger, who, during my doctorate, became not only technical supervisors but also mentors. I learned a lot from you, and thank you for your continuous encouragement.

Special thanks go to Lukas Auer, Dr. Sven Plaga, Stefan Tatschner, and Philip Sperl for the mental support that contributed to the success of this endeavor.

Thanks to Prof. Dr.-Ing. Georg Sigl for supervising my dissertation and Prof. Dr.-Ing. Antonia Wachter-Zeh for accepting the role of the second examiner. I also want to thank Prof. Dr. Claudia Meitinger, my mentor, who helped me gain a different perspective whenever necessary.

Thanks to Prof. Dr.-Ing. Dominik Merli for encouraging me to pursue a Ph.D. Without this encouragement, I would have missed an important opportunity for professional and personal growth.

# Contents

# List of Prior Publications

[GO20] Kathrin Garb and Johannes Obermaier. Temporary Laser Fault Injection into Flash Memory: Calibration, Enhanced Attacks, and Countermeasures. In *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2020.

[GOFK21] Kathrin Garb, Johannes Obermaier, Elischa Ferres, and Martin König. FORTRESS: FORtified Tamper-Resistant Envelope with Embedded Security Sensor. In *2021 18th International Conference on Privacy, Security and Trust (PST)*, pages 1–12, 2021.

[GSHO21] Kathrin Garb, Marc Schink, Matthias Hiller, and Johannes Obermaier. Attacks and countermeasures for capacitive puf-based security enclosures. In *2021 IEEE Physical Assurance and Inspection of Electronics (PAINE)*, pages 1–8, 2021.

[GXKF22] Kathrin Garb, Marvin Xhemrishi, Ludwig Kürzinger, and Christoph Frisch. The Wiretap Channel for Capacitive PUF-Based Security Enclosures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(3):165–191, 2022.

[HGA$^+$21] Florian Hauschild, Kathrin Garb, Lukas Auer, Bodo Selmke, and Johannes Obermaier. ARCHIE: A QEMU-Based Framework for Architecture-Independent Evaluation of Faults. In *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, pages 20–30. IEEE, 2021.

[MXP$^+$21] Georg Maringer, Marvin Xhemrishi, Sven Puchinger, Kathrin Garb, Hedongliang Liu, Thomas Jerkovits, Ludwig Kürzinger, Matthias Hiller, and Antonia Wachter-Zeh. Analysis of Communication Channels Related to Physical Unclonable Functions. *WCC 2022 The Twelfth International Workshop on Coding and Cryptography*, 2021.

# List of Figures

# List of Tables

# Acronyms

| | |
|---|---|
| ACA | Anisotropic-Conductive Adhesive. |
| ADC | Analog-to-Digital Converter. |
| AES | Advanced Encryption Standard. |
| AGWN | additive Gaussian white noise. |
| ALT | Accelerated Life Testing. |
| API | Application Programming Interface. |
| ARCHIE | ARCHitecture-Independent Evaluation. |
| ASIC | Application-Specific Integrated Circuit. |
| BCH | Bose Chaudhuri Hocquenghem. |
| BER | Bit Error Ratio. |
| CC | Common Criteria. |
| CEK | CSP-encryption-key. |
| CSDA | Continuous Slowing Down Approximation. |
| CSP | Critical Security Parameter. |
| Cu | copper. |
| DBT | Dynamic Binary Translation. |
| DMA | Direct Memory Access. |
| DMC | discrete memoryless channel. |
| DRAM | Dynamic Random Access Memory. |
| DuT | Device under Test. |
| ECDHE | Elliptic Curve Diffie Hellman Ephemeral. |
| EEPROM | Electrically Erasable Programmable Read-Only Memory. |
| EKMS | Embedded Key Management System. |
| EMFI | Electromagnetic Fault Injection. |
| EPROM | Erasable Programmable Read-Only Memory. |
| FEA | Finite Element Analysis. |
| FER | Frame Error Rate. |
| FFC | Flat Flexible Cable. |
| FIB | Focused Ion Beam. |
| FIPS | Federal Information Processing Standards. |

| | |
|---|---|
| FPGA | Field Programmable Gate Array. |
| FPU | Floating Point Unit. |
| GDB | GNU Debugger. |
| GPIO | General Purpose Input / Output. |
| HDA | Helper Data Algorithm. |
| HMAC | Keyed-Hash Message Authentication Code. |
| HSM | Hardware Security Module. |
| HVL | Half-Value Layer. |
| IC | Integrated Circuit. |
| IFA | Ineffective Fault Analysis. |
| KEK | Key-Encryption-Key. |
| LED | Light-Emitting Diode. |
| LFI | Laser Fault Injection. |
| LMC | Limited Magnitude Code. |
| LSB | Least Significant Bit. |
| MLC | multi-level cell. |
| MOSFET | metal–oxide–semiconductor field-effect transistor. |
| MPU | Memory Protection Unit. |
| MSB | Most Significant Bit. |
| MSE | Mean Squared Error. |
| NVM | Non-Volatile Memory. |
| OS | Operating System. |
| PCB | Printed Circuit Board. |
| PEA | Photonic Emission Analysis. |
| PI | polyimide. |
| PLL | Phase-Locked Loop. |
| POWF | Physical One-Way Function. |
| PUF | Physical Unclonable Function. |
| PWM | Pulse Width Modulation. |
| RNG | Random Number Generator. |
| ROP | Return-Oriented Programming. |
| RS | Reed Solomon. |
| RTL | Register Transfer Level. |

| | |
|---|---|
| SC | successive cancellation. |
| SCL | successive cancellation list. |
| SFA | Statistical Fault Analysis. |
| SIFA | Statistical Ineffective Fault Analysis. |
| SLC | single-level cell. |
| SLLC | Systematic Low Leakage Coding. |
| SMD | Surface Mounted Device. |
| SPI | Serial Peripheral Interface. |
| SRAM | Static Random-Access Memory. |
| TB | Translation Block. |
| TCG | Tiny Code Generator. |
| TRNG | True Random Number Generator. |
| UART | Universal Asynchronous Receiver/Transmitter. |
| VT | Varshamov-Tenengolts. |

# Chapter 1

# Introduction

The field of IT security has significantly gained importance in recent years and constitutes a rapidly growing market. In Germany alone, the expenses for IT security have almost doubled in the last five years, from 3.7 billion euros in 2017 to an estimated 6.2 billion euros in 2021 [Bit21], with an estimated annual growth rate of 10.2% [Boc21]. The worldwide expenses for IT security in 2020 amount to 133.78 billion US dollars [Gar21b], which constitutes 3.45% of all information technology expenses worldwide in 2020 [Gar22].

Many security concerns evolve around remote access and the exploitation of software vulnerabilities. At the same time, hardware-related threats, such as fault or side-channel attacks, reverse engineering, or especially manipulating components, are not as prevalent in public awareness. However, the worldwide expenses for IT security grouped by market segments show that infrastructure protection is ranked second after expenses for security services with 20.46 billion US dollars in 2020 [Gar21a]. Furthermore, a poll with business members of the VDE — a German association for Electrical, Electronic, and Information Technologies — and universities in Germany from 2019 showed that 67.6% of the respondents named human error or misconduct a threat, 31.4% feared the compromise of extranet and cloud components, 28.6% named sabotage, and 28.6% were afraid of compromise of smartphones and tablets in the production environment [VDE19]. Hence, entrepreneurs are well aware of the physical threats to their devices and systems.

Especially in high-security environments, protection against physical manipulation of Critical Security Parameters (CSPs) is crucial. These CSPs are often stored and managed in Hardware Security Modules (HSMs) that perform cryptographic operations. As specified in Common Criteria (CC) standards [Fed08, Int22a, Int22b, Int22c, Int22f] or Federal Information Processing Standards (FIPS) [Nat02, Nat19, BB19], HSMs require a mechanism for physical protection against tampering. This protection through a physical bound [Fed08, EL] comes in the form of coatings, covers, or enclosures to detect and respond to a tamper event through an alarm and zeroization procedure [EL, Int12]. One such example is the development of *battery-backed* enclosure technologies, which contain a mesh of traces whose electrical resistance is continuously monitored to detect a tamper event [OI18, GOR, Adv12].

Since the battery-backed solutions have several drawbacks, like a reduced lifetime or a higher sensitivity to environmental changes, *batteryless* capacitive enclosures based on PUFs have been developed recently. These enclosures make use of minuscule manufacturing variations within the capacitances of the electrode mesh in order to generate a key. If the enclosure is damaged, the PUF-key can not be reproduced, which triggers anti-tamper mechanisms.

Previous work on PUF-based capacitive security enclosures demonstrated the reliable measurement of the small capacitances in the femtofarad range [OIHS18] and confirmed the Gaussian distribution of capacitances [IOK$^+$18, ION$^+$19] and, hence, laid the groundwork for this PUF-based technology. The goal of this thesis is to investigate the tamper-sensitivity of PUF-based security enclosures, thereby taking the next steps towards their commercial deployment.

In **Chapter 2**, I give a general overview of tamper protection mechanisms and the history of Hardware Security Modules. Then, I introduce Physical Unclonable Functions emphasizing PUF-based tamper protection and previous work on security enclosures.

This is followed in **Chapter 3** by introducing FORTRESS, a prototype HSM based on the capacitive enclosure. This includes the design and assembly of FORTRESS and the development of the Next Generation EKMS, implementing a complete key generation and management, and incorporating supply chain aspects through a secure system life cycle. The results of this chapter were published in the IEEE Proceedings of the 18th Annual International Conference on Privacy, Security & Trust [GOFK21].

In **Chapter 4**, I discuss the most relevant physical attacks on FORTRESS. As I will show, FORTRESS is vulnerable to magnetic probing and micro-drilling attacks. In order to restore the enclosure's tamper-sensitivity, I propose countermeasures against both of the described attacks and a bypass attack published in 2019 by Obermaier [Obe19]. I presented the results of Chapter 4 at the 2021 IEEE Physical Assurance and Inspection of Electronics (PAINE) conference [GSHO21].

Post-processing of the PUF-response is a crucial step in the key generation process that incorporates error correction codes to compensate for noise and environmental influences. However, drilling attacks destroy electrodes and alter the PUF-response, and hence, the PUF-key. Therefore, the error correction code design has to ensure tamper-sensitivity to attacks while incorporating reliability to environmental changes. I tackle this issue in **Chapter 5** by first analyzing the effects of environmental influences to derive a model for the PUF. Subsequently, based on the PUF-model, $q-$ary polar codes are constructed and verified in a Monte Carlo simulation. The proposed code construction achieves a physical layer security of 100 bits for a PUF-secret length of 306 bits. The analysis and code construction were published in the IACR Transactions on Cryptographic Hardware and Embedded Systems [GXKF22].

A further class of attacks is targeted in **Chapter 6**. Even an HSM physically protected by the capacitive PUF-based enclosure is still potentially vulnerable to radiation fault attacks. In the first half of Chapter 6, I demonstrate how fault attacks can impact implemented cryptographic algorithms and firmware execution. The insights of this demonstration were published at the 2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS) [GO20]. In the second half, I determine vulnerable sections of the PUF post-processing software through ARCHIE, a framework for automated fault injection. Finally, I discuss possible countermeasures that can be incorporated into FORTRESS. ARCHIE, the architecture-independent framework for fault evaluation, is published in the IEEE proceedings of the 2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC) [HGA$^{+}$21].

I summarize and conclude the outcome of my thesis in **Chapter 7** and give an overview of possible future developments.

# Chapter 2

# Tamper Protection and Physical Unclonable Functions

In this chapter, I discuss the requirements of tamper protection mechanisms and provide an overview of their history. Furthermore, I introduce the term "Physical Unclonable Functions“ and motivate the need for PUF-based tamper protection.

## 2.1 Tamper Protection

As discussed in the introduction, tampering and physical manipulations severely threaten security and privacy, leading to serious personal, financial, and economic damage. To dive deeper into the field of tamper protection, I first discuss basic terms and definitions before addressing the formal background and standardization.

According to the NIST SP 800, *tampering* is defined as an "intentional but unauthorized act resulting in the modification of a system, components of systems, its intended behavior, or data“ [Nat20]. The associated *tamper event* can either be answered passively or through active countermeasures. *Tamper evidence* describes the automatic determination of a tamper event. The *tamper response*, however, constitutes an automatic action after detecting the tamper event, for instance, the zeroization of all relevant CSPs [Nat02]. *Tamper-sensitivity* entails sensing and counteracting a tamper event through a corresponding system design.

When tampering with a device, the goal of an attacker is to obtain information on

- Intellectual Property (IP), such as design or manufacturing specifics, signals, and data processing, or the firmware itself,
- Critical Security Parameters (CSPs), such as keys or other confidential data, which can be extracted, e.g., by reading out the memory or manipulating the execution of cryptographic algorithms,
- possible vulnerabilities in the hardware or software design; either to exploit them for financial gain and disparate personal values or to gain visibility and credit.

This shows that physical access to the device, signals, and electronic components is crucial for an attacker and thus, must be impeded through tamper protection measures.

Measures for tamper protection are not only demanded by the manufacturer or the customer, but in some cases, they are even legally required. One such example is the directives effective in the European Union that regulate the tamper-evidence of tachographs in road transport [Eur14] or the monitoring of emission limits in vehicles [Eur09] and non-road machinery [Eur16a, Eur16b]. These directives generally state that tamper protection measures are required but do not provide details on how to implement them. Different industry standards give slightly more specific requirements for their implementation [Int19, Int17, Int12, LLC09]. In the field of embedded security, certification based on Common Criteria (CC) [Fed08, Int22a, Int22b, Int22c, Int22d, Int22e, Int22f, JIW22, KS11, Fed11], or the Federal Information Processing Standard (FIPS) [Nat02, BB19, Nat19, EL] is often required to sell security-related products, such as smartcards, Trusted Platform Modules, or Hardware Security Modules.

According to the NIST Special Publication 800-57, a Hardware Security Module (HSM) is defined as a "physical computing device that safeguards and manages cryptographic keys and provides cryptographic processing. An HSM is or contains a cryptographic module." [BB19]. The international standard ISO/IEC 24759 defines a *hardware module* as a module "whose cryptographic boundary is specified at a hardware perimeter" [Int17]. Such perimeters or boundaries can be "physical structures, such as enclosures, potting, or encapsulation materials, connectors, and interfaces" [Int12]. These perimeters protect against unauthorized physical access, which could lead to a modification or substitution of the module. According to FIPS 140-2, the penetration of the enclosure has to be detected such that it results in the immediate zeroization of all plaintext CSPs [Nat02]. Furthermore, the cryptographic module has to be protected against compromise due to fluctuating environmental conditions.

HSMs are employed for managing keys or certificates to establish secure communication, verify transactions, or encrypt and authenticate critical data.

## 2.2 Historical Overview of Security Enclosures

Approaches protecting critical devices by additional physical measures have been developed for decades. Already in 1981, the United States government described the encapsulation of objects in "protective membranes" or "protective cocoons" that — if penetrated — provide a "penalty", for instance, in the form of shaped charges triggered by applying an external voltage [Boa81]. This penalty can be anything from an explosion to a remote alarm, destruction of circuitry, or obliteration of sensitive data. The National Security Agency (NSA) declassified the corresponding document in 2008 [Boa81].

In 1984, Chaum discussed tamper-responding containers that follow a multi-layered approach and incorporate active or passive sensors to detect a tamper event [Cha84]. Weingart and Chaum noted that, in 1986, Price had described potting of circuits to hamper accessibility, where wires embedded in the potting served as intrusion sensors [Wei87, Cha84].

In 1987, the IBM µABYSS was published, a co-processor system design encapsulated by a package surrounded by a wire "cocoon" [Wei87, Whi87]. Upon a break-in event, the four wire layers are damaged, which leads to an erasure of the protected data. The wire cocoon is covered by a hard, opaque epoxy potting. A thin nichrome layer insulates the wires to prevent short or open circuits. If an attacker attempts to dissolve the potting, the insulation layer of the wires will also dissolve, causing shorts and triggering the alarm. To make the system more susceptible to heat, The potting material is filled with aluminum or silica. Hence, thermal stress will cause the package to crack and break wires. The IBM µABYSS continuously monitors the electrical resistance of the wire cocoon to detect a tamper event.

In 1989, MacPherson filed a patent for security enclosures consisting of layers of flexible material [Mac90]. Two layers, made of a semiconductive material that are separated by an insulating layer cover the enclosure. Due to the thin insulating layer, piercing the enclosure will force the semiconductive fibers into contact. MacPherson's proposal is also based on measurements of electrical resistance. MacPherson described a similar idea in a further patent in 1993 [Mac93].

In 1992, Hubert filed a patent describing a device embedded in a piezo-electric material, which generates an electric field upon an exerted force [Hub92]. This electric field is monitored and triggers the destruction of the secure information stored on the device.

In 2005, Eren and Sandor presented capacitive sensors for tamper-resistant enclosures [ES05]. The capacitive sensors printed onto a circuit board are arranged in a serpentine structure. Placing an object in the vicinity of two adjacent electrodes alters their capacitance due to fringing. Removing the enclosure also removes the object, which leads to an alarm and an erasure of cryptographic keys. According to the authors, the capacitive sensors conform with levels 3 and 4 of FIPS 140-2.

One year later, Fischer and Froschermeier filed a patent describing an electronic security module that contains a protected area covered by a circuit board on the top and bottom side [FF06]. The printed circuit boards are contacted, and a resistance measurement is performed. The resulting resistance is compared with stored values to detect an intrusion into the security module.

In 2013, the GORE envelope [GOR, IMJFC13] was published, which protects the IBM 4765 Crypto Coprocessor [OI18]. It consists of a meander mesh of electrodes whose resistance is monitored against known base values [IMJFC13]. The four mesh layers have a trace width of 300 µm. According to the authors, the HSM based on the IBM 4765 achieved compliance with FIPS 140-2 level 4 [Adv12].

Another cryptographic module based on resistance measurements is the HP Atalla Cryptographic Subsystem (ACS) [OI18, Hew15]. The top and bottom of the ACS PCB are protected by covers containing an electrode mesh. Additional sensors detect the removal of the covers, which triggers the alarm and deletes all CSPs. Due to its coarse electrode mesh, the ACS does not protect against drilling attacks with a diameter below 1 mm [OI18].

These proposals for security enclosures relied on resistance measurements, and the electrode monitoring required a continuous power supply in the form of a battery. However, batteries make the HSM bulky and more susceptible to environmental changes. Furthermore, the battery charge limits the lifetime of the HSM and requires maintenance.

Little is known about the attack susceptibility of these HSMs since public information on the technical details and vulnerabilities of certified security enclosure systems is not readily available. Furthermore, obfuscation and nondisclosure of implementation details are encouraged by the certification process and awarded additional points in the overall score [Int22f, JIW22, Fed08]. Obermaier and Immler disassembled several battery-backed HSMs to counteract this lack of information and analyzed their components and potential weaknesses [OI18]. They found the enclosure meshes to be relatively coarse, allowing an attacker to circumvent the security mechanisms by drilling holes with standard retail equipment. Conceptually, battery-backed enclosures are based on the enclosure's resistance. The static voltage at specific points in the mesh is measured to determine if the resistance is within the expected range. Hence, an attacker could bypass the electrode mesh by forcing an external voltage into the circuit. This leaves portions of the mesh unprotected. Furthermore, the resistance-based approach allows successively replacing an electrode part with a corresponding resistor. Attacks on software components are unknown since no details on the implementation were published. In Section 2.4 and Chapter 4, I discuss attacks on security enclosures in the context of Physical Unclonable Functions.

Recently, *batterless* enclosures based on Physical Unclonable Functions (PUFs) have been developed to overcome the drawbacks of battery-backed HSMs [IOK$^+$18, ION$^+$19]. I discuss PUF-based security enclosures in Chapter 3. However, to first provide a context for PUF-based tamper protection, I give an overview of Physical Unclonable Functions in the next section.

## 2.3 Physical Unclonable Functions

In this section, I introduce PUFs and give an overview of their historical development, focusing on their application in tamper protection.

### 2.3.1 Physical Fingerprints

A Physical Unclonable Function (PUF) can be seen as an object's fingerprint [Mae12] that is based on minuscule variations in a *physical* structure, such as an integrated circuit. The unique physical patterns occur during the manufacturing, whenever the precision in the fabrication process is limited. These are, for instance, variations in the layer thickness, surface roughness, doping, or height and width of electrodes and connectors. Also, slight natural variations in the raw materials can lead to unique patterns and structures in the fabricated device. However, the physical properties contributing to a PUF have to be chosen such that they do not represent global manufacturing variations.

The minuscule varying patterns occur randomly and are, in general, *unclonable*. Hence, they should neither be replicable physically nor mathematically through machine learning or statistical methods.

Furthermore, a PUF is a *function* that gives a response when provided with a challenge. Depending on its use case, the design of the PUF includes single or multiple challenges. Furthermore, depending on the space of challenge-response pairs, PUFs are classified into weak or strong PUFs. The ISO/IEC 20897 [Int20, Int22g] provides a standard for PUFs that categorizes them according to their use case, i.e., key generation, identification, or authentication. Single response PUFs basically represent a "storage" for a single CSP, while in the case of authentication, a large list of challenge-response pairs is stored for attestation.

PUFs should fulfill different requirements [Int20], such as steadiness, randomness, uniqueness, tamper-resistance, mathematical unclonability, and physical unclonability. Steadiness is a measure of the reproducibility of the PUF-response, while uniqueness describes the difference between two PUF instances. Mathematical unclonability refers to simulating or learning the PUF.

Apart from these requirements, the PUF community has defined additional properties for evaluating a PUF [Mae12, Wil21, MGS13, HYKS10, MKP08]. Furthermore, an analysis of entropy and possible correlations is necessary, and testing methods for True Random Number Generators (TRNGs) can be included [Nat10, KS11, Fed11]. However, these tests alone are insufficient since they only address the randomness of the PUF while discarding its reliability and other PUF-properties. Furthermore, tests for TRNGs are designed for a large input sequence; hence, multiple PUF-responses have to be concatenated to adjust the input data to custom TRNGs tests [Nat10, KS11, Fed11]. What is more, evaluating a PUF requires a large number of physical devices, which often are not available.

### 2.3.2 Historical Overview

The idea of using physical properties for security applications is not new.

In 1978, Brosow and Furugard addressed an object's authentication against forgery through random imperfections in the object's base material [BF78]. Their patent described that the random imperfections are converted to a binary code, which is stored and later compared to the regenerated binary code to verify the object's identity.

In 1986, Samyn proposed authenticating banknotes or credit cards comprised of randomly conductive fibers [Sam86]. Microwaves scan a section of the fibers to obtain a digitally coded signal.

Graybeal and McFate reported already in 1989 that Sandia National Laboratories had experimented with unique reflected particle patterns that were installed on weapon systems for verification purposes [GM89]. In 1992, Tolk extended the use of reflective particle tags —verified through image comparison — to uniquely identify other critical equipment [Tol92]. Tolk also provided guidelines on the system design.

In 1995, van Renesse built a prototype verification system, which he named 3 Dimensional Structure Authentication System (3DAS) [vR95]. 3DAS was composed of randomly arranged 40 µm polymer fibers that could be mounted on an ID card or product label. The recorded fiber patterns were read and verified through Light-Emitting Diodes (LEDs).

In the following year, a patent filed by Denenberg et al. addressed the authentication of art or jewelry through unique patterns at the microscopic level [DPDC93]. The object is identified through visual analysis of its unique intrinsic features.

In 1998, another patent was published proposing multiple scattering of coherent radiation in an inhomogeneous medium to detect intrusion into a tamper-proof package [ADG96]. The tamper attempt is detected through a change of radiation intensity. The response of the medium provides a unique identity key.

This approach was followed in 1999 by a prototype implementation for the physical identification of documents [SS99]. The authors used the randomly varying physical features of printed paper to verify documents, illuminating a portion of the paper to identify the structural pattern of the paper texture.

The extraction of unique and repeatable information from integrated circuits was demonstrated in 2000 by Lofstrom et al. [LDT00]. Due to a mismatch between transistors stemming from randomness in silicon processing, the drain currents differ randomly for each die, leading to a sequence of random voltages that can be used for integrated circuit identification.

In 2001, Pappu et al. introduced the notion of *Physical One-Way Functions (POWFs)* [Pap01, PRTG02]. They build an optical POWF based on laser speckle fluctuations translating the microscopic changes in a 3D epoxy structure to a fixed-length key. The epoxy structure is illuminated by a 632.8 nm HeNe laser and filtered, resulting in an optical hash. The optical hash provides a discretely sampled image of the speckle intensity representing a fixed-length key. The one-wayness of the physical function requires it to be non-invertible through any algorithm [Pap01].

In 2002, Layman et al. filed a patent for electronic fingerprinting of integrated circuits [LCNR02]. They based the Integrated Circuit (IC) identification on the initial state of certain memory cells. The startup value of memory cells is consistent at each power-up. This is caused by manufacturing variations that lead to a mismatch between the transistors of each memory cell. This idea is closely related to what later became known as SRAM PUFs.

In the same year, Gassend et al. introduced the term *Physical Random Function*, abbreviated as PUF — which stands for *Physical Unclonable Function* — to prevent confusion with Pseudo-Random Functions [Gas03, GCvDD02]. They defined a Physical Random Function as a function mapping challenges to responses, which is easy to evaluate and hard to characterize. According to Gassend et al., PUFs and POWFs are similar and differ mainly in one aspect: control. A Controlled PUF is evaluated by an algorithm that is physically bound to the PUF. The POWF presented by Pappu et al. could be modified to fulfill that requirement by integrating the light source into the epoxy structure instead of manually changing the position of the light source in correspondence with the challenge. Furthermore, light sensors embedded on a chip would have to be integrated into the epoxy structure surrounded by a reflecting material [Gas03].

Gassend et al. suggested constructing PUFs based on statistical variations in delays of devices and wires [GCvDD02, GCvDD03]. They also proposed an architecture for a delay-based PUF, which became known as the Arbiter PUF [LLG$^+$04, LLG$^+$05]. The simple Arbiter PUF is based on the simultaneous excitation of two delay paths that are configured through switch components. Both delay paths are racing against each other, and an arbiter block at the output detects which rising edge arrives first. The binary input challenge determines which path is to be taken. The total delay is the sum of the individual component and wire delays, assuming an additive delay model. The Arbiter PUF maps a multi-bit challenge input to a single-bit response output. Due to an exponentially large number of challenge-response pairs, the authors proposed the Arbiter PUF to identify and authenticate ICs.

An error-correcting code is necessary to ensure the PUF's reliability since external factors lead to slight variations in the measured responses [GCvDD02]. In 2004, Dodis et al. published the Fuzzy Extractor, a method to extract strong keys from noisy data, which originated in the biometrics community. The repeated scans of biometric data, such as human fingerprints, are prone to errors and deviations that must be corrected. Hence, schemes like the Fuzzy Extractor are also suitable for the key generation from noisy PUF-data. Even though an earlier scheme for error correction was already published in 1999 [JW99], the Fuzzy Extractor triggered a new development in the PUF community. The first known application of the Fuzzy Extractor to PUFs was published by Škorić et al., who applied it to an optical PUF [ŠTO05].

In 2006, Tuyls et al. applied the Fuzzy Extractor to the Coating PUF, a capacitive PUF protecting on-chip components [TSŠ+06]. The Coating PUF is formed by randomly arranged particles in a protective coating. A tamper event destroys the original makeup of the particle structure, which is detected by the capacitive measurement.

Another well-known Physical Unclonable Function, the Ring Oscillator PUF, was proposed by Suh and Devadas for the low-cost authentication of individual ICs [SD07]. However, the extraction of unique properties from oscillating circuits had already been proposed by Gassend et al. [GCvDD02]. The Ring Oscillator PUF consists of many ring oscillators, each oscillating with a different frequency. The challenge selects different ring oscillator pairs whose frequencies are compared to obtain the PUF-response.

In 2007, Guajardo et al. build a PUF based on SRAM cells [GKST07]. At power up, each SRAM cell will yield a consistent startup value, which is, however, influenced by manufacturing variation, hence, introducing randomness. Guajardo et al. derived a PUF-secret from the SRAM PUF, and investigated its behavior under thermal influences. An idea similar to the SRAM PUF had already been described by Layman et al. in 2002 [LCNR02]. More recent memory-based PUFs make use of small variations in nanoelectronic devices to derive a PUF-secret [GRAS+16].

In the following years, further PUF constructions were proposed and implemented, e.g., [SHO07, MTV08, SSK12, CCL+11, BNCF13, MBC16, HAP09], including the publication of large PUF datasets for well-known PUF constructions [WBG17, MCMS10, HWGH18, SGC+21]. Apart from authentication, another use case for PUFs is tamper protection, which I discuss in Subsection 2.3.4.

### 2.3.3 Attacks on Physical Unclonable Functions

Since the early days of Physical Unclonable Functions, it was known that multi-challenge PUFs are generally susceptible to modeling attacks. However, with a large number of emerging PUF constructions, mathematically cloning PUFs through machine learning became increasingly interesting.

### Machine Learning Attacks

Already in 2002, in one of the initial publications on PUFs, Gassend et al. labeled modeling attacks of the proposed delay-based PUFs as one of the most relevant attacks [GCvDD02]. They argued that if an attacker knew the physical parameters defining the PUF, simulation and cloning the system would be feasible. Arbiter-based PUFs, in particular, can be described by a linear model. Hence, the attacker can use machine learning to build a software clone of the PUF circuit that could predict the responses associated with a certain challenge [LLG+05].

In 2008, Majzoobi et al. demonstrated practically that said linearity of the system is exploitable and that the delay-based PUFs could be broken [MKP08]. As a countermeasure, they proposed introducing additional non-linearity, leading to the interleaved XOR Arbiter PUF.

Two years later, Rührmair et al. showed that various types of Arbiter and Ring Oscillator PUFs can be broken through machine learning given a subset of challenge-response pairs [RSS+10]. However, they stated that the attacks were not feasible for an exponentially large set of challenge-response pairs and single-challenge PUFs. The attacks were replicated and further developed by Tobisch and Becker, proposing a more efficient implementation [TB15].

In the following years, machine learning attacks on different types of multi-challenge PUFs were extended towards increasing, and even exponential, challenge-response pairs [RSS+13, Bec15b]. A further goal was to reduce the attack complexity [SNMC15]. Even for PUF constructions that were considered difficult to model, e.g., the Bistable Ring PUF [CCL+11] or the Interpose PUF [NSJ+18], machine learning attacks found a corresponding mapping from challenges to responses [SH14, XRHB15, GTFS16, CMH20].

Additionally, machine learning attacks have been theoretically analyzed and optimized to reduce the time required to learn a PUF [GTS15, GTS16]. Even though new PUF constructions and extensions to well-known PUF designs have been proposed, the future of multi-challenge PUFs remains uncertain due to continuous progress within the field of machine learning.

### Side-Channel Analysis

Apart from machine learning attacks, the second class of relevant attacks for PUFs is side-channel attacks. Side channels can leak information about the PUF and corresponding CSPs through, for instance, the observed power consumption or electromagnetic radiation of the PUF and its post-processing operations. Karakoyunlu and Sunar [KS10] performed one of the first side-channel attacks in the context of PUFs. The authors presented two power side channel attacks aiming to extract the PUF-secret from a secure sketch and fuzzy extractor implementation with BCH and Reed Solomon codes.

Merli et al. [MSSS11a] conducted one of the first electromagnetic (EM) side-channel attacks on PUFs. They targeted a Field Programmable Gate Array (FPGA) implementation of Ring Oscillator PUFs with an 8-bit PUF-response. By measuring the EM emission of the Ring Oscillator PUF, they obtained the PUF-response bits.

In the same year, Merli et al. conducted a second EM side-channel attack on an FPGA Fuzzy Extractor prototype [MSSS11b]. Their proof-of-concept implementation showed that the EM side channel leaks information about the cryptographic key. In 2013, Merli et al. targeted the error correction module of a secure sketch through differential power analysis and helper data manipulation. [MSS13]. They were able to extract single bits of the PUF-key. Tebelmann et al. later extended the attack  by combining electromagnetic side-channel analysis with the parallel manipulation of helper data and an advanced correlation method. Through this, they extracted the complete key of a Fuzzy Commitment scheme.

Rührmair et al. and Mahmoud et al. performed power and timing side-channel attacks on Arbiter PUFs [RXS$^+$14, MRMK13]. Since the side channel analysis alone was insufficient to determine the Arbiter PUF outputs, they tailored the attack to machine learning algorithms, decreasing the attack's complexity.

Delvaux and Verbauwhede exploited a repeatability side-channel, where repeatability refers to the PUF's short-term reliability affected by noise, to successfully model an Arbiter PUF without any additional machine learning algorithms [DV13]. Furthermore, they proposed accelerating the attack by triggering evaluation faults through environmental changes [DV14b].

Also, memory-based PUFs were shown to be susceptible to side-channel attacks. In 2015, Zeitouni et al. recovered the PUF-key by observing time-based and voltage-based remanence decay of SRAM cells, through which they determined the startup pattern of the SRAM PUF [ZOW$^+$16]. In 2018, Anagnostopoulos et al. showed that temperature-induced data remanence could be used to manipulate cryptographic keys produced by SRAM PUFs [AAR$^+$18].

Another type of side channel is Photonic Emission Analysis (PEA). Tajik et al. showed that timing-based PUFs are vulnerable to photonic emission. This enabled them to measure specific PUF-delays and characterize different Arbiter PUFs [TDF$^+$14, TDF$^+$17].

A more recent attack performed by Tebelmann et al. showed that through power and electromagnetic analysis, the response of a Loop PUF could be fully recovered [TDP20]. Other recent work focuses on EM-based side channel attacks on Ring Oscillator PUF implementations [TPI19, SF19].

Since side channel analysis alone often does not suffice to determine all PUF-responses accurately, a variety of hybrid attacks was proposed to overcome this drawback. Several of these attacks were performed on Arbiter PUFs and their derivatives [RXS$^+$14, MRMK13, BK14, GKST15]. Becker and Kumar combined both a power side channel and a fault attack with a machine learning algorithm to target delay-based PUF designs [BK14]. They incorporated additional reliability information about the PUF to obtain the PUF model. Ganji et al. combined photonic side channel analysis and a lattice-based reduction technique to compromise the security of XOR Arbiter PUFs [GKST15]. Their approach does not require access to challenges or responses.

### Further Attacks

Besides creating a mathematical PUF model, PUFs can also be physically cloned. This was shown for memory-based PUFs by Helfmeier et al., who physically cloned an SRAM PUF by altering the circuitry with a Focused Ion Beam [HBNS13]. The cloned circuit produced the same response as the original SRAM PUF. In the same year, Nedospasov et al. read out the memory content of an SRAM PUF in a backside laser attack [NSHB13].

Nguyen et al. predicted the responses of a Ring Oscillator PUF corresponding to a specific challenge through a cryptanalytic attack [NSCM15]. Their attack required access to specific challenge-response pairs and helper data.

Many key generation schemes for PUFs require the generation of public helper data (see Section 2.4). Exploiting helper data manipulation provides additional information about the PUF [DGSV15, DV14a, TPS17]. Through helper data manipulation, Delvaux and Verbauwhede attacked the key generation of a Ring Oscillator PUF. They demonstrated that a partial to full key recovery is possible through helper data manipulation and observation of failure rates [DV14c].

Besides PUF constructions and key generation schemes, challenge-response protocols based on PUFs have been successfully attacked via quadratic attacks [RD12] and machine learning [Bec15a].

As already discussed, multi-challenge PUFs, like, e.g., Arbiter PUFs, are susceptible to machine learning attacks. Even though different new PUF constructions and extensions of well-known designs have been proposed, the long-term security of multi-challenge PUFs is threatened by new and improved machine learning attacks.

Single-challenge PUFs are not vulnerable to machine learning attacks; however, SRAM PUFs, as proposed by Guajardo et al. [GKST07], can be physically cloned by duplicating the circuitry through Focused Ion Beams or memory read out via laser attacks. Hence, instead of integrating SRAM PUFs, in many current applications, replacement via a One-Time-Programmable Memory might already be a good solution for storing CSPs. However, memory-based PUFs could regain importance in the future due to increasingly smaller chip sizes, which might lead to a reduction or even removal of large memory spaces.

More promising applications of PUFs are constructions protecting against tampering attempts. In the following, I discuss different PUFs applied in the context of tamper protection.

## 2.3.4 PUF-Based Tamper Protection

Table 2.1 provides an historical overview of PUF-based tamper protection. One of the first secure architectures incorporating a PUF is the AEGIS secure processor architecture [SOD07, SOSD05]. AEGIS was proposed in 2005 by Suh et al. as a single chip secure processor authenticated by a delay-based PUF. The authors considered opening and tampering with the chip while running as "prohibitively expensive". However, analysis methods and hardware attacks have developed significantly, constituting a severe threat to the AEGIS system.

**Table 2.1:** Historical overview of PUF-based tamper protection.

| **Year** | **PUFs in the Context of Tamper Protection** |
|---|---|
| 2005 | AEGIS secure processor architecture [SOD07, SOSD05] |
| 2006 | Protective capacitive coating PUF [TSŠ+06] |
| 2012 | Optical PUF in the context of smartcards [EFK+12] |
| 2014 | Optical waveguide polymer PUF [SFIC14] |
| 2015 | Secure architecture based on the optical waveguide polymer PUF [VNK+15] |
| 2018 | Capacitive PUF-based envelope (B-TREPID) [IOK+18] |
| 2019 | Capacitive PUF-based enclosure (COVER) [ION+19] |
| 2020 | Electromagnetic enclosure PUF [TZP20] |
| 2021 | Switched Capacitor PUF standard cell [ZHW+21] |
| 2021 | Anti-Tamper Radio [STZP21] |

Another secure architecture proposed by Vai et al. [VNK+15] applied the optical waveguide polymer PUF [SFIC14] as the protective cover of a PCB. Light from LEDs is sent through a polymer waveguide covering the PCB's top area. A change in the detected light patterns indicates the occurrence of a tamper event. The secure architecture prototype does not cover the PCB's edges or the bottom area. Prior to the optical waveguide polymer, Esbach et al. had already applied optical PUFs in the context of smartcard security [EFK+12].

A further electromagnetic PUF in the context of tamper protection was proposed by Tobisch et al. [TZP20]. Their approach is based on the manufacturing variations of electromagnetic-sensitive sealing material. The channel state information between different antennas is measured within the sealing. The wavelength of the radio channels depends on the sealing material's manufacturing variations, yielding a measurable PUF-response. A similar approach by Staat et al. investigated radio wave propagation in an enclosed system with a complex geometry [STZP21]. Their Anti-Tamper Radio (ATR) monitors the wireless signal propagation within a metal case and triggers an alarm if the radio signal response is altered. The ATR can detect 16 mm needle insertions with a diameter of 100 µm.

For the protection of components on-chip, Tuyls et al. proposed a capacitive coating PUF [TSŠ+06]. The protective coating containing randomly arranged particles is added to the top area of an IC. A capacitive measurement of the coating determines whether the particle structure is intact. A more recent approach published by Zhang et al. aims at protecting electrical circuits through Switched Capacitor (SC) PUFs [ZHW+21]. The authors integrated an anti-invasive attack protection into the SC PUF, from which they generated a key. The protective element is implemented through metal meshes covering the corresponding sensitive circuits. A probing attack alters the parasitic capacitance between the metal meshes and changes the reproduced key.

PUF-based capacitive envelopes and enclosures have been developed in recent years by Immler et al. [IOK+18, ION+19] to protect larger PCB areas or entire devices. The enclosures contain a mesh of overlapping electrodes forming small capacitors. Minuscule capacitive differences due to manufacturing variations determine the PUF-response. The destruction of the mesh through a tamper event alters the original capacitances and hence, the PUF-response. In the following, I give an overview of previous work on capacitive PUF-based security enclosures.

## 2.4 Capacitive PUF-Based Security Enclosures

This section provides an overview of previous work on capacitive PUF-based security enclosures, including the overall enclosure system, post-processing of the PUF-response, and previously conducted attacks.

### 2.4.1 The Capacitive Enclosure

The purpose of the capacitive PUF-based enclosure is to cover an entire device to protect it from physical manipulations. Over time, two enclosure versions were developed: B-TREPID [IOK$^+$18] and COVER [ION$^+$19], which are depicted in Figure 2.1. B-TREPID is a protective envelope that is wrapped around the device within a casing, while COVER

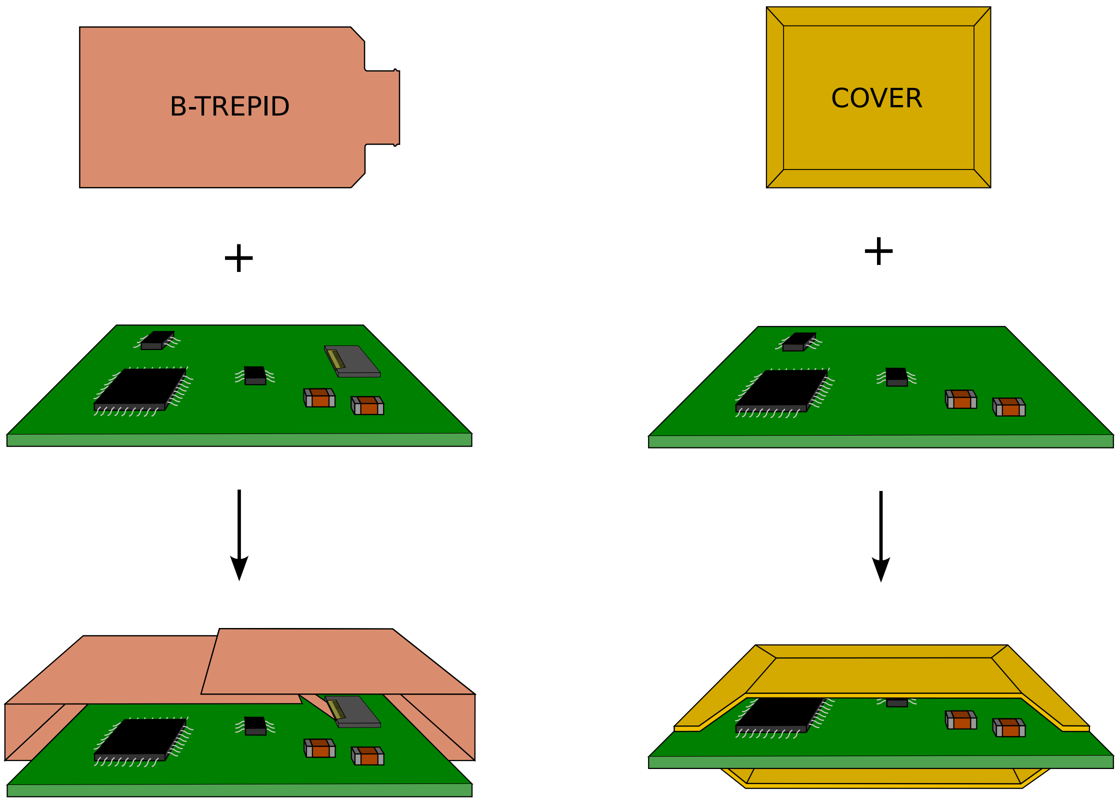


**Figure 2.1:** Simplified schematic of B-TREPID (left) and COVER (right).

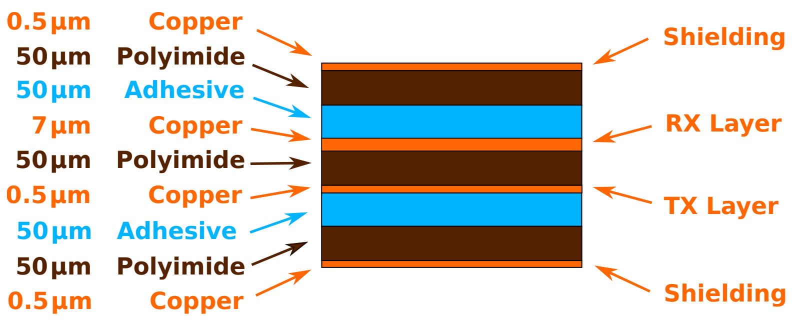


**Figure 2.2:** The layer stack of the capacitive enclosure (B-TREPID).

is a non-flexible enclosure that covers the top and bottom layer of the PCB, leaving the edges unprotected.

Both enclosures are built up of several layers. Figure Fig. 2.2 depicts the layer stack for B-TREPID. Two layers of meander-shaped copper (Cu) electrodes (RX and TX) are separated by an insulating layer of polyimide (PI), forming small capacitors. Since alternating electric fields disrupt the capacitive measurement, a Cu-shielding was attached to the top and bottom of the envelope [Obe19, IOK+18]. The electrode layers are formed by sputtering Cu onto the PI substrate. In order to create the conductive interconnection between the electrodes, vias are formed through laser ablation. This requires the thickness of the Rx electrode layer to be increased to 7 µm through a semi-additive galvanic process. The overall thickness of the envelope layer stack amounts to approximately 0.25 mm.

The envelope consists of 16 TX (top) and 16 RX (bottom) electrodes with a width and distance of 100 µm, as depicted in Figure 2.3. Due to their orthogonal arrangement, the electrodes overlap and form small capacitors whose capacitance varies due to manufacturing variations, hence, creating a PUF. Measuring one TX electrode against one RX electrode results in 256 *absolute* capacitances that do, however, depend on global manufacturing variations [Bri04, IHOS17]. Since this dependency makes them unsuitable as a PUF-response, the *absolute* capacitances are subtracted to form 128 *differential* capacitances in the range of $[-73\,\text{fF}, 73\,\text{fF}]$ with a maximum entropy of 560 bits [IOK+18]. These *differential* capacitances constitute the PUF-response and are obtained from measuring two TX electrodes — forming a TX pair — against one Rx electrode. The PUF-response serves as input for the key generation process from which a Key-Encryption-Key (KEK) is derived. The advantage of a PUF-based solution is that the KEK is not stored in the system but continuously reproduced from the envelope. Hence, the system can be powered down without requiring a battery for continuous monitoring.

Due to the structure size of 100 µm, the envelope is designed to withstand drilling attacks with a diameter of 250 to 300 µm. The electrode mesh is depicted in Figure 2.3. Experiments have shown that these attacks destroy two traces of the electrode mesh, affecting at least 32 of the 128 *differential* capacitances [ION+19]. This leads to a destruction of 80 of the maximum 560 bits of entropy [ION+19, IOK+18].

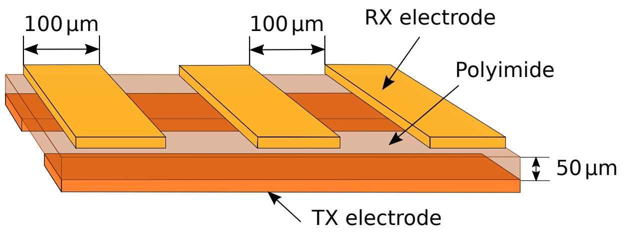


**Figure 2.3:** Schematic cross section of the envelope's mesh with RX and TX electrodes, and polyimide layer.

To measure the 128 differential capacitances and perform an integrity check of the electrodes, Obermaier developed a discrete measurement circuit [OIHS18, Obe19]. The differential capacitance, in this case, is obtained by measuring the capacitance of two TX electrodes against one RX electrode. Subtracting two absolute capacitances from one another is highly error-prone and can not determine the minuscule differential capacitances in the femtofarad range. Due to an antiphasic excitation signal, Obermaier performed the subtraction ”within“ the enclosure, reaching high accuracy. It takes 390 µs on average to measure a single differential capacitance, also referred to as a ”node“. Measuring the entire enclosure requires 50 ms.

However, due to the large size of the discrete measurement circuit, integrating it into an HSM prototype becomes difficult. A smaller ASIC of approximately 5 mm $\times$ 5 mm was developed to overcome this issue [FIU+18], which can be embedded into the envelope. The time for the measurement is significantly reduced since the ASIC measures 16 differential capacitances — each between two RX electrodes and one TX electrode — in eight parallel channels. The ASIC quickly detects attacks at runtime by periodically monitoring the electrodes. The ASIC provides several security features besides the capacitive measurement and integrity check. Two redundant alarm signals, a dynamic Pulse Width Modulation (PWM) signal and a static high signal, impede external manipulation. The PWM signal acts as a heartbeat, whose frequency is generated by the ASIC's Random Number Generator (RNG). Upon a tamper event, the static high signal goes to low, and the PWM signal stops. A tamper event corresponds to shorted or interrupted traces or out-of-bounds capacitances. The ASIC also has sensors for voltage and temperature monitoring.

PUF-based security enclosures, and the ASIC for capacitive measurement, are the foundation of FORTRESS, the *FORtified Tamper-Resistant Envelope with an Embedded Security Sensor*, which I discuss in Chapter 3.

### 2.4.2 Post-Processing

In order to obtain a key from the analog PUF-response, several steps are necessary. The ASIC's Analog-to-Digital Converter (ADC) provides integers in the range [-16383, +16383] that correspond to the 128 PUF values. The discrete measurement circuit maps the full-scale range of $[-134\,\text{fF}, +134\,\text{fF}]$ to integers in the interval $[-10000, +10000]$, where one point corresponds to a digital resolution of 13.4 aF. Previous measurements of the PUF-response reveal a Gaussian distribution of the differential capacitances [IOK+18, ION+19].

Several steps are necessary before generating a key from the analog PUF-response. These steps, as depicted in Figure 2.4, are as follows:

1. Shift of TX/RX groups (normalization)
2. Generation of quantization mapping
3. Generation of analog helper data

The first step after the capacitive measurement is the normalization of the PUF-response. In the case of the ASIC, the differential capacitance is obtained from measuring two RX traces (RX group) against all other TX traces. In contrast, the discrete measurement circuit measures two TX electrodes (TX pair) against one RX electrode. *Global* manufacturing variations in the electrode thickness lead to offsets in the raw differential capacitances of some of the TX or RX groups. Hence, each group is shifted by a particular value. In the first post-processing step, this global dependency is removed by subtracting the offset and forcing the group's mean to zero. By subtracting each group's mean separately, the overall PUF-distribution is also affected. In general, I observed that the standard deviation of the PUF-distribution is reduced, as depicted in Figure 2.4. The side effect of this group shift is that if an attack changes only a single value, all other values in the group shift as well. This increased tamper-sensitivity enhances the overall security [OIHS18]. I will refer to the first step as "normalization".

In the second step, the quantization intervals for the PUF-response are generated. The number of bits per PUF-value depends on the measurement uncertainty. Due to noise and aging effects, the points at the edge of an interval are prone to shift into the neighboring interval. To reliably reproduce the PUF-key, so-called *analog helper data*¸ are generated in the third step. The analog helper data stored in Non-Volatile Memory (NVM) represent the offsets from the interval centers that are subtracted from each PUF-value. Since these data stem from the post-processing of the analog PUF-response, they are referred to as *analog helper data* to distinguish them from the helper data created during key generation.

After the quantization of the PUF-response and the application of analog helper data, the PUF-key is generated or reproduced, respectively. Although the analog helper data

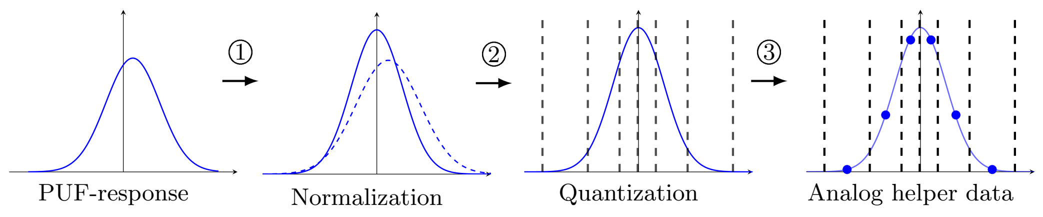


**Figure 2.4:** Post-processing steps of the measured PUF-response, including TX/RX group shift (normalization), quantization, and generation of *analog helper data*.

reduce the quantization error, the PUF-response is still affected by noise and environmental changes. Hence, during the key reproduction, error correction codes are applied to enhance reliability.

### 2.4.3 Key Generation

After the described post-processing steps, a key is generated from the quantized PUF-response. The PUF-secret is embedded into the secure key chain as the KEK. The successful verification of the KEK leads to the decryption of the protected device. If the KEK can not be verified correctly, the decryption fails, and an alarm is triggered, which leads to the zeroization of all CSPs.

Various schemes have been proposed for generating (and reproducing) a key from the PUF-response [DRS04, JW99, HYP15, MB17]. Since the PUF-response is noisy and error-prone, these schemes combine the key generation with an error correction code to ensure the correct reproduction of the key. The first key generation schemes applied to Physical Unclonable Functions originated in the field of biometric authentication [DRS04, JW99], where keys are created from biometric features that are subject to random noise.

The generation or reproduction of a key from the PUF-response includes several steps. Figure 2.5 shows a high-level overview of the building blocks for the most common key generation schemes. One of the main building blocks is an encoder and decoder to correct errors $\epsilon$ from noise or environmental changes. During key generation — also referred to as enrollment — most key generation schemes use so-called helper data $W$ to construct a codeword $C$ from the PUF-response $X$. The corresponding Helper Data Algorithm (HDA) maps the PUF-response to a codeword resulting in the helper data $W$. The helper data are stored in an NVM on the device and hence, in general, must be considered public. In general, information about the PUF-secret can be gained through helper data manipulation attacks, where the attacker observes if the key $K$ remains correct after manipulating the helper data $W$ [MSS13, DV14a, TPS17].

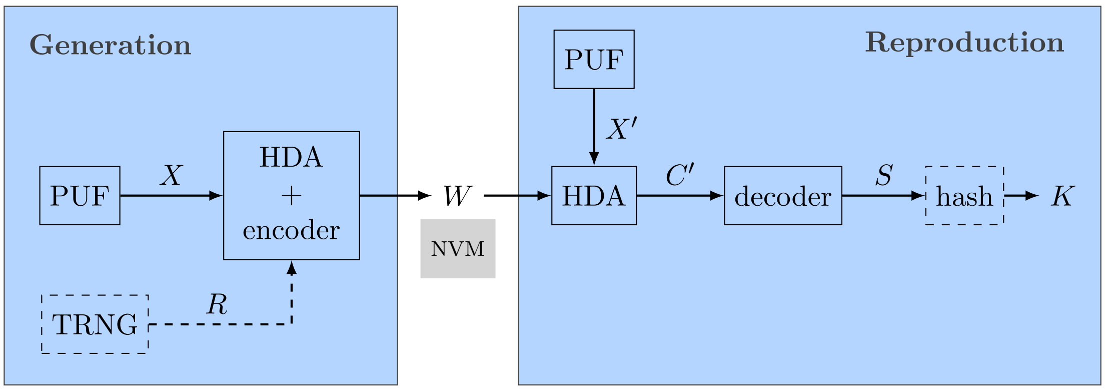


**Figure 2.5:** Exemplary key generation and reproduction. Optional blocks are depicted through dashed lines.

Some schemes also require an additional random number $R$ from a TRNG [JW99, DRS04]. Others use a portion of the PUF-response as a random number [HYP15] or adjust the code construction such that the PUF-response corresponds to a codeword [MB17]. Depending on the chosen scheme, the PUF-secret $S$ either corresponds to the random number $R$, the PUF-response $X$, or a portion of the PUF-response. For the reproduction of the key $K$, the PUF-response $X' = X + \epsilon$ is measured, which, due to environmental changes and noise, will differ from the original PUF-response $X$. The PUF-response $X'$ and the helper data $W$ from NVM yield the codeword $C' = C + \epsilon$, which is decoded to the secret $S$. Depending on the scheme, an additional step is required after the decoding to ensure sufficient entropy of the PUF-key $K$ and reduce helper data leakage. In most cases, this is achieved through a hash function.

Alternative key generation schemes perform the encoding considering noise properties of the PUF to ensure reliable reproduction [YD10, HMSS12, HWRL$^{+}$13, YHD15]. Furthermore, code concatenation can be applied to reduce the decoding complexity [For65a, Hil16].

Various codes were proposed and applied in the context of PUFs to improve and optimize the error correction, with each approach pursuing a different design goal. Since PUFs are often integrated into embedded devices as a root of trust, the size and performance of the implemented codes are crucial parameters. Therefore, many implementations of error correction in the context of PUFs focused on optimizing hardware resources, decoding complexity, and implementation overhead of Bose Chaudhuri Hocquenghem (BCH), Reed Solomon, or Reed-Muller Codes [BGS$^{+}$08, MVHV12, HKS20, MHK$^{+}$19, PMB$^{+}$15]. In addition, the implementation overhead can be further reduced by including reliability information about the PUF [MTV09a, MTV09b, MPB18, HOSB16].

Besides implementation efficiency, many publications focus on reducing secrecy leakage in key derivation schemes either by removing bias in the PUF-response [MLSW16, BY19, IHL$^{+}$19] or by targeting helper data leakage [CW19, BY21]. An alternative approach to tackle helper data leakage was published by Müelich and Bossert, who proposed a novel secure sketch that does not require additional helper data [MB17]. A more recent approach in the context of PUFs was presented by Chen et al., who applied polar codes to SRAM PUFs in order to reduce the failure probability of the error correction code.

As I will show in more detail in Chapter 5, the capacitive PUF-based enclosure requires an error correction scheme that ensures a high tamper-sensitivity while correcting changes in the PUF-response due to environmental effects or noise. Limited Magnitude Codes (LMC) [JL12, IU19], which I discuss in more detail in Chapter 3, are a straightforward method with sensitivity to large interval shifts that occur during an attack, correcting only errors of a small magnitude. LMCs provide a simple approach to limit the number of corrected errors to only small interval shifts. To include more complex error patterns and model both environmental and attack effects, I present an error correction scheme in Chapter 5, implementing a wiretap channel through $q-$ary polar codes.

### 2.4.4 Attacks on Security Enclosures

The capacitive enclosure system has been subjected to various attacks. Immler et al. showed that X-ray inspection of the enclosure reveals the layout structure and design [ION$^{+}$19]. Preventing these reverse-engineering attacks is hardly feasible. Hence it has to, in general, be assumed that the attacker knows how the electrodes are arranged within the enclosure. Since the device is not directly accessible, physical attacks on security enclosures are of particular interest from a security perspective [Int15]. Unfortunately, hardware security testing procedures following Common Criteria standards are a well-kept secret, and the overall certification score often relies on hidden or obscure countermeasures. To counteract the secrecy surrounding HSMs, Obermaier and Immler [OI18] analyzed several non-capacitive enclosure systems through partial or complete disassembly. They determined the size of the enclosure mesh and the applied security mechanisms. Hence, the security of an enclosure system should not rely on hidden countermeasures or obfuscation since an attacker can purchase a decommissioned device and reverse-engineer it.

A second class of attacks relates to bypassing the electrode mesh aiming to partially remove it and, thus, access conductive traces or pins to gain critical information. Battery-backed enclosure systems verify that the protective mesh is intact by continuously measuring the electrodes' resistance [OI18]. However, since these enclosures are not based on the intrinsic physical properties of the mesh, bypassing the electrode could be achieved by successively replacing a portion of the electrode with a corresponding resistor. In the case of the capacitive enclosure, a bypass attack is more complex, as shown by Obermaier [Obe19]. Obermaier built a specific current probe to duplicate the small RX currents during measurement. He attached fine wires to a severed RX electrode and read out the RX current signal, which he redirected back into the enclosure. The attack was not detected during capacitive measurement, which could potentially enable an attacker to remove a portion of the enclosure. I discuss a countermeasure against the bypass attack in Chapter 4.

One of the most critical threats is attacking the enclosure via mechanical drills or energy probes, such as lasers, aiming to gain access to critical components and connections. The enclosure's susceptibility to drilling depends on the electrode mesh size. In the case of the capacitive enclosure, the electrode mesh was designed to withstand drilling attacks with a diameter of 300 µm. Thus, with an electrode with and distance of 100 µm, the drill destroys two electrodes, which affects 32 of the 128 differential capacitances that constitute the PUF-response. However, micro-drilling attacks with diameters below 300 µm are feasible and pose a potential threat to the enclosure system. I discuss micro-drilling attacks and countermeasures in Chapter 4.

Apart from invasive and destructive attacks, there are also non-invasive attacks that could potentially affect the security of the enclosure system. Non-invasive attacks based on machine learning that aim at mathematically cloning the PUF are relevant for multi-challenge PUFs [RSS$^{+}$10, Bec15b, SH14]; however, they do not pose a threat to single-challenge PUFs such as the enclosure. Sensor monitoring mostly mitigates side-channel analysis, with one exception: In Chapter 4, I show that magnetic probing of the SPI-interface between the ASIC and the system's main microcontroller leaks information about the PUF-response. Glitching attacks, such as clock or voltage glitching, are counteracted by the system design. Since the enclosure inhibits physical access to the clock, clock glitching attacks are of low relevance to the enclosure system, as well as voltage glitching attacks, which are prevented by large capacitors as part of the power supply and voltage stabilizers. Faults induced through alternating electric fields are mitigated through the envelope's shielding. Even though magnetic fields permeate the enclosure, they are unlikely to influence the capacitive measurement since they are counteracted by the narrow excitation frequency and canceled out through the meander structure of the enclosure mesh. However, particle radiation might permeate the enclosure and disturb critical components or operations, which I discuss in Chapter 6.

### 2.4.5 New Contributions Presented in This Thesis

Within the scope of this thesis, I investigate the tamper-sensitive design of PUF-based capacitive security enclosures and take further steps toward the operational deployment of this PUF-based technology.

Previous work focused on improving the enclosure design while optimizing the capacitive measurement and provided a statistical analysis of the PUF-response [ION$^+$19, IOK$^+$18, Obe19, OIHS18, Imm19]. In Chapter 3, I bring all these components together, introducing FORTRESS, the *FORtified Tamper-Resistant Envelope with an Embedded Security Sensor*. FORTRESS incorporates the ASIC embedded into the enclosure and includes a full key generation tailored to the enclosure, considering all post-processing steps. The hardened FORTRESS software was extended through a secure life cycle covering critical parts of the supply chain.

The security of the enclosure system depends on its attack susceptibility. As discussed in Subsection 2.4.4, drilling attacks and bypassing of electrodes belong to the most critical physical attacks on the enclosure system. I focus on both attacks in Chapter 4 and propose countermeasures. Furthermore, I demonstrate a new attack on the enclosure exploiting the side channel leakage of the SPI-interface connecting the ASIC and the microcontroller. I show that the PUF-response can be read out through magnetic probing and provide an analysis of how external electric and magnetic fields affect the enclosure system.

The error correction for the enclosure has to reconcile two opposing goals. On the one hand, it has to correct errors from environmental influences to achieve a reliable reproduction of the PUF-secret; on the other hand, it may not correct errors stemming from an attack since this would compromise the enclosure's tamper-sensitivity. However, most error correction codes for PUFs do not focus on tamper-sensitivity. To limit the magnitude of the corrected errors, Immler and Uppund proposed using Limited Magnitude Codes (LMCs) for the enclosure [IU19, JL12]. LMCs limit the error correction to the Least Significant Bits (LSBs), thus, only allowing changes in the PUF-response within neighboring intervals. However, when applying LMCs, a suitable error magnitude has to be chosen. Furthermore, these codes require a base change that restricts the choice of possible error correction parameters. Nevertheless, using Limited Magnitude Codes is a simple approach to improve the tamper-sensitivity of the enclosure. To model more complex error patterns and to enhance the tamper-sensitivity of the PUF-response, I present a wiretap channel implementation for the capacitive enclosure in Chapter 5.

The enclosure prevents physical access to the protected device, which reduces the risk of glitching attacks. Also, electromagnetic interference is unlikely to affect the capacitive measurement due to the measurement principle and enclosure design. However, as I will discuss in Chapter 6, particle radiation can permeate the enclosure and cause faults that negatively affect critical operations. In Chapter 6, I first demonstrate the effects of transient faults injected into critical operations through a realistic laser fault attack on flash memory. Then, I show how searching for vulnerabilities in the firmware of embedded devices can be facilitated through ARCHIE, an architecture-independent fault emulation framework. With ARCHIE, I analyze the fault susceptibility of the enclosure system and discuss possible countermeasures.

# Chapter 3

# Fortified Tamper-Resistant Envelope with an Embedded Security Sensor

Batteryless PUF-based security enclosures were developed to cover entire devices with a protective envelope without the drawbacks of a continuous power supply. As discussed in Section 2.4, previous work on capacitive PUF-based enclosures took the first steps towards a proof-of-concept implementation by showing that the minuscule capacitances in the femtofarad range can indeed be measured and that the measured PUF-response is Gaussian distributed. This chapter aims to integrate the envelope and the ASIC into a prototype HSM and develop it further into FORTRESS - the *FORtified Tamper-Resistant Envelope with an Embedded Security Sensor*.

The assembly of FORTRESS comprises several steps. In Section 3.1, I focus on embedding the ASIC into the capacitive envelope, which becomes directly accessible through a digital interface. Furthermore, I discuss the components of the FORTRESS PCB. This is followed in Section 3.2 by a discussion of the second generation EKMS, the software of FORTRESS, with Operating System (OS) hardening, extended crypto-functionality, and a secure life cycle covering essential supply chain aspects. In Section 3.3, I describe a full key generation scheme that I tailored to the security objectives of the capacitive enclosure and the PUF post-processing. I summarize the results of the FORTRESS assembly in Section 3.4.

I presented the results in this chapter at the 18th Annual International Conference on Privacy, Security & Trust (PST2021). Further details are published in the IEEE Conference Proceedings [GOFK21].

## 3.1 The Assembly of FORTRESS

In this section, I give an overview of FORTRESS and present its basic components.

### 3.1.1 Overview

Figure 3.1 shows the FORTRESS PCB within a casing enveloped by the capacitive enclosure. The envelope of 18.5 cm × 9 cm is wrapped around a PCB of 5 cm × 6 cm together with a Flat Flexible Cable (FFC) for external communication. Furthermore, the enclosure is covered by a protective potting material to hamper access to the envelope.

The FORTRESS software, the so-called EKMS, runs on a microcontroller located on the PCB. The EKMS is an intermediate firmware between the ASIC and the host system, residing within the physical boundary of the envelope. The ASIC is mounted onto the capacitive enclosure and provides a digital interface, reducing the envelope's connector size from 80 to 20 pins.

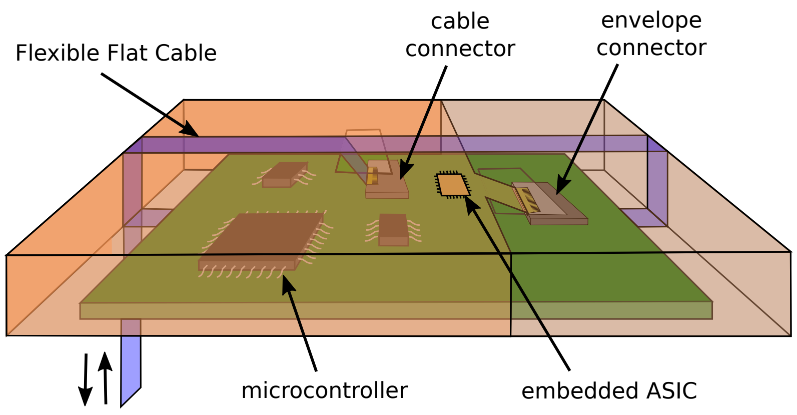


**Figure 3.1:** Schematic of the PCB within a casing, enveloped by the enclosure. For external communication an FFC is wrapped around the casing together with the envelope.

### 3.1.2 Envelope Manufacturing and ASIC Mounting

The envelope is fabricated via lithographic patterns that allow the production of a complex mesh of traces with 100 µm width. Due to the scalability of the manufacturing process, even smaller structures are, in general, feasible. The first layer of the RX electrodes is formed by sputtering copper onto the polyimide substrate. Subsequently, the RX electrode layer's thickness is increased to 7 µm in a semi-additive galvanic process. This provides a defined stop interface creating the vias in the polyimide substrate for laser ablation. The TX electrode layer is sputtered onto the backside of the polyimide substrate, resulting in a layer thickness of approximately 600 nm. The electrode width and distance amount to 100 µm. Additionally, the contact pads for the ASIC are fab-

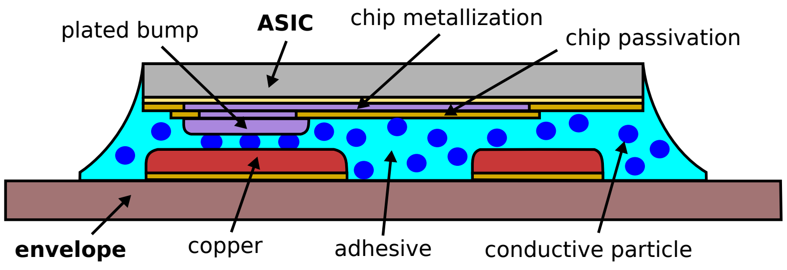


**Figure 3.2:** Result of ACA flip chip procedure (schematic), where the ASIC is mounted onto the envelope.

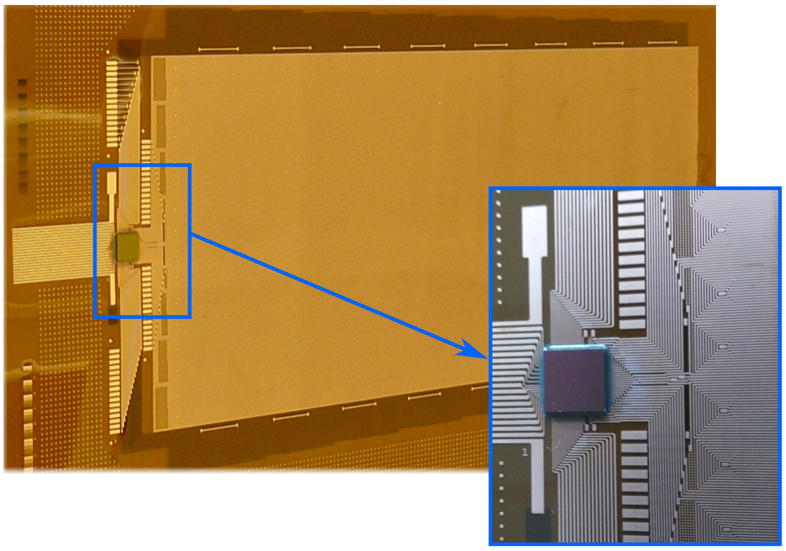


**Figure 3.3:** Result of ACA flip chip procedure, with a close up of the embedded ASIC. The bare electrodes are visible since the Cu-shielding has not yet been attached.

ricated. The envelope manufacturing process enables thin layers and small structures, which is an advantage over commonly available flexPCB technology with a layer thickness of approximately 12.5 µm.

After the fabrication of both electrode layers, the ASIC is mounted onto the envelope via Anisotropic-Conductive Adhesive (ACA) flip chip technology. In this process, the electrical interconnection between the ASIC and the enclosure is created through conductive particles dispersed in the adhesive. This is depicted in Figure 3.2. The ASIC is permanently bonded onto the envelope by applying a force of 30 N over several tens of seconds at a peak temperature of 220 °C.

Figure 3.3 depicts the enclosure with the embedded ASIC. In this case, the electrode mesh is still visible since the copper shielding with the second polyimide layer has not yet been attached to the envelope.

### 3.1.3 The FORTRESS Components

The envelope and the embedded ASIC were assembled and integrated into FORTRESS, a PUF-based prototype HSM. FORTRESS is comprised of several components that are depicted in Figure 3.4. The 5.5 cm×4.5 cm prototype PCB was designed specifically for FORTRESS, with light and temperature sensors, a power supply, and connectors for the envelope and external communication. The additional sensors enhance the security of FORTRESS by monitoring external influences, and triggering an alarm if a certain threshold is exceeded. An alarm is also triggered if the PUF-secret derived from the capacitive measurement is not correctly reproduced due to, for instance, a drilling attack. This leads to a zeroization of all CSPs. The modes of the FORTRESS system life cycle are discussed in more detail in Section 3.2.

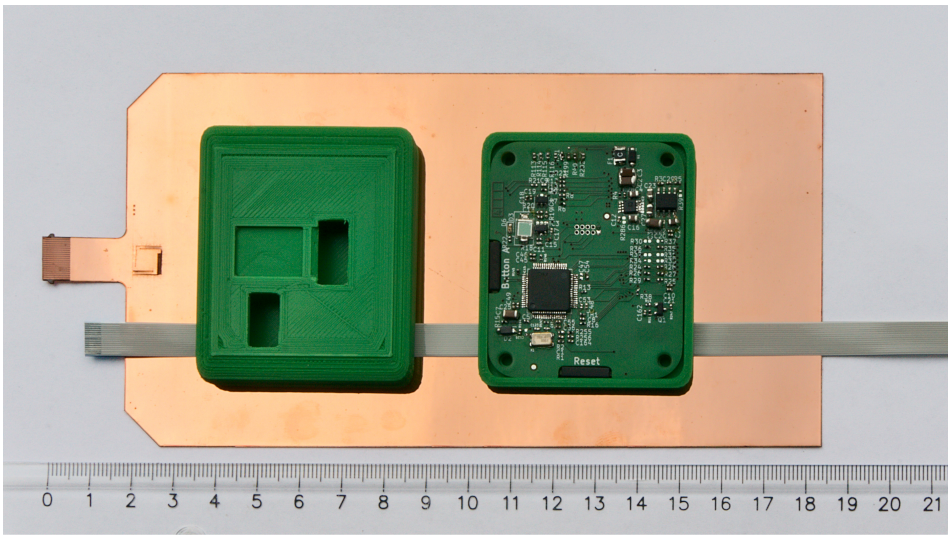


**Figure 3.4:** The components of FORTRESS.

The main microcontroller runs the FORTRESS software, the Embedded Key Management System EKMS, which implements the key management, key generation, and post-processing of the PUF-response. The EKMS is discussed in Section 3.2. It also orchestrates the measurement process and monitors all security sensors. The FORTRESS PCB is placed into a printed casing, which can be enveloped by the capacitive enclosure together with the 15-pin FFC for power and external communication. The ASIC and the envelope are connected to the PCB via an SPI interface (included in the 20-pin envelope connector).

The front side of the HSM, as depicted in Figure 3.5, includes the STM32F303 with an ARM Cortex-M4F microcontroller for the EKMS, infrared LEDs, and photodiodes for light detection during runtime, and a temperature sensor (NTC-thermistor). The enclosure's and external components' connectors are placed on the PCB's backside (see

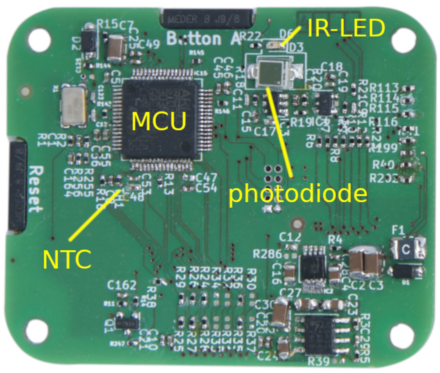


**Figure 3.5:** Frontside of the FORTRESS PCB.

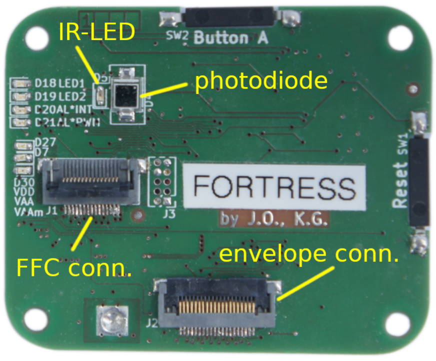


**Figure 3.6:** Backside of the FORTRESS PCB.

Figure 3.6). Reed switches were added to the PCB to contactlessly reset the HSM when placed into the casing.

To counteract blinding attacks, and thus, the opening of the casing, FORTRESS is enhanced by frequently pulsing infrared LEDs detected by photodiodes placed on both sides of the PCB. The alarm is triggered if the light intensity exceeds or drops below a certain threshold. The same process is triggered if an extraordinarily high or low temperature is measured, e.g., in case of a cold boot attack. Similarly, in case of undervoltage detected by voltage sensors, or sudden power failure, all critical data are deleted from volatile memory, and the system is securely shut down. This is enabled through an energy storage of several hundred microfarads, which carries sufficient energy for a spontaneous shutdown.

## 3.2 Second Generation EKMS

The second generation EKMS — the software of FORTRESS — orchestrates the measurement process and the monitoring of sensors; it processes the raw PUF-response and manages CSPs and their generation. Hence, the EKMS is a critical component of FORTRESS, whose security functionality was significantly hardened, extended, and refined. In this section, I give an overview of the EKMS architecture, key management, and OS hardening. I introduce a secure system life cycle concept covering different aspects of the supply chain, including transportation, in-field operation, and decommissioning.

### 3.2.1 Operating System Hardening

The first generation EKMS was based on a modified FreeRTOS 10 [OHHS18], which provides a relatively small and simplistic kernel, reducing the OS attack surface. FreeRTOS supports Memory Protection Unit (MPU) access permissions and two privilege levels (kernel and user space). Obermaier et al. extended the FreeRTOS by secure system calls [OHHS18]. In the original implementation, a user space task can raise its privilege level to execute the desired operation, thus allowing privilege escalation. Obermaier et al. implemented a preemption-compatible system call interface and handler taking over privilege handling of the user space task during the system call.

The second generation EKMS was extended for ARM Cortex-M-based cores by the following security features:

- **Limited data access permission**
  In the previous version of the EKMS, all data (functions and variables) were accessible from both kernel and user space unless otherwise specified. This concept was inverted, such that by default, data are specified as kernel-access-only. Hence, shared data accessible from user space has to be explicitly marked. This reduces the attack surface for, e.g., Return-Oriented Programming (ROP) and enhances the security by default concept.

- **Data execution prevention**
  In the original FreeRTOS implementation, the stack was executable, which enabled attacks based on stack overflows. The write-xor-execute approach was adapted to overcome this issue, allowing for a section to be either writable or executable. This was achieved by setting the execute-never (NX) bit for the stack and variable data regions.

- **Floating Point Unit (FPU) context isolation**
  In the original EKMS, FPU registers were only saved and restored upon FPU usage, which potentially leaked data into the subsequent task. This was counteracted by implementing FPU flushing for kernel task switches, where FPU content is only deleted if a transition from an FPU task to a non-FPU task occurs.

- **Dedicated system call stack**
  In the previous EKMS implementation, system calls were executed in the context of the invoking user space task. Thus, secret data accessed within a system call could end up in the user space task stack and leak confidential information. As a countermeasure, system calls dispatched the processing of high-security data to another task with its own stack. However, this affected the overall performance due to additional task switches. Therefore, in the second generation EKMS, a dedicated stack for system calls is automatically allocated, inaccessible by the user space task. The kernel was modified such that the stack pointer is switched when entering or leaving a system call.

- **Secure copy to / from user interface**
  The original FreeRTOS did not provide a kernel function for a secure write/read from/to user space. A function similar to the `copy_to / from_user()` of the Linux kernel was implemented to overcome this, which guarantees that any access to non-user-accessible regions creates a fault.

Further details on FORTRESS' OS hardening are published in the IEEE proceedings [OHHS18, GOFK21].

### 3.2.2 EKMS Architecture

FORTRESS consists of the envelope with embedded ASIC, the EKMS, and at least one host device, as depicted in Figure 3.7. The EKMS is the intermediate software between the ASIC and the host system. It controls the measurement and processing of the PUF-response, the key management, monitors security sensors, and provides basic cryptographic functionalities, such as encryption and decryption.

The second generation EKMS is depicted in Figure 3.8, which shows a division into the user application and cryptographic core functionalities. An example of a user application would be the decryption of the host device's firmware. In this case, the user application executes the task via the Crypto-API when triggered by the host system. The EKMS Core verifies the PUF, manages the generated keys, and decrypts the firmware. If data are sent over the Universal Asynchronous Receiver/Transmitter (UART)-interface, the EKMS user application handles the received data via the Secured-Communications-API. Hence, the host systems that reside within the envelope boundary can not access the PUF-response or functions related to its processing.

The EKMS implements a keystore that triggers the generation, reproduction, and deletion of CSPs that reside in NVM. Thus, the keystore manages all CSPs over the entire key chain. It also manages the secure system life cycle, which I discuss in Subsection 3.2.3.

A key generation algorithm generates a KEK from the PUF-response. This KEK is the root of the key hierarchy, as depicted in Figure 3.7. In the following, I will refer to the generation of all CSPs, including the KEK, as *enrollment*, while the *re-enrollment* describes the generation of only a new KEK. The re-enrollment process required the extension of the key chain, which enabled the implementation of the secure life cycle, including the compensation of aging effects. The repeated re-generation of the KEK after startup is referred to as *key reproduction*.

The first generation EKMS [OHHS18] provided only a primitive key generation scheme without full key management and re-enrollment capabilities. The KEK was directly used to encrypt CSPs. Hence, a change of the KEK required the decryption and re-encryption of all CSPs. In the second generation EKMS, an additional CSP-encryption-key (CEK) was inserted into the key chain, which encrypts the CSPs. The KEK encrypts the CEK. Hence, if the KEK changes, the CEK is re-encrypted instead of re-encrypting all CSPs.

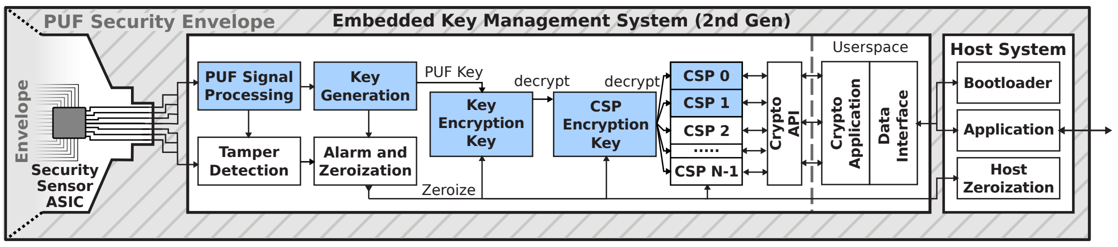


**Figure 3.7:** System architecture of the FORTRESS software, the second generation EKMS. The key chain is depicted in blue color.

Furthermore, during zeroization of the CSPs, it is sufficient to delete only the CEK and hence, prevent any access to the CSPs. All keys and CSPs, apart from the KEK, originate from an RNG. For the generation of the KEK, the measurement task triggers the ASIC and measures the PUF-response, which is executed through the IC Driver (icDrv) task.

The generated KEK is stored in SRAM and repeatedly reproduced at startup. It decrypts the CEK and verifies CSP 0, which is a known plaintext. If the key chain is compromised through an attack, CSP 0 is not correctly decrypted, which triggers the alarm. During runtime, the PUF-key is repeatedly reproduced and compared to a hash of the KEK. If the integrity check of the envelope or the key match fails, the alarm and zeroization procedure are triggered, deleting the CEK, CSPs, and SRAM content. Furthermore, the EKMS notifies the host system.

In the first generation EKMS, an AES ECB implementation was added as a proof-of-concept. The extended EKMS provides additional cryptographic primitives, such as 128- and 256-bit AES in ECB and CBC mode for encryption and decryption, SHA256 for hashing, and ECC secp256k1 for signing.

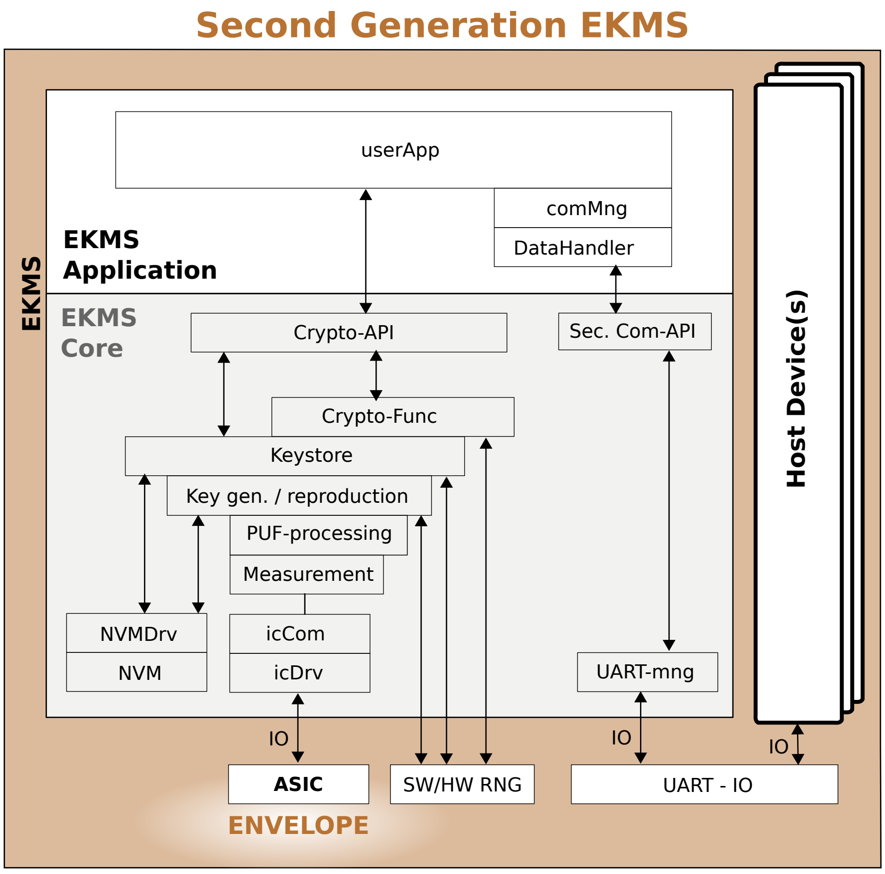


**Figure 3.8:** Layers of the FORTRESS software, the second generation EKMS, with EKMS Core and Application.

Since the EKMS is designed as a modular system, other cryptographic algorithms can be easily integrated. This modularity extends to the measurement and layout designs since the EKMS supports all measurement systems (discrete and ASIC) and enclosure designs (FORTRESS, B-TREPID, COVER).

After assembly and functional tests of the FORTRESS HSM, the EKMS was run non-stop over several months in order to test its stability and availability. These long-term tests were conducted successfully without failures. Availability is a major factor for productive operation since HSMs remain in the field for many years or even decades.

### 3.2.3 Secure System Life Cycle

Enabled by the re-enrollment capabilities, the second generation EKMS was extended by a secure system life cycle, which is depicted in Figure 3.9. The life cycle incorporates important parts of the supply chain, such as transportation, operation in the field, and decommissioning. For the secure life cycle, I assume the trustworthiness of the manufacturer.

#### Unprovisioned

After the fabrication, the HSM is unprovisioned. The manufacturer starts a system check testing the envelope's basic functionality and electrical integrity. If the test is successful, the manufacturer triggers the enrollment process in the keystore and sets the device into shipping mode. The transition between stages, i.e., modes, is implemented through the keystore.

#### Shipping Mode

The shipping mode is crucial in the detection of manipulations that occur during transportation. After delivery of the device, the user tests the device for possible manipulations. This test requires two steps. In the first step, the device verifies the encryption procedure, the PUF, and the key chain, by successfully decrypting CSP 0 (*Verify Plain*), a known plaintext. Hence, this first step is a form of *self-verification.* In the second step, the user verifies that the correct device was delivered. This step is referred to as *user-verification.* CSP 1 (*Ident*) — an elliptic curve key pair — was created in the unprovisioned stage upon the initial enrollment. The public key is either provided by the manufacturer through a second channel or read from the delivered device. To verify the correct identity of the delivered device, the user signs data with the private and then verifies its signature through the provided public key. To be more precise, the user sends data to the device via its UART interface and accesses the EKMS Application Programming Interface (API) to sign the data with the CSP-1 private key. The user then verifies the signature via the CSP-1 public key to attest to the correct identity of the shipped device.

If both tests are successful, the user triggers a second enrollment, which sets the device into field mode. This mode transition deletes the CEK and CSPs and creates a new CEK, CSP 0, and CSP 1. The enrollment also triggers the generation of a new KEK corresponding to the PUF-key. The definition of the PUF-secret, and hence, the KEK, varies depending on the chosen key generation scheme. I chose the Fuzzy Commitment scheme [JW99] for the implementation described in Section 3.3. In this case, the PUF-secret corresponds to a random number generated by a TRNG, which is masked by the

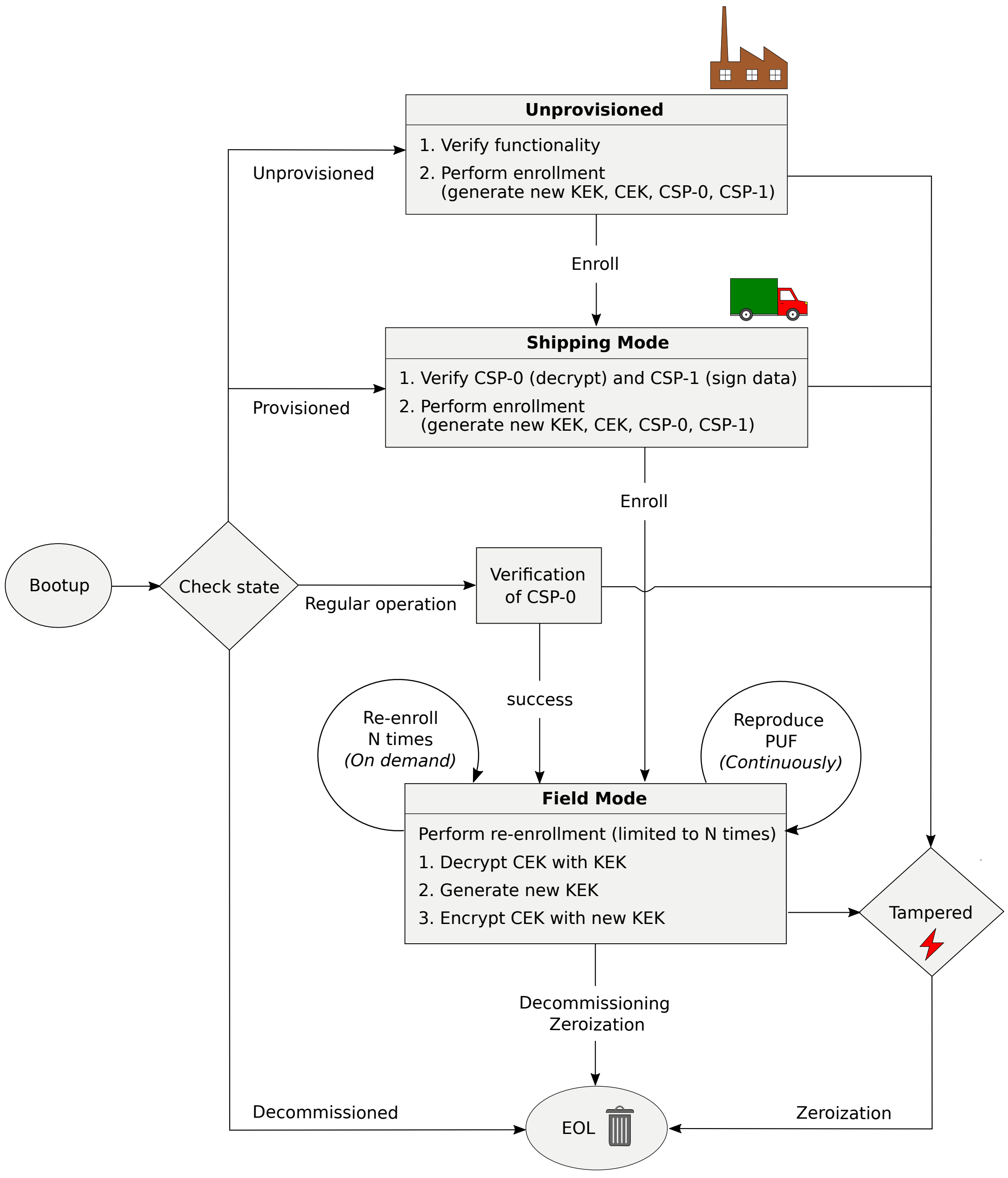

Unprovisioned
1. Verify functionality
2. Perform enrollment (generate new KEK, CEK, CSP-0, CSP-1)
Unprovisioned
Enroll
Shipping Mode
1. Verify CSP-0 (decrypt) and CSP-1 (sign data)
2. Perform enrollment (generate new KEK, CEK, CSP-0, CSP-1)
Provisioned
Enroll
Bootup
Check state
Regular operation
Verification of CSP-0
success
Re-enroll N times (On demand)
Reproduce PUF (Continuously)
Field Mode
Perform re-enrollment (limited to N times)
1. Decrypt CEK with KEK
2. Generate new KEK
3. Encrypt CEK with new KEK
Tampered
Decommissioning Zeroization
Decommissioned
EOL
Zeroization

**Figure 3.9:** The secure system life cycle of FORTRESS, covering transportation, regular operation, and decommissioning.

PUF-response. Thus, the enrollment triggers the generation of a sequence of random numbers corresponding to a new secret that leads to a new KEK.

### Field Mode

After verifying the successful delivery of the correct device, the HSM is set into *field mode*. This mode is the common operational mode, where no further enrollment is possible. However, since the aging of the envelope over time leads to slight changes in the PUF-response, a limited amount of re-enrollments is allowed to generate a new KEK. In contrast to reproduction, re-enrollment is not performed continuously but is triggered on demand. Limiting the number of re-enrollments is necessary to counteract helper data manipulation attacks [MSSS11b, DV14a, MSS13, TPS17].

Through helper data manipulation attacks, the attacker can gain knowledge about the PUF-secret by observing the reproduction of the PUF-key while manipulating the helper data. In 2011, Merli et al. conducted an EM side-channel attack on a fuzzy extractor targeting the Toeplitz hash function [MSSS11b]. They showed that the PUF-key could be derived from 2,000 traces. However, the additional randomization of helper data did not significantly enhance the quality of the captured traces. An attack not based on side-channel analysis was proposed by Delvaux and Verbauwhede, who targeted Pattern Matching Key Generators (PMKG) [DV14a]. They demonstrated that through statistical observation of the failure rate and helper data manipulation, the PUF-secret could be retrieved with approximately 10,000 samples. Furthermore, Merli et al. [MSS13] and Tebelmann et al. [TPS17] conducted power and EM side-channel attacks for a concatenated code, where a repetition decoder is followed by a BCH decoder. Both attacks, in total, require $n_{\mathrm{BCH}} \times 2^{n_{\mathrm{rep}}}$ helper data manipulations, where $n_{\mathrm{rep}}$ and $n_{\mathrm{BCH}}$ denote the code length of the repetition code, or BCH code, respectively. For $n_{\mathrm{rep}} = 7$ and $n_{\mathrm{BCH}} = 127$, the number of necessary manipulations amounts to 16256. To counteract helper data manipulation attacks, the number of re-enrollments, and thus, the generation of new helper data, has to be limited. In the following, I restrict the re-enrollments in field mode to a maximum number of 1.

A further attack scenario is the step-wise partial removal of the envelope followed by re-enrollment. Therefore, limiting the number of re-enrollments to a small single-digit number is essential in counteracting these attacks.

The re-enrollment procedure is again processed by the keystore. The keystore decrypts the CEK with the current KEK, generates a new KEK from the PUF-response and re-encrypts the CEK. This process leaves the CEK and CSPs unchanged.

Quality indicators, such as *confidence* of the uncorrected PUF-response or the number of corrected errors, are generated during the key reproduction. The confidence of the PUF denotes how much the quantized (uncorrected) PUF-response deviates from the interval center after applying all post-processing steps, as described in Section 2.4. These health data provide additional information on the state of the PUF. An overall shift of the PUF-response indicates changes related to aging. However, fast and large changes within the PUF-response likely originate from an attack. These changes are considered in the error correction, as discussed in Chapter 5.

#### Decommissioned / End-of-Life

When the device has reached its end-of-life (EOL), the user triggers the transition into *decommissioned* mode. In this mode, no further enrollments or re-enrollments are possible. What is more, switching into this mode triggers the zeroization procedure, where all critical data are deleted. The transition into decommissioned mode also occurs when a tamper event is detected. This irreversible mode transition can be triggered at any point in the life cycle.

## 3.3 PUF-Key Generation

The first generation EKMS incorporated only a rudimentary key generation scheme based on a binary decision, mapping PUF-values $\leq 0$ to a binary 1 and values $< 0$ to 0. This "quantization" into two intervals results in a considerable entropy loss. Furthermore, the handling of edge cases is not covered. This means that environmental changes can easily shift the quantized differential capacitances, also referred to as differential nodes, residing at the interval edges of the neighboring interval. This primitive key generation scheme served as a first proof-of-concept without considering post-processing, error correction, or any security objectives.

To overcome the drawbacks of this method, I extended the EKMS with a full key generation and reproduction scheme tailored to the enclosure's security requirements. This includes all steps, from the envelope measurement to the post-processing (see Section 2.4) and the error correction.

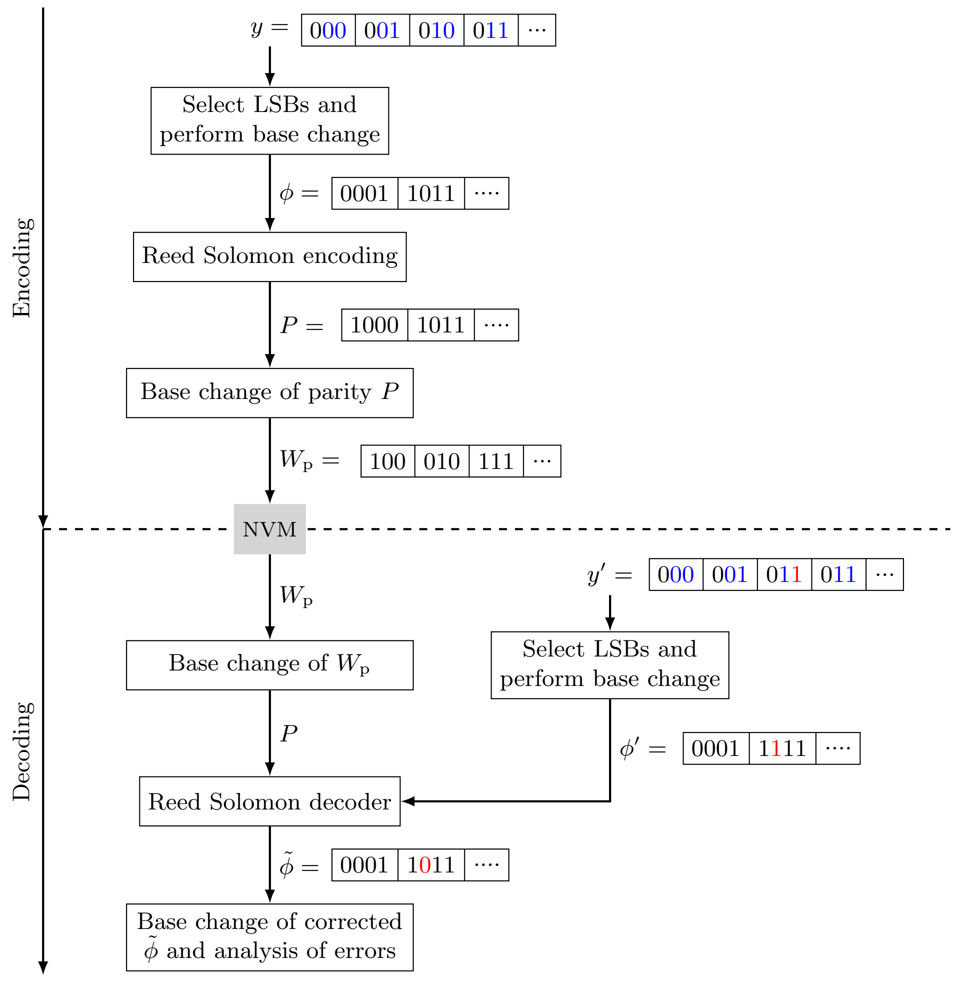
The key generation is based on the Fuzzy Commitment scheme [JW99], where the PUF-response masks the secret that originates from an RNG. Compared to other schemes like the Fuzzy Extractor [DRS04], the Fuzzy Commitment does not define the PUF-response as the secret. This enables the implementation of a secure system life cycle since, during each triggered enrollment, a new random secret is generated, and thus, the KEK changes significantly. Hence, the secret after provisioning differs from the secret during regular operation of the HSM.

As noise and environmental effects lead to changes in the measured PUF-response, the key generation scheme comprises an error correction algorithm. However, this error correction code typically does not distinguish between changes stemming from environmental effects and deliberate changes in the PUF-response originating from an attack [ION+19, IOK+18, IHL+19].

To ensure reliable reproduction of the PUF-response under normal conditions while discarding deliberate degradation of the PUF, I extended the key generation scheme with Limited Magnitude Codes (LMCs) [JL12, IU19] that provide the required tamper-sensitivity.

Previous measurements of the differential capacitances showed that drilling attacks cause large offsets in the PUF-response, while temperature changes lead to smaller offsets [ION+19, IOK+18]. LMCs limit the error correction to a specific number of interval shifts that may occur through external influences. This is implemented by correcting only the LSBs of each PUF-symbol, and hence, leaving the Most Significant Bits (MSBs) unchanged, except for carries that can occur during the calculation.

Figure 3.10 shows the encoding and decoding steps for Limited Magnitude Codes for exemplary LMC parameters $q = 8$, $q' = 4$, and $p = 16$. The message $y$, which corresponds to the PUF-secret, comprises 128 symbols of a $q-$ary alphabet, where $q$ also corresponds to the number of quantization intervals. The first step of the encoding "reduces" $y$ to its LSBs. The number of selected LSBs per symbol depends on the LMC parameter $q'$. The selected bits $y \mod (q')$ are then grouped into $p-$ary symbols $\phi$, depending the field size $p$ of the chosen Reed Solomon code. This grouping from $q-$ary symbols to $p-$ary symbols requires a base change. For Limited Magnitude Codes, the parameters are chosen such that $q' \leq q \leq p$. Only certain parameter combinations may be chosen to match the number of bits in both representations ($q$ and $p$), . The encoder then generates the parity $P$ and codeword $C = \phi \parallel P$ for the corresponding input $\phi$.



Encoding
Decoding
$y =$ 000 001 010 011 ···
Select LSBs and perform base change
$\phi =$ 0001 1011 ····
Reed Solomon encoding
$P =$ 1000 1011 ····
Base change of parity $P$
$W_p =$ 100 010 111 ···
NVM
$W_p$
$y' =$ 000 001 011 011 ···
Select LSBs and perform base change
Base change of $W_p$
$P$
$\phi' =$ 0001 1111 ····
Reed Solomon decoder
$\tilde{\phi} =$ 0001 1011 ····
Base change of corrected $\tilde{\phi}$ and analysis of errors

**Figure 3.10:** Overview of Limited Magnitude encoding and decoding.

After encoding, the helper data $W_\mathrm{p}$, obtained from the base change of the parity $P$, are stored in an NVM of the device. This terminates the encoding procedure.

For decoding the erroneous message $y' = y + \epsilon$, where $\epsilon$ is the corresponding error vector, the helper data $W_\mathrm{p}$ are subject to a base change. The resulting parity symbols $P$ together with $\phi$, the reduced message after base change, yield the erroneous codeword. The decoding yields the corrected $\tilde{\phi}$. The last step of the decoding comprises a base change of $\tilde{\phi}$ and an analysis of the resulting error vector, only keeping errors according to the specified LMC parameters. This means that large changes over several neighboring intervals are discarded from the error vector.

Measurements of the capacitive enclosure demonstrate that, due to global manufacturing variations, some RX and TX groups display larger offsets than others. Hence, interleaving of PUF-symbols could be applied to reduce resulting burst errors. However, since the reduction of burst errors reduces the tamper-sensitivity of the PUF, I omit interleaving the PUF-symbols in my implementation. In the following, I describe the generation and reproduction of the PUF-key, which is tailored to the capacitive envelope.

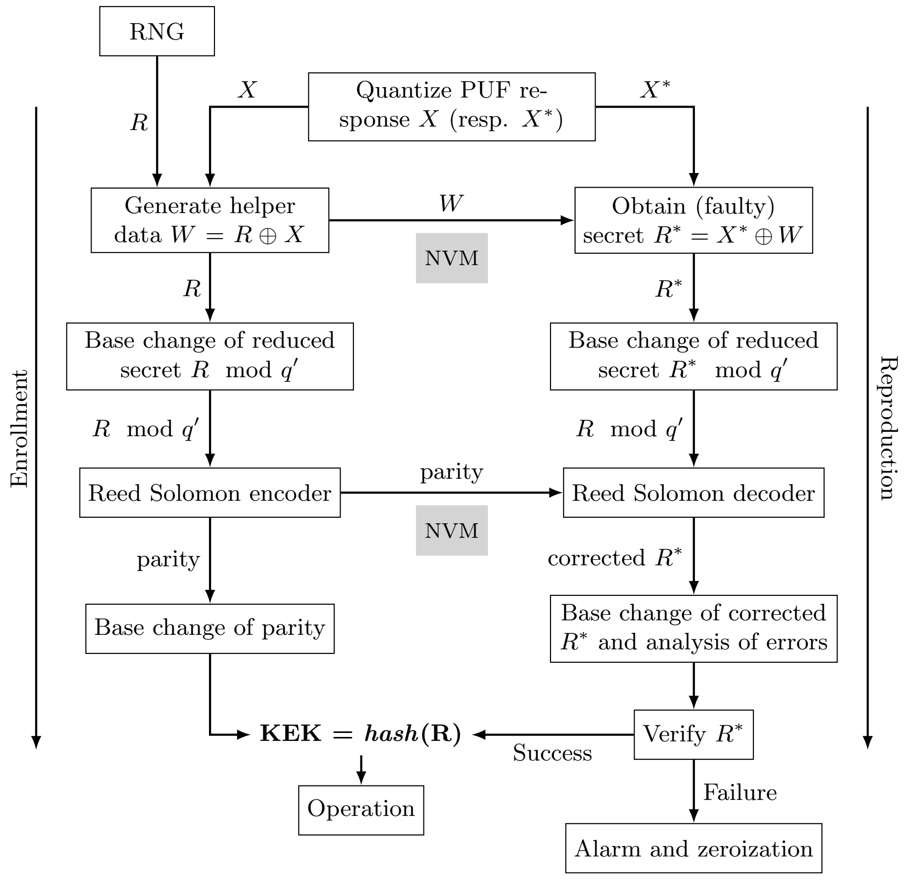
### 3.3.1 Enrollment

Figure 3.11 shows the enrollment and reproduction of the PUF-secret. During the enrollment (or re-enrollment) stage, a new KEK is generated. After measuring the differential capacitance, the PUF-response is normalized and quantized, as described in Section 2.4. In the fuzzy commitment scheme [JW99], the secret $R$ is a random number $R$ obtained from an RNG. In the final post-processing step during enrollment, as described in Subsection 2.4.2, the analog helper data $W = R \oplus X$ are generated from the quantized PUF-response $X$ and the secret $R$. They are stored in NVM. The PUF-response itself is not corrected, but changes in $X$ result in changes in $R$.

LMCs are based on Reed Solomon (RS) codes [RS60]; however, since only the LSBs of the secret are corrected during reproduction, the secret $R$ is *reduced* to $R \mod q'$, where $q' = 2^m$, and $m$ is the number of LSBs to be corrected. This is followed by a base change, where the reduced secret in PUF-representation is mapped to RS code representation. Subsequently, the RS encoder calculates the parity from the reduced secret and stores it in NVM together with the analog helper data.

For the proof-of-concept implementation, I chose $q = 8$ equidistant quantization intervals, i.e., intervals of the same width, with $q' = 4$, a message length of 32, and parity of 24. All parameters are listed in Table 3.1. The 128 symbols of the secret are each reduced to $\log_2 q'$ LSBs. During the base change, they are rearranged such that the reduced secret length corresponds to $\log_2 p$ times the message length 32 for the RS encoder ($128 \times 2 = 32 \times 8$). Similarly, the base change for the parity is performed, matching $\log_2 p$ times 24 parity symbols to 64 times the $\log_2 q$ bits of the PUF interval quantization ($24 \times 8 = 64 \times 3$).

Concatenating the secret and the parity yields the codeword. After encoding, the reverse base change from the RS representation to PUF-representation is performed, and the parity is stored in NVM.



RNG
$R$
$X$
Quantize PUF response $X$ (resp. $X^*$)
$X^*$
Generate helper data $W = R \oplus X$
$W$
NVM
Obtain (faulty) secret $R^* = X^* \oplus W$
$R$
$R^*$
Base change of reduced secret $R \mod q'$
Base change of reduced secret $R^* \mod q'$
$R \mod q'$
$R \mod q'$
Reed Solomon encoder
parity
NVM
Reed Solomon decoder
parity
corrected $R^*$
Base change of parity
Base change of corrected $R^*$ and analysis of errors
**KEK = *hash*(R)**
Success
Verify $R^*$
Failure
Operation
Alarm and zeroization
Enrollment
Reproduction

**Figure 3.11:** Overview of key generation scheme, with enrollment (left) and reproduction (right).

Through the higher order alphabet and quantization of the PUF-response, more errors can be corrected compared to fully binary schemes. In total, 12 correctable symbols in RS representation map to 48 PUF-symbols. One of the drawbacks of this scheme is that the error correction parameters have to be chosen according to the base change. This limits the number of possible configurations. The implementation for the enclosure was based on a Gray code representation for the quantized PUF-response. Furthermore, in the error analysis, only symmetric errors in the PUF-response were considered.

Upon boot up, the KEK is reproduced and verified by correctly decrypting CSP-0, which is a known plaintext. The reproduced KEK, corresponding to the hashed secret, is held in SRAM and, subsequently, compared to the continuously reproduced PUF-key in order to verify the integrity of the enclosure. The advantage of additionally hashing the secret — which is not comprised in the fuzzy commitment — is that even if the PUF key only changes slightly, its hash will change significantly.

Additional enrollments, and re-enrollments, are crucial for the secure system life cycle. However, their number has to be limited, as described in Subsection 3.2.3.

**Table 3.1:** Key generation parameters.

| **Parameter** | **Value** |
|---|---|
| PUF symbols | 128 |
| Field size ($p$) | $2^8$ |
| Message length ($k$) | 32 |
| Parity | 24 |
| Quantization intervals ($q$) | 8 |
| LMC parameter ($q'$) | 4 |
| Base change (secret) | $128 \times 2 = 32 \times 8$ |
| Base change (parity) | $24 \times 8 = 64 \times 3$ |

### 3.3.2 Reproduction

The enrollment is only performed in the unprovisioned stage and after shipment, whereas the key reproduction is performed repeatedly after boot up. The steps of the reproduction are depicted in Figure 3.11. First, the measured PUF-response that might have changed due to an attack or environmental influences is processed and quantized. The secret $R^* = X^* \oplus W$ is derived from the stored analog helper data $W$ and the quantized PUF-response $X^*$. After the base change, the codeword, which is a concatenation of the reduced secret $R^*$ and the stored parity, is decoded. For the RS decoder, I implemented the Berlekamp Massey algorithm [Mas69, Ber68], Chien search [Chi64], and the Forney algorithm [For65b]. The output of the decoder — after the base change — is the reduced secret from which the KEK is obtained.

After boot up, the reproduced KEK decrypts the CEK, which decrypts further CSPs. The correctly verified KEK in SRAM is compared against the subsequently reproduced key. If the reproduced KEK differs from the correct key, the alarm and zeroization of all CSPs are triggered. During the decoding, PUF health data are generated. One such example is the confidence of the reproduced PUF-response, i.e., the average distance to the interval center after subtracting the analog helper data. Moreover, the number of corrected errors is stored as an additional parameter. The PUF health data serve as additional quality indicators of the reproduced PUF-key.

The key generation and reproduction were successfully tested and executed for the implementation test on the EKMS with the data sets obtained from the envelope [IOK$^+$18]. This was followed by a successful key derivation from the FORTRESS envelope. I also tested the implementation with previously published attack data [IOK$^+$18, ION$^+$19], from which the correct key could not be reproduced. The tests demonstrate that the proposed scheme is practically feasible, even on a resource-limited microcontroller.

## 3.4 Conclusion

In this chapter, I presented the assembly of all FORTRESS components into an exemplary PUF-based HSM. The FORTRESS software was extended by the second generation

EKMS providing a hardened operating system, a secure life cycle, and a full key generation tailored to the enclosure. The secure life cycle incorporates the main aspects of the supply chain, supported by the EKMS key management. The key generation was extended through Limited Magnitude Codes, which are a simple method to model environmental influences considering errors based only on their magnitude. In Chapter 5, I provide a detailed analysis of external influences and discuss an extension of the PUF post-processing to model more complex error behavior.

In the future, the FORTRESS prototype could be further developed by reducing the envelope's layer stack size through different materials and hence, increasing its bendability. In terms of the measurement system, the ASIC's optimization towards size results in reduced accuracy. Thus, future work should also focus on improving the ASIC's resolution of the PUF-response.

# Chapter 4

# Physical Attacks and Countermeasures

In this chapter, I analyze three of the most relevant attacks on the enclosure system. Previously, Immler et al. investigated drilling attacks with a diameter of 250-300 µm [IOK$^{+}$18, ION$^{+}$19]. I extend this analysis by considering micro-drilling of the enclosure, bypassing of electrodes, and magnetic probing of critical bus systems. For each of these attacks, I propose countermeasures to restore the envelope's tamper-sensitivity.

## 4.1 Attack Resiliency of the Capacitive Enclosure

The capacitive enclosure was subjected to several threats, to test its attack resiliency. Immler et al. showed that the X-ray inspection revealed the structure and location of TX and RX electrodes [ION$^{+}$19]. This is unavoidable since thick shieldings against high-energy radiation can not be applied in all use cases. Hence, information leakage about the envelope layout is considered an acceptable risk.

Invasive attacks such as complete or partial disassembly can be hampered by potting and adhesives. Hence, in case of complete removal, wrapping and bending would further damage the envelope. The partial removal of the shielding serves as a preparation for probing attacks that aim at extracting data about the PUF. In 2019, Obermaier extracted parts of the PUF-response by bypassing and reconnecting electrodes of the capacitive mesh [Obe19]. To restore the tamper-sensitivity of the enclosure, I discuss the impact of this attack and propose countermeasures in Section 4.3. In Section 4.4, I demonstrate a magnetic probing attack targeting the communication between the ASIC and microcontroller. I also discuss the effects of electric and magnetic fields through Finite Element Analysis (FEA).

Apart from complete or partial removal, the attacker can also drill holes through the enclosure to probe signals of the PCB. Immler et al. showed that drilling attacks with a 300 µm diameter lead to significant entropy loss [IOK$^{+}$18]. So far, drilling attacks with a diameter below 300 µm have not been attempted. Since the electrode layout was designed to withstand drilling attempts $\geq 300\,\mu m$, permeating the enclosure with energy beams, such as lasers or Focused Ion Beams (FIBs), is theoretically possible. To address this attack vector, I discuss micro-drilling attacks with diameters $\ll 300\,\mu m$ in Section 4.2. Figure 4.1 depicts the enclosure system targeted by the three physical attacks discussed in this chapter.

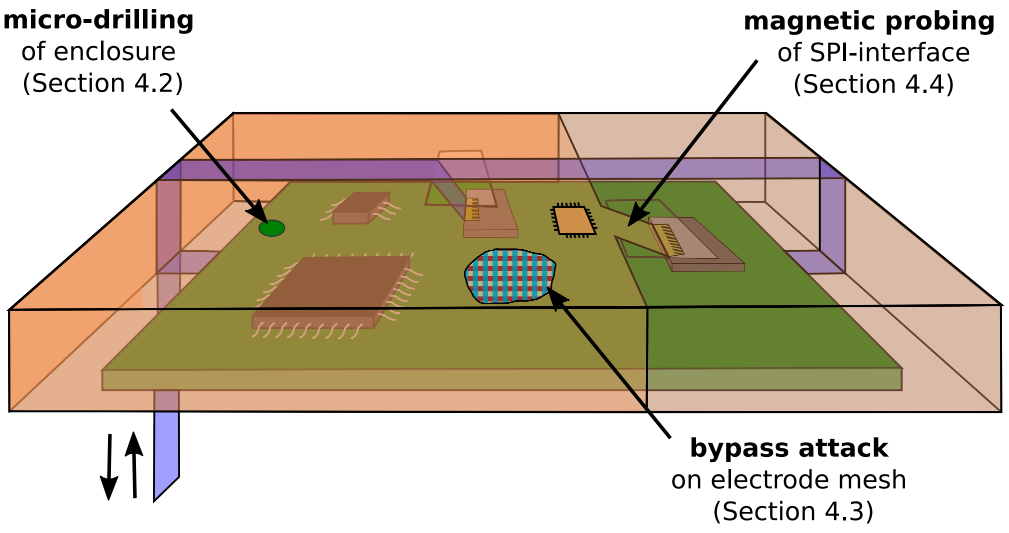


**Figure 4.1:** Overview of the envelope with embedded ASIC wrapped around the casing containing the PCB (see Figure 3.1). Three physical attacks are discussed in this chapter: The micro-drilling attack (Section 4.2), the bypass attack (Section 4.3), and magnetic probing of the SPI-interface (Section 4.4).

I presented the results discussed in this Chapter at IEEE Physical Assurance and Inspection of Electronics (PAINE) 2021. Further details are published in the IEEE conference proceedings [GSHO21].

## 4.2 Micro-Drilling

In this section, I discuss micro-drilling attacks on the capacitive enclosure. The goal of these attacks is to probe wires or pads of the PCB in order to eavesdrop and manipulate critical signals.

### 4.2.1 Micro-Drilling Techniques

Immler et al. investigated drilling attacks on the envelope with a minimum diameter of 250-300 µm [IOK$^+$18, ION$^+$19]. However, attacks with smaller diameters have become more and more feasible in recent years. Since the electrode width and distance between traces of the same layer amounts to 100 µm, drilling attacks with 300 µm destroy two neighboring electrodes, which is reliably detected through the capacitive measurement. In current retailing, drills with a diameter of 250 µm are available that are still reliably detected. Since holes $\ll$ 300 µm are feasible through energy probes such as lasers or FIBs, their impact on the capacitive enclosure needs to be investigated.

Table 4.1 shows an overview of different drilling techniques, including attack diameters and aspect ratios across different sources [HZJ17]. The aspect ratio is the ratio between the depth of a hole and its diameter.

**Table 4.1:** Comparison of micro-drilling techniques covering aspect ratios and hole sizes [HZJ17]).

| **Technology** | **Common aspect ratio** | **Min. aspect ratio** | **Common hole size in µm** | **Min. hole size in µm** |
|---|---|---|---|---|
| Micro drilling | 2 – 10 : 1 | 8 – 24 : 1 | > 10 | 2.5 |
| Laser | 10 : 1 | 600 : 1 | 50 – 400 | 1 |
| Electro Discharge | 10 : 1 | 30 : 1 | > 100 | 5 |
| Electro Chemical | 8 : 1 | 250 : 1 | > 50 | 8 |
| Electron Beam | 10 : 1 | 25 : 1 | 8 – 200 | 50 |
| Ultrasonic | 10 : 1 | 31 : 1 | 100 – 500 | 5 |

For sophisticated attackers, drilling small *and* deep holes with very high aspect ratios of more than 100 : 1 is feasible. In general, even higher aspect ratios of more than 250 : 1 are realistic with laser drilling or electro-chemical micro-drilling, which lead to holes $< 100\,\mu m$ that are small enough not to perforate any electrodes. The impact of different aspect ratios for micro-drilling is depicted in Figure 4.2.

In general, some technologies are more suitable for drilling attacks on the capacitive enclosure. FIB attacks, for instance, are well-suited for attacking integrated circuits or surface structures [HNT+13] but are less suited for drilling deep holes. This is due to debris created from incoming ions removing atoms at the bottom of the drilled hole. If the aspect ratio is large, the atoms agglomerate in the drilled hole without any possibility of removal. Hence, deep holes with diameters of approximately 10 µm are generally not feasible with a FIB.

When adding a metal casing around the PCB, the attacker is forced to drill wider holes at the surface to reach the necessary depth required for probing the PCB. With a layer stack of approximately 250 µm and an internal metal casing of 1 mm enclosing the host system, micro-drilling with a typical aspect ratio of 10 : 1 requires an attack diameter of more than 125 µm to drill a hole of 10 µm. If the casing thickness is increased to 2.25 mm, the attacker is forced to increase the diameter to 300 µm, leading to the destruction of

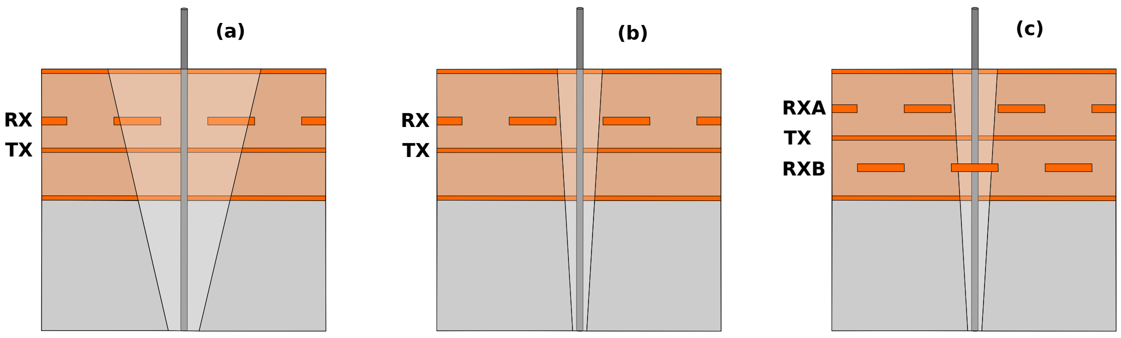


**Figure 4.2:** Schematic image for deep micro drilling with (a) low aspect ratios, (b) high aspect ratios, and (c) countermeasure against tools with high aspect ratios.

two electrodes. However, the area above the PCB connectors for external communication is not protected from a metal casing.

### 4.2.2 Micro-Probing

After successfully drilling a deep hole piercing the envelope, the PCB is accessible for micro-probing. In this section, I focus on deep insertion, as depicted in Figure 4.3. I discuss surface probing in Section 4.3.

Probing tools are available as commercial wafer probing needles in different sizes. Micromanipulator needles to contact fine electrodes have a tip point radius $\geq 0.6\,\mu m$. [Ame, Mic]. The probing needle tip wire has a length of 3.3-5 mm and a 76 µm diameter. It is attached to a 30-35 mm long shank with a diameter of 250-500 µm.

The micromanipulator's 250-500 µm diameter shank must be moved through the drilled hole to probe internal wires and pads of the PCB more than 3 mm from the enclosure. Holes of this size are easily and reliably detected during the capacitive measurement.

Even finer tungsten wires with a tip length of 3.3-5 mm and a diameter of 10-20 µm [GGB] are commercially available for probing. With a tip shaft length of 5.1 mm and 22 µm diameter, reaching the PCB is feasible, assuming a casing thickness $< 2\,mm$.

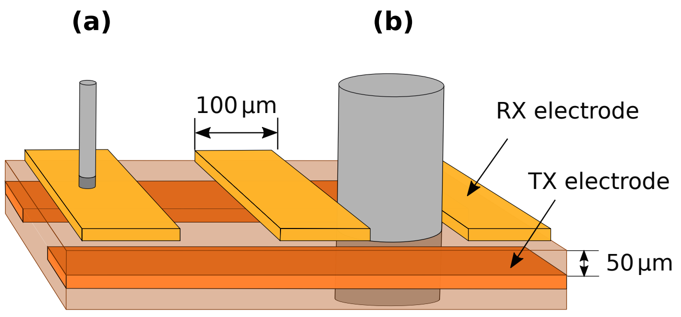


**Figure 4.3:** Schematic image for (a) surface probing (as discussed in Section 4.3) and (b) deep insertion.

### 4.2.3 Countermeasures

To hamper drilling attempts with high aspect ratios and PCB probing, the thickness of the case is a crucial parameter. The mechanical stability of the probing needles requires a shaft of greater diameter than the tip. Hence, even if an attacker successfully perforates the envelope with a small hole, the length of the tip might not suffice to reach the PCB. To also protect the connectors for external communication, the casing holes above the connectors can be significantly reduced in size, thereby enforcing larger surface drill diameters.

Additionally, a temperature-resistant potting material, e.g., epoxy resin or silicone, could seal the enclosure. Obermaier and Immler analyzed different HSM implementations and showed that a strong adhesion between the potting and the external envelope layer is indeed possible [OI18]. Hence, the force of mechanical removal of the potting tears the envelope layers apart and destroys the PUF before the potting is detached from the envelope. The potting additionally increases the surface drill diameters, thereby impeding micro-drilling attacks with small diameters.

Apart from these countermeasures, the enclosure itself could be modified. An additional RX electrode layer fills the 100 µm × 100 µm gaps of the electrode mesh. This layer is depicted as RXB in Figure 4.2. Furthermore, the envelope's feature size could be further reduced from 100 µm to smaller sizes that are still feasible within the lithographic process. This reduces the necessary surface drill diameter and raises the bar for an attacker.

## 4.3 Bypass Attack

In this section, I discuss a surface probing attack previously performed by Obermaier [Obe19] and propose a countermeasure impeding its successful execution. This is preceded by a summary of the measurement principle that yields the differential capacitances.

### 4.3.1 Measurement Principle

The PUF-response is comprised of 128 differential capacitances in the two-digit femtofarad range. The goal of the measurement, hence, is to obtain these minuscule capacitive variations, i.e., the differential capacitance $\Delta C$ between two TX and one RX electrodes. The key idea of the measurement principle is an "in-enclosure" current subtraction proposed by Obermaier [Obe19]. Obermaier found that subtracting the measured absolute capacitances to obtain the differential capacitance $\Delta C$ does not yield sufficient accuracy and even described it as "infeasible" since the small differential capacitances submerged in the system's gain-mismatches, biases, and offsets. To solve this problem, Obermaier developed a way to subtract the capacitances "within the envelope" instead of subtracting the absolute capacitances after the measurement.

The balanced enclosure design ensures that two neighboring TX electrodes have the same mutual capacitance $C_m$ towards an RX electrode. The two neighboring TX electrodes are excited via a sinusoidal voltage signal of amplitude $V_{\text{TX}}$. A phase shift of 180° is applied to the signal of the even-numbered electrode, generating two currents $j$ of inverse signs — from two TX electrodes to one RX electrode — that are proportional to the mutual capacitance $C_m$. The 180° phase shift leads to a mutual cancellation of currents resulting in a complex electrode current

$$\underline{I}_{\text{RX}} = j \cdot V_{\text{TX}} \cdot C_m + j \cdot (-V_{\text{TX}}) \cdot (C_m + \Delta C) = -j \cdot V_{\text{TX}} \cdot \Delta C \tag{4.1}$$

directly proportional to the differential capacitance $\Delta C$. This measurement is repeated for all 8 TX pairs and all 16 RX electrodes.

In contrast, the ASIC currently employs an "out-enclosure" measurement concept leading to a reduced accuracy of the differential capacitances, where two RX electrodes are measured against one TX electrode [FIU+18]. The subtraction of capacitances, in this case, occurs after the absolute capacitive measurement. Since the "in-enclosure" measurement of the discrete measurement circuit provides more accurate results, a future development goal for the ASIC is to deploy the above-described concept. In contrast to the discrete measurement circuit, the ASIC can measure multiple electrodes in parallel. This, together with the implementation of the improved measurement principle, would increase the ASIC's accuracy while reducing the measurement time. In the following, I focus on the "in-enclosure" principle, which is currently only implemented for the discrete measurement circuit.

### 4.3.2 Surface Probing Attack

In 2019, Obermaier proposed a probing attack severing and reconnecting one of the RX traces. He intercepted the RX current signal, cloning and redirecting it into the envelope. This modification was not detected in the measured PUF-response [Obe19].

The differential capacitances corresponding to the PUF-response stem from sub-nanoampere currents on the RX traces. During the measurement, the voltage on the electrode is constant. Hence, the information about the PUF is contained within the sub-nanoampere currents on the RX electrodes. Through access to the cloned RX current signal, an attacker can, hence, derive information about the PUF-response. A schematic of the attack is depicted in Figure 4.4.

Obermaier prepared his attack by removing an area of 5 mm×5 mm from the envelope's shielding close to the envelope's connector. The vicinity to the connector is necessary since only here the entire current, and thereby capacitance, of this trace is accessible.

Removing the shielding enabled him to attach fine shielded wires to one of the RX traces to extract the RX current signal. Obermaier developed an RX-probe specifically tailored to the capacitive envelope that duplicates, amplifies, and converts the RX cur-

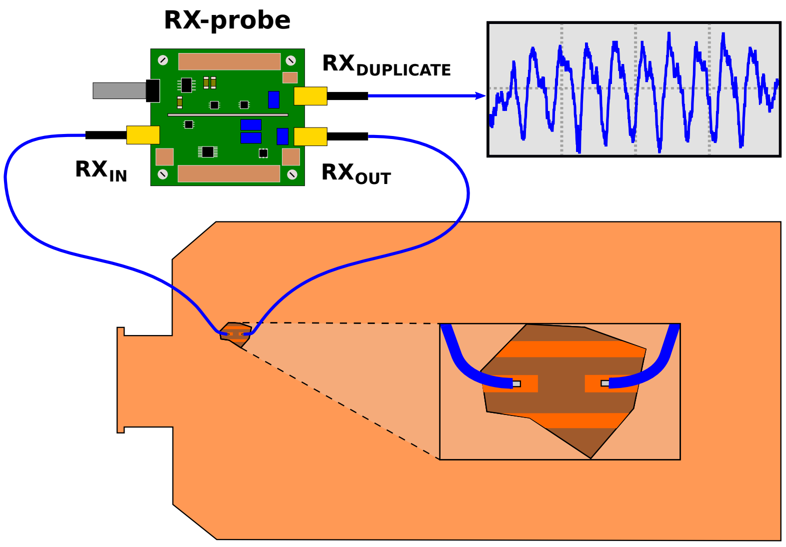


**Figure 4.4:** Schematic of the bypass attack.

rent signal and sends it back into the enclosure. The duplicated RX signal matched the original signal, which led to the identical PUF-response. Neither the integrity check nor the capacitive measurement detected the RX probe.

Since the attack can be performed for multiple RX electrodes, this severely impacts the security provided by the enclosure. Hence, the attack requires countermeasures, which I discuss in the following.

### 4.3.3 Countermeasure

The current probing attack affected the noise components of the original signal, shifting the (unwanted) second harmonic in phase. However, the slight distortion in the RX current did not alter the original PUF-response [Obe19]. Obermaier found that the small phase changes could not reliably be associated with a specific cause. Furthermore, reliably extracting the phase change is complex, and circuit modifications could easily be circumvented. Hence, these minuscule phase changes are unsuitable for counteracting surface probing attempts.

The countermeasure I propose can be integrated either into the measurement circuit or the enclosure itself. The key idea of the differential capacitance measurement [OIHS18] is an antiphasic, i.e., 180°, excitation signal carried by the two TX electrodes measured against one RX electrode. Since the equal parts of the absolute capacitance cancel each other out, the resulting RX trace current is directly proportional to the differential capacitance [OIHS18]. The differential capacitances corresponding to the PUF-response are in the two-digit femtofarad range. Hence, the RX currents are equally small, i.e., in the sub-nanoampere range. The proposed countermeasure, depicted in Figure 4.5, is based on this measurement concept. The basic idea is to hide the RX current within a larger offset current by adding capacitors in the two-digit picofarad range at both ends of each RX electrode, as depicted in Figure 4.5 for the $RX_1$ electrode. Due to the antiphasic excitation signals during the capacitive measurement, the values of both added capacitors are automatically subtracted at the end of the RX electrode.

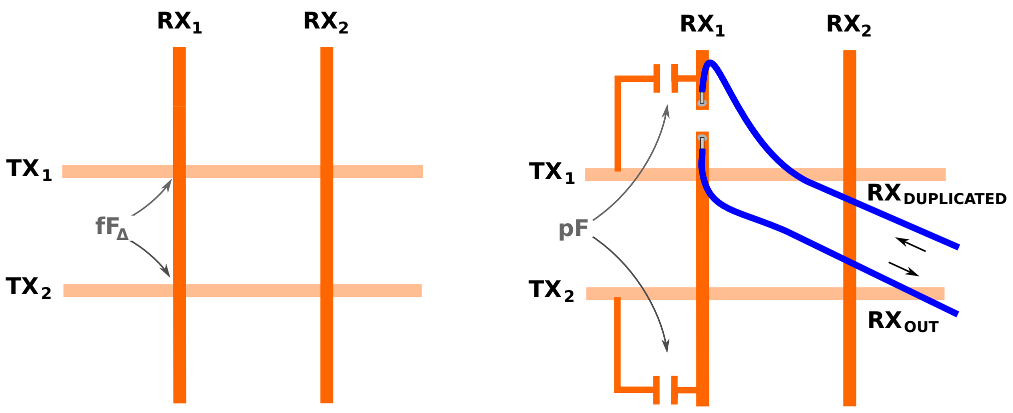


**Figure 4.5:** Regular arrangement of RX and TX traces (left) and countermeasure against the bypass attack (right).

As both electrodes' ends reside within the enclosure boundary and are not accessible from the outside, the attacker can only target the RX current between both electrode ends. Hence, the attacker will only extract the current from one of the additional capacitors since the subtraction occurs at the end of the electrode. Furthermore, the current of the additional capacitor is approximately a thousand times higher than that of the PUF-response. Hence, this countermeasure hides the small differential capacitance in the femtofarad range in a much larger capacitance in the picofarad range. This makes the extraction of the PUF-response infeasible, even if the picofarad capacitance is known.

In total, this countermeasure requires $2 \cdot 16 = 32$ capacitors that are connected to the antiphasic signal generator. These capacitors should be inaccessible and hence, can be embedded into the measurement system or attached to the innermost layer of the envelope as discrete Surface Mounted Device (SMD) components.

Matching both additional capacitors is crucial since these non-tamper-sensitive capacitances could otherwise influence the final PUF-response. For example, embedded capacitors could be matched by symmetric design, while discrete capacitors could be selected through preceding measurement. The calibration of both capacitors requires an extension of the production process with an integrated selection process to avoid an exact match of both capacitances.

## 4.4 Magnetic Probing Attack

The bypass attack presented in the previous section is based on probing the minuscule currents transported on the RX trace. Voltage probing of the electrode does not yield additional information since the information of the PUF-data is contained in the nanoampere currents. The measurement concept itself, hence, prevents any voltage sensing.

There is, however, another side channel. As I will show in the following, the communication between the ASIC and microcontroller is susceptible to magnetic probing. Furthermore, I analyze the effects of electric and magnetic fields on the envelope through Finite Element Analysis.

### 4.4.1 Simulation of Magnetic and Electric Fields

I conducted a Finite Element Analysis (FEA) on a section of the envelope with 4 RX and 4 TX traces, creating 16 overlaps in order to analyze the effects of electric and magnetic fields on the enclosure. FEA is a method to numerically solve differential equations by discretizing space and dividing it into smaller parts, the so-called finite elements.

Figure 4.6 shows the simulation result for the envelope segment depicting the electric field originating from 4 Cu electrodes. For the simulation, I assumed that the envelope was surrounded by air. The results show that the electric field lines terminate perpendicular to the shielding. Hence, the electric field is not-detectable outside of the enclosure.

Figure 4.7 depicts the simulation of the magnetic field. The results show that the 0.5 µm copper shielding does not inhibit the magnetic field. Since the magnetic field lines

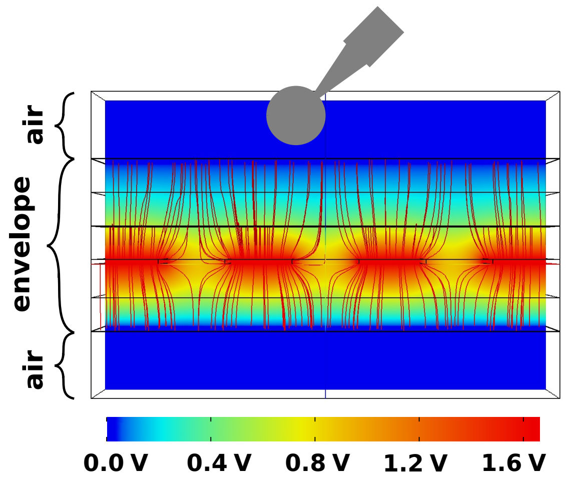


**Figure 4.6:** Electric field of the envelope during capacitive measurement.

expand into the air outside the envelope, they are likely to be detected by a suitable H-probe. The FEA simulation also indicates that the magnetic field emitted from the Serial Peripheral Interface (SPI) communication between the ASIC and the microcontroller leaks information about the PUF-response. In the following, I investigate this further by probing the magnetic field in the vicinity of the envelope.

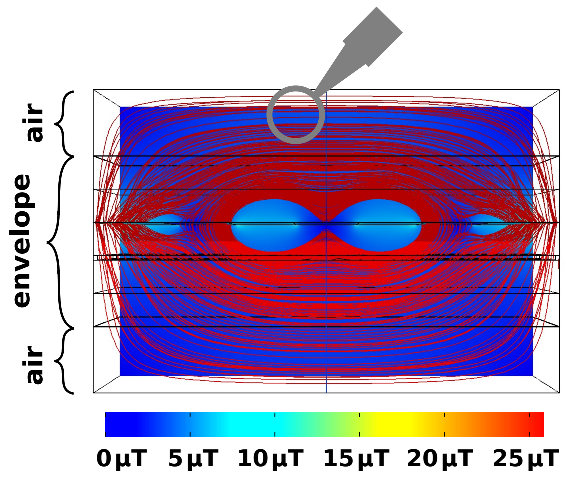


**Figure 4.7:** Magnetic field of the envelope during capacitive measurement.

### 4.4.2 Magnetic Probing

The ASIC communicates with the microcontroller via an SPI interface. To test the susceptibility of this interface to magnetic probing, I targeted a write operation to an ASIC configuration register. To simplify the experiment, I accessed the trigger signal via an external General Purpose Input / Output (GPIO).

The ASIC repeatedly measures the PUF-response every few milliseconds and stores it in SRAM. An attacker that has physical access could, hence, record many traces for later analysis. As I will show, the magnetic signal emitted from the SPI interface is visible to the naked eye. This enables an attacker to adjust the oscilloscope trigger accordingly.

To prepare the setup for the attack, I attached the envelope and the PCB to the table with adhesive tape and tested the signal strength of different magnetic field probes by altering their position. In general, larger coils could detect the target signal more reliably than smaller coils. I identified the Langer RF-R-50-1 as the best-suited H-field probe with the largest signal strength.

To enforce realistic conditions, I added a second envelope on top of the target enclosure. This corresponds to wrapping the envelope around the casing, which results in two overlapping envelope layers covering the ASIC. The additional layer resulted in a significantly decreased signal strength, which can be enhanced by removing a small portion of the shielding. Since a small missing portion of the shielding is not detected in the capacitive measurement (see Section 4.3), I removed an area of approximately 5 mm × 5 mm of the envelope's shielding, significantly increasing the signal strength. For the removal, I protected the shielding outside of the 5 mm × 5 mm area with adhesive tape. A drop of citric acid was placed on the area to be removed, together with a voltage of 3 V applied between the acid and the shielding. This dissolved the copper and revealed the polyimide layer. Figure 4.8 shows the upper envelope with the removed shielding. The galvanic removal resulted in a significantly enhanced signal strength.

The next step of the experiment is to determine if a single bit is distinguishable when transmitted over the SPI interface. I tested this by writing the exemplary values `0x0209` and `0x0249`, differing only in a single bit, to the configuration register of the ASIC. Figure 4.9 depicts the payload (16-bit) of a single register write operation. The amplitude (normalized voltage) of the signal is increased for `1` bits, and can be well-distinguished from `0` bits with the naked eye without further statistical analysis.

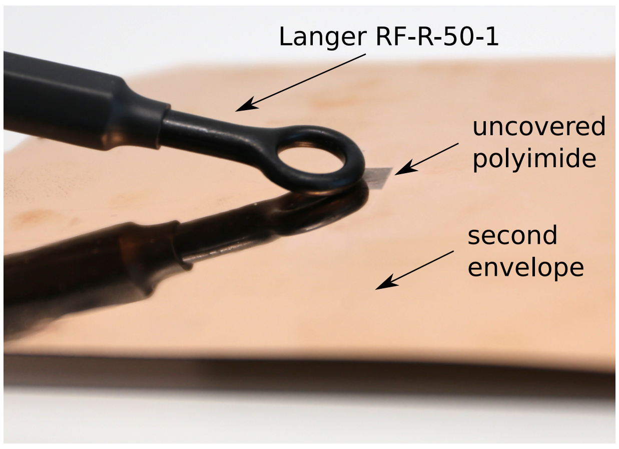


**Figure 4.8:** H-probe on top of uncovered polyimide. Through galvanic removal, I dissolved an area of 5 mm × 5 mm from the Cu shielding.

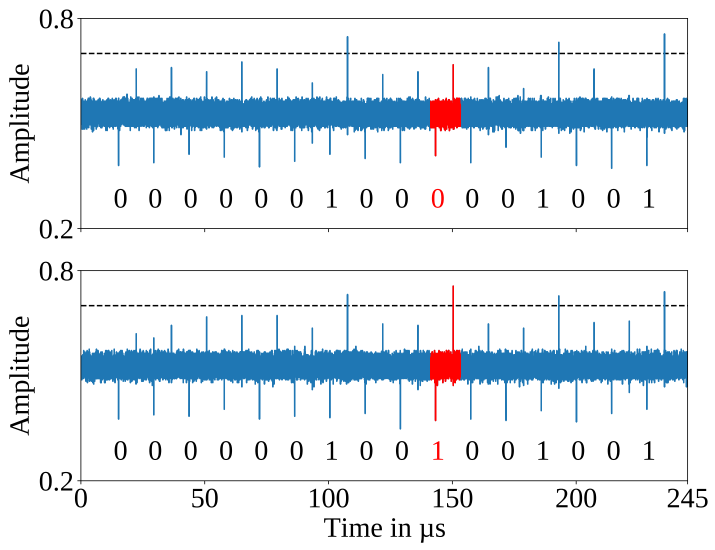


**Figure 4.9:** H-probe signals for the transmission of values `0x0209` (top) and `0x0249` (bottom) over the SPI-interface (payload only). The threshold distinguishing binary `0` from `1` is depicted as a dashed black line.

To automatically distinguish `0` bits from `1` bits, the attacker can define a threshold value for the full PUF-response, i.e., 128 differential capacitances. In general, the attack can be performed on both registers and SRAM data transmitted over the SPI interface. Furthermore, since the PUF-response is read out repeatedly, even additional signal noise could be compensated by recording multiple traces.

### 4.4.3 Countermeasures

The H-probe attack shows that a non-invasive read-out of the PUF-response is possible, which significantly weakens the security concept of the capacitive enclosure and requires adequate countermeasures.

A simple solution would be adding an extra shielding against the magnetic field. The shielding materials available, e.g., steel or mu-metal [Jil98], depend on the desired permeability. Replacing the envelope shielding with these materials is not feasible, whereas substituting the casing material might reduce the measured magnetic field emitted from components of the PCB. Nevertheless, a replacement of the casing will still not counteract the magnetic probing of the SPI interface.

A possible countermeasure is adjusting the ASIC's output driver strength so that the SPI communication edges are smoothened. However, optimizing the microcontroller's drive strength is more complex, if possible at all. Also, reducing the signal strength by decreasing the SPI voltage level is insufficient to counteract the attack. As HSMs stay in the field for a long time, e.g., $> 10$ years, future attack tools will likely detect even smaller magnetic fields.

An alternative approach is encrypting the communication between the microcontroller and the ASIC and exchanging the corresponding keys via Elliptic Curve Diffie Hellman Ephemeral (ECDHE) at each system boot. By choosing ECDHE, no prior exchange or persistent storage of certificates or keys is necessary, and thus, the authentication entirely occurs within the envelope's boundary. Symmetric encryption algorithms, such as the Advanced Encryption Standard (AES), without ECDHE key exchange requires the plain encryption key to be stored on both the microcontroller and the ASIC beforehand. This enables the attacker to record the encrypted PUF-response via magnetic probing and to read out the encryption keys after removing the envelope. Through this, the attacker can decrypt the PUF-response. Hence, a prior key exchange via ECDHE is crucial for securing the SPI interface. However, this comes with higher demands placed on the hardware and firmware of both the ASIC and the microcontroller. In general, the implementation of cryptographic primitives like ECDHE still has to be hardened on an algorithmic level against other side channels.

Another countermeasure is to remove the SPI interface entirely, in a future revision, by integrating the ASIC's functionality into the main microcontroller that is embedded into the envelope. In this case, the only remaining interfaces are the communication with the host system and the enclosure interface. Implementing additional encryption for this interface is more realistic since it is less time-critical and provides additional hardware capabilities. Furthermore, the SPI interface could be replaced by a differential interface reducing the emanated magnetic field [TIA01, Col02]. Differential interfaces, however, are not common in microcontrollers. Their integration into the ASIC is not straightforward and would require additional development effort.

## 4.5 Conclusion

In this chapter, I extended the enclosure's vulnerability analysis to three additional physical attacks. As a first step, I considered micro-drilling attacks with a diameter below 300 µm, investigating the suitability of various countermeasures. Subsequently, I proposed a countermeasure to Obermaier's surface probing attack, where he successfully intercepted the electrode current signal. The countermeasure can either be integrated into the envelope or the measurement circuitry. Finally, I investigated the envelope's susceptibility to electric and magnetic field probing. I demonstrated that the communication between the FORTRESS microcontroller and ASIC is vulnerable to magnetic probing attacks. Furthermore, I discussed the suitability of different measures to counteract the attack. The enclosure's tamper-sensitivity is restored through the proposed countermeasures, bringing it another step closer to its commercial deployment.

# Chapter 5

# The Wiretap Channel for PUF-Based Enclosures

In Chapter 3, I presented FORTRESS and its software, the EKMS, with full key management, post-processing, and key generation. To compensate for errors due to noise and environmental effects while incorporating tamper-sensitivity, I implemented Limited Magnitude Codes (LMCs), as described in Section 3.3. LMCs consider the LSBs of each PUF-symbol, thereby correcting only shifts to neighboring intervals and ignoring large offsets caused by an attack. However, one of their shortcomings is that they require a base change, which limits the choice of possible error correction parameters. Nevertheless, Limited Magnitude Codes provide a simple method to correct errors due to noise and environmental changes while incorporating tamper-sensitivity.

To consider more complex error patterns and changes in the PUF-distribution, I present an implementation of the wiretap channel via $q-$ary polar codes. This implementation provides a separate model for both environmental and attack effects while optimizing the entropy obtained from the PUF-response. Through this, the key generation and error correction of the PUF-response are further refined and developed.

## 5.1 Related Work and Overview

An overview of previous work in the field of PUFs covering error correction codes, was given in Section 2.4. In this section, I discuss wiretap code implementations for Physical Unclonable Functions and outline the contributions of this chapter.

### 5.1.1 The Wiretap Channel for Physical Unclonable Functions

Wiretap codes incorporate another security aspect besides the mere error correction capability. They achieve security by introducing additional randomness [Wyn75]. This makes them a promising candidate for PUFs. However, not all code classes are suitable for implementing a wiretap channel. Previous wiretap channel implementations were applied to binary silicon PUFs to hide secrecy leakage from unstable or biased PUF-bits [HÖ17, BY19, BY21]. Hiller and Önalan used wiretap coset coding to add randomness to the helper data of silicon PUFs, where they considered code lengths $\leq 64$. In recent publications, this idea has been extended to binary polar codes [BY19, BY21].

Compared to previous work, applying wiretap codes to capacitive PUF-based enclosures comes with major differences in the attacker model and implementation. The goal of applying wiretap codes to the enclosure is to enhance physical layer security instead of debiasing. Previous work focused solely on leaky helper data, assuming an error-induced PUF-response, while tamper-sensitivity was not considered. With the wiretap code presented in this chapter, noise, environmental effects, and drilling attacks are modeled separately, enabling fine-tuning the code construction to the desired security level (see Section 5.3). Furthermore, previous wiretap codes operated entirely on binary codes and silicon PUFs. As I have shown in Chapter 3, the enclosure's PUF-response is analog; hence, $q-$ary polar codes are a natural choice for extracting more entropy from the PUF-response compared to the binary case.

### 5.1.2 Outline and Structure

In the following, I present a wiretap channel implementation for the capacitive enclosure based on $q-$ary polar codes. As the first step in Section 5.2, I analyze how temperature effects and drilling attacks affect the PUF-distribution and derive a system model of the enclosure (see Subsection 5.2.1). With the system model as a starting point, I analyze the impact of different choices of quantization and key generation schemes.

In Section 5.3, I discuss the construction of a wiretap channel with $q-$ary polar codes based on the attacker model and the PUF-data analysis. For the derived PUF-secret with a length of 306 bits, the constructed wiretap code reaches a physical layer security of 100 bits, which is verified in a Monte Carlo simulation. Finally, I conclude this work in Section 5.4.

I presented the results herein at the 2022 Conference on Cryptographic Hardware and Embedded Systems (CHES). Further information is available in the IACR Transactions on Cryptographic Hardware and Embedded Systems 2022 [GXKF22].

## 5.2 System Model

The construction of a wiretap code requires a realistic enclosure model, which is discussed in this section. In Subsection 5.2.1, I analyze how the PUF-response and its distribution are affected by temperature effects and drilling attacks with a diameter of 250-300 µm. This analysis is based on measured PUF-data obtained from the COVER [ION$^{+}$19] and the post-processing steps discussed in Section 2.4. In Subsection 5.2.2, I calculate the distortion and error for different choices of quantization, estimating the error probability of the corresponding channel model. Furthermore, I demonstrate the unsuitability of binary quantization for the construction of a wiretap code. This is followed in Subsection 5.2.3 by a discussion of different key generation schemes suitable for constructing a wiretap channel.

### 5.2.1 Analysis of the PUF-Response

The data for the following analysis was obtained from the COVER [ION$^+$19] and measured with the discrete measurement circuit. As discussed in Section 4.3, the "in-enclosure" measurement of the discrete circuit is more accurate than the subtraction of absolute capacitances implemented in the ASIC. Since a future development goal is to deploy the "in-enclosure" concept on the ASIC, I focus on the more accurate data obtained from the discrete measurement circuit for the following analysis.

Two versions of the enclosure system have been developed: B-TREPID, which is wrapped around the entire device, and the COVER, which covers the top and bottom area of the PCB. Details of the enclosure design were discussed in Section 2.4. For B-TREPID, data from 50 envelopes are available, while the data set for COVER is more extensive, comprising 115 enclosures. Both designs behave similarly, resulting in Gaussian distributed PUF-responses with capacitances in the femtofarad range. However, the standard deviation for the COVER is greater compared to B-TREPID. Furthermore, the PUF-responses were measured with the discrete measurement circuit, which — compared to the ASIC — is optimized in terms of accuracy rather than size.

During the analysis, the measured PUF data were subjected to all post-processing steps described in Section 2.4. For the COVER, the resulting differential capacitances are Gaussian distributed over the interval $[-10000, +10000]$ with a standard deviation of $\sigma = 30\,\text{fF}$, which corresponds to 2241 points after the TX group shift (normalization) [ION$^+$19]. One point is equivalent to a digital resolution of 13.4 aF [Obe19]. Even though the discrete measurement circuit is optimized through low-noise components and filtering, the full setup with the enclosure attached still leads to a certain amount of noise, which has to be compensated by the quantization and error correction code. The measured noise — obtained from 200 consecutive measurements of the same enclosure — is Gaussian with a standard deviation of 1.7 fF, corresponding to 129 points.

The PUF-response is influenced by external effects, such as changes in temperature, humidity, or vibrations, which can alter the measured differential capacitances. Temperature measurements of the raw PUF-response have been published by Immler et al. for the COVER [ION$^+$19], which are the basis for the following analysis. Internal measurements of B-TREPID in a climatic chamber indicate that a high humidity leads to temporary changes in the PUF-response, which are reversed after dehumidification of the envelope. Furthermore, strong vibrations also alter the measured PUF-response. This is attributed to the vibration-sensitive measurement circuit and can be overcome by optimizing the circuit design. Another external influence is electromagnetic interference. The enclosure design and the measurement circuit were adjusted accordingly to prevent electromagnetic radiation from interfering with the capacitive measurement. A Cu shielding added to the enclosure's top and bottom layer counteracts electric fields, as shown in Chapter 4. External magnetic fields that could influence the capacitive measurement are counteracted by the narrow excitation frequency and canceled out by the meander structure of the electrode mesh. Previous measurements of the PUF-response cover temperature changes and drillings attacks [ION$^+$19].

## Temperature Distribution

The enclosure is susceptible to thermal changes that have to be compensated by error correction. Figure 5.1 depicts the thermal changes in the absolute capacitance — also referred to as *absolute nodes* — of a COVER, ranging from $-20\,°C$ to $60\,°C$. The 256 absolute nodes, each obtained from measuring one TX electrode against one RX electrode, behave similarly under thermal changes. Each line represents one absolute node measured over time and under thermal changes. The absolute capacitances in the picofarad range are mapped to approximately 2900 to 3700 points, where one point corresponds to a digital resolution of 11.8 fF. However, the differential capacitances in the femtofarad range are mapped to $[-10000, +10000]$ points, with one point corresponding to a digital resolution of 13.4 aF. In-enclosure subtraction of the absolute capacitances yields the corresponding PUF-response, i.e., 128 differential capacitances depicted in Figure 5.2a, where each line represents a single differential node. Figure 5.2c depicts the differential capacitance for each PUF-node between $-20\,°C$ and $60\,°C$. The data for each temperature was obtained over, on average, 100 measurements. Node numbers 0 to 15 denote the first TX group, i.e., electrodes TX1 and TX2 measured against electrodes RX1 to RX16; node numbers 16 to 31 represent the second TX group, and so forth. As shown in Figure 5.2c, certain PUF-nodes are more susceptible to thermal changes. This is likely due to stress in the enclosure material. The COVER's edges are slightly bent, which causes tension and makes the corresponding nodes more susceptible to environmental changes.

The post-processing steps described in Section 2.4 have a significant impact on the (processed) differential capacitances forwarded to the key generation. The first process-

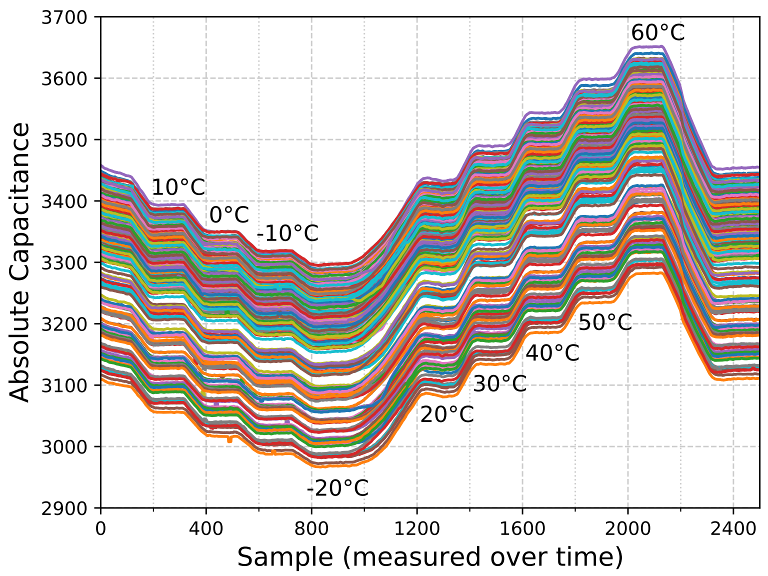
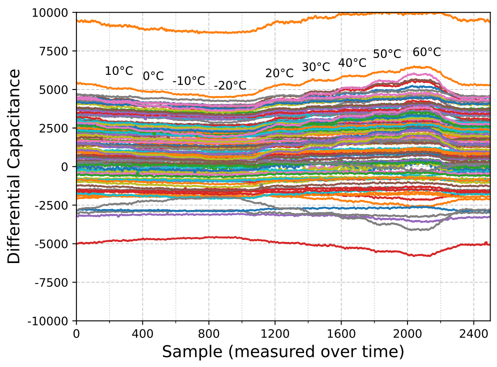
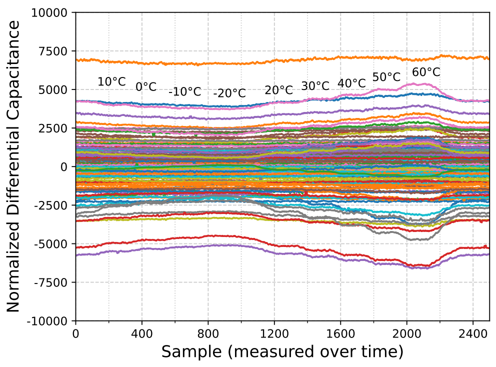


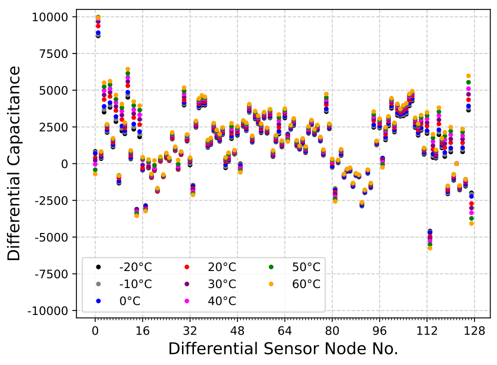
**Figure 5.1:** 256 absolute nodes (each depicted by a different color) influenced by temperatures in the range $[-20\,°C, 60\,°C]$. PUF-data taken from [ION+19].



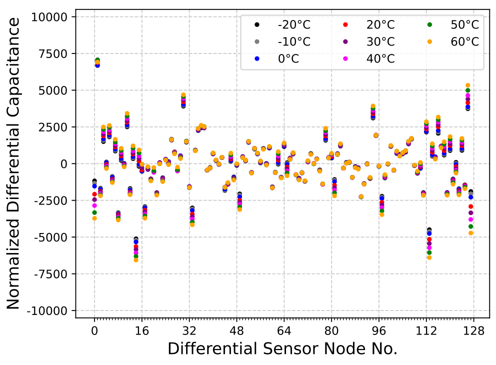
**(a)** *Unprocessed* differential capacitance under thermal changes. Temperatures as in Figure 5.1. The PUF-data was taken from previous COVER measurements [ION+19].



**(b)** Differential capacitance under temperature change (as in Figure 5.1) *after* the first processing step (normalization), shifting the TX group mean.



**(c)** Differential capacitance under temperature changes. The data for each temperature was obtained over, on average, 100 measurements.

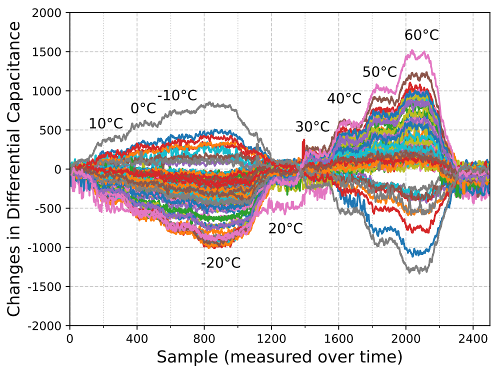


**(d)** (Normalized) differential capacitance under temperature changes. The data for each temperature was obtained over, on average, 100 measurements.

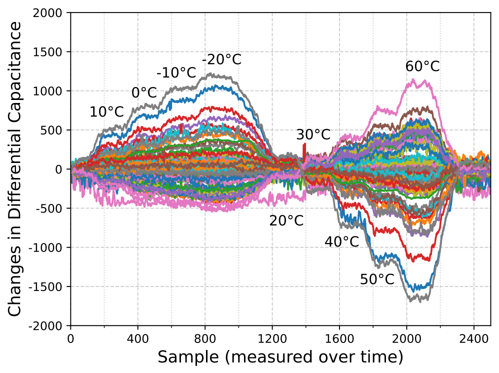
**Figure 5.2:** Analysis of the differential capacitance under temperature changes comparing the raw (left) and normalized (right) PUF-response.

ing step, the normalization, removes *global* manufacturing variations by subtracting the TX group mean. After normalization, outliers are shifted towards the center, and the distribution is narrowed (see Figures 5.2b and 5.2d). The narrowing of the distribution is also depicted in Figure 5.4.

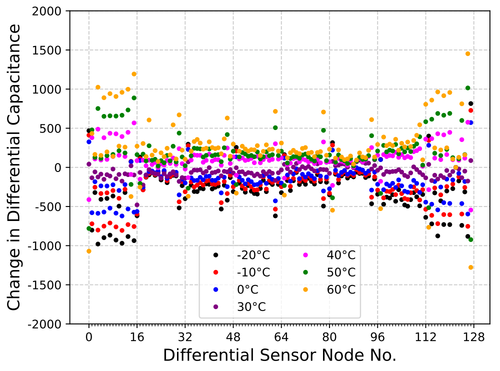
Figures 5.3a and 5.3c show how thermal changes affect the nodes of the PUF-response compared to a reference temperature of 20 °C. Certain PUF-nodes are more susceptible to thermal changes, as depicted in Figure 5.3c. This could be caused by mechanical stress where the COVER is bent. The maximal difference at 60 °C amounts to approximately 1500 points. Normalizing the PUF-response does not change the maximum difference,



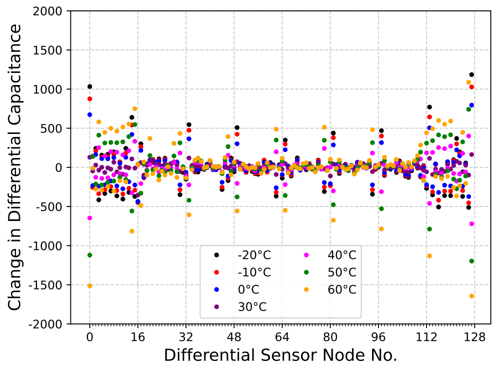
**(a)** Thermal changes of the *unprocessed* differential capacitance compared to the reference measurement at 20 °C.



**(b)** Thermal changes of the *normalized* differential capacitance compared to the reference measurement at 20 °C.



**(c)** Thermal changes of the *unprocessed* differential capacitance for each sensor node compared to the reference measurement at 20 °C. The data for each temperature was obtained over, on average, 100 measurements.



**(d)** Thermal changes of the *normalized* differential capacitance for each sensor node compared to the reference measurement at 20 °C. The data for each temperature was obtained over, on average, 100 measurements.

**Figure 5.3:** Analysis of changes of the differential capacitance under temperature changes comparing the raw (left) and normalized (right) PUF-response for a reference temperature of 20 °C.

as depicted in Figures 5.3b and 5.3d. Most node changes after normalization are in the range of $[-700, +700]$ points.

The PUF-distribution for temperatures between 0 °C and 60 °C is depicted in Figure 5.4. The data corresponding to each temperature was obtained by averaging approximately 100 measurements. On the top, the histogram of the raw differential capacitance is shown with Gaussian fits for all temperatures. The bottom depicts the differential capacitance after normalization, i.e., TX group shift. No plot was added for temperatures below 0 °C since the distributions behave equivalently to their non-negative counterpart. Hence, the distribution for $T = -10\,°\text{C}$ is almost identical to the Gaussian fit at

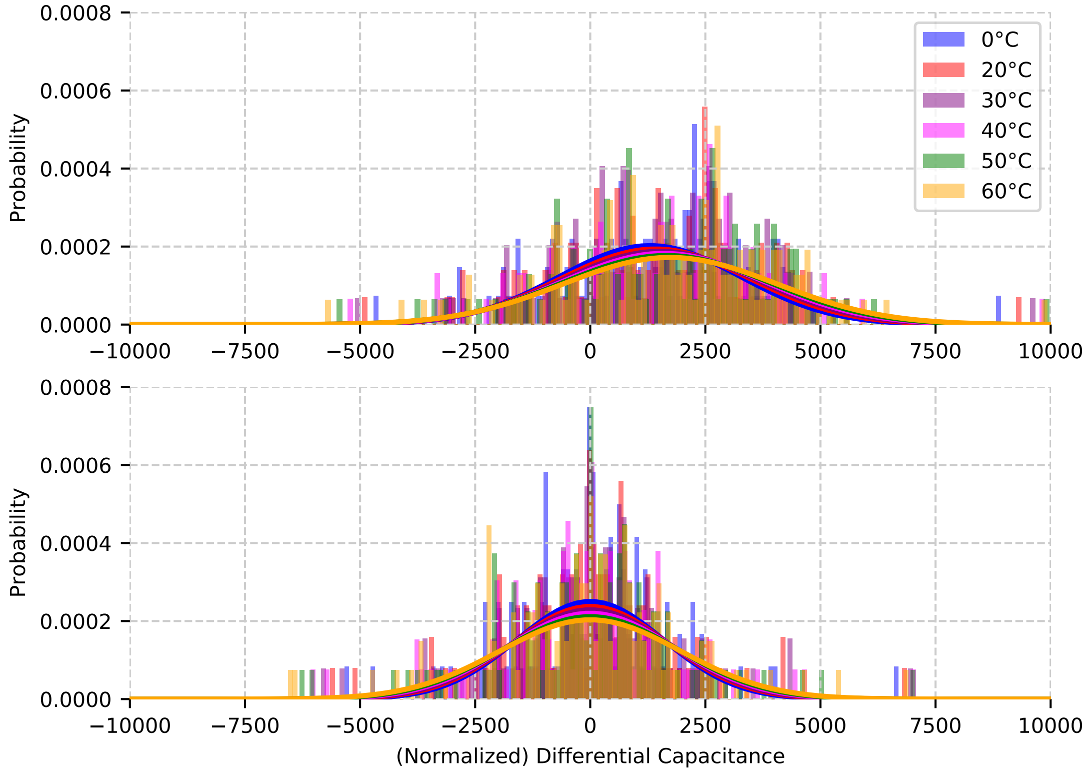


**Figure 5.4:** Histograms of the differential capacitance under temperature changes before and after normalization. Top: Distribution of the differential capacitance (with Gaussian fit) for temperatures between 0 °C and 60 °C *before* normalization. Bottom: Distribution of the differential capacitance (with Gaussian fit) for temperatures between 0 °C and 60 °C *after* normalization. The data for each temperature was obtained over, on average, 100 measurements.

$T = 10\,^\circ\mathrm{C}$. In both cases, i.e., before and after normalization, the distribution broadens as the temperature increases. However, through the TX group shifts, the distribution's standard deviation decreases ($\Delta\sigma = 207$ from $20\,^\circ\mathrm{C}$ to $60\,^\circ\mathrm{C}$) after normalization. In the unprocessed case, the increase in temperature shifts the PUF-distribution mean to the right.

Depending on reliability requirements for the particular use case, the enclosure will have to withstand large thermal changes in the field. For instance, the security policy of HSMs such as the HP Atalla Ax160 PCI HSM [Hew] generates a tamper event for temperatures outside the range $[-20\,^\circ\mathrm{C}, 100\,^\circ\mathrm{C}]$. Hence, the envelope or the COVER protecting the HSM has to withstand thermal changes outside the range $[-20\,^\circ\mathrm{C}, 60\,^\circ\mathrm{C}]$. The histogram in Figure 5.4 shows that larger temperature changes are expected to lead to greater offsets in the PUF-response, thus broadening the distribution even further.

### Attack Distribution

In Chapter 4, I gave an overview of the most relevant attacks in the context of the capacitive enclosure. With the current feature size, B-TREPID and COVER withstand drilling attacks with a diameter of 300 µm. Smaller holes are generally possible, but they have to be counteracted by additional measures, such as adding a potting or increasing the casing thickness. The post-processing and key generation alone can only recognize drill diameters of 250-300 µm that destroy at least two neighboring electrodes. This is depicted in Figure 5.5, which shows the change in differential capacitance before and after normalization.

Due to the attack, the affected nodes in the raw PUF-response will change significantly, leading to large offsets where nodes are shifted outside the defined interval $[-10000, +10000]$. Through normalization, these nodes are shifted back to the center of the PUF-distribution, thereby reducing the offsets. In general, nodes outside of the $[-10000, +10000]$ full-scale range — highlighted in gray in Figure 5.5 — are subjected to non-linear effects (clipping) during the capacitive measurement. However, depending on the enclosure sample and environmental influences, values outside the full-scale range could occur even during a regular measurement. Hence, a simple check for nodes outside the full-scale range is not sufficient to determine an attack. What is more, with decreasing drill diameters, the resulting holes might lead to smaller changes in the PUF-response.

A drilling attack affects the enclosure in two different ways. Firstly, the affected TX groups are subjected to burst errors. This is depicted in Figure 5.5, e.g., for the third TX

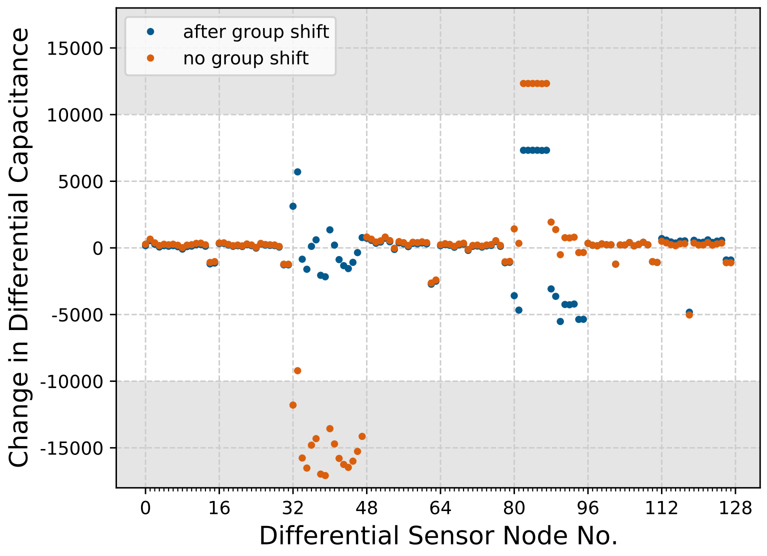


**Figure 5.5:** Drilling attack with a diameter of 250 µm affecting two TX groups before and after normalization. Values outside the full-scale range $[-10000, +10000]$ (highlighted in gray) are subject to non-linear effects.

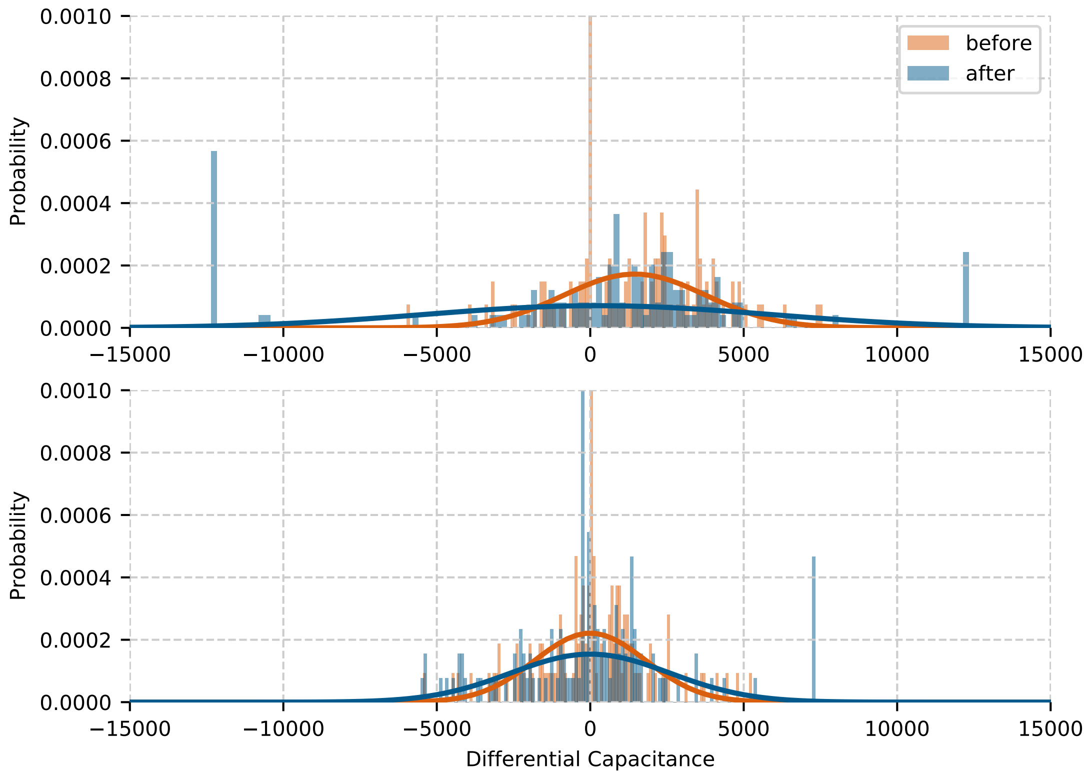


**Figure 5.6:** Effects of a drilling attack with 250 µm on the distribution of the PUF-response. Top: Distribution of the raw differential capacitance (with Gaussian fit) before and after the attack. Bottom: Distribution of the normalized PUF-response (with Gaussian fit) before and after the attack.

group (nodes 32 to 47), the PUF-nodes are "muddled up" through the attack. Secondly, the PUF-distribution is broadened through the drilling attack even after normalization. This is shown in Figure 5.6, depicting the histograms of the raw (top) and normalized (bottom) PUF-response before and after the attack. The unprocessed PUF-distribution broadens significantly, leading to a change in standard deviation by 3295 points. The goal of the normalization is to reduce offsets in the TX groups that leak information about the PUF-response (see Section 2.4). The normalization, i.e., subtraction of the newly calculated TX group offsets, is performed after each measurement. In general, this approach — to a certain degree — reduces the large offsets stemming from an attack. Alternatively, the initial offsets could be stored in NVM, which, however, leaks additional information about the PUF-response. Nonetheless, even after normalization, the distribution is still broadened compared to the untampered PUF-response since specific TX groups are more affected than others by the attack. In this case, the difference in standard deviation is reduced to 787 points, which still significantly exceeds the distribution broadening due to thermal changes.

### 5.2.2 Quantization of the PUF-Response

In the previous section, I analyzed how the normalized PUF-response is affected by thermal effects and drilling attacks. As described in Chapter 3, the next processing step after normalization is the quantization of the PUF-response. The choice of the quantization scheme has a major effect on the channel model and its error probability, as I will show in the following.

#### Previous Work on Quantization

Before analyzing different choices of quantization for the PUF-based security enclosure, I give an overview of previous work in the context of quantization. Quantization, in this case, represents an additional post-processing step performed on the already discretized PUF-response obtained after the ADC. The main goal of the quantization is to reduce the effects of noise during regular measurement.

Previous work covering the quantization of the 128 differential capacitances compared equidistant and equiprobable quantization intervals [IHKS16, IHL$^+$19, IU19, ION$^+$19]. Equidistant intervals distribute the discretized values unevenly, while in the case of equiprobable quantization intervals, the quantized values are uniformly distributed. This is depicted in Figure 5.7 for an exemplary distribution quantized into six intervals. In general, both schemes leak information about the PUF-response. In the case of equidistant quantization, information leakage is associated with varying symbol probabilities. However, if equiprobable intervals are chosen, the analog helper data that shift the discretized nodes to the middle of the interval leak information about the location of the nodes within the distribution.

Immler et al. focused on optimizing equidistant intervals through variable-length quantization that they tailored to Varshamov-Tenengolts (VT) codes [IHL$^+$19]. To optimize the per-bit minimum entropy, the PUF-response is mapped to binary values of variable length. However, one of the shortcomings of variable-length codes is increased error propagation if critical symbols or bits are lost or change during the transmission. If PUF-nodes change, the symbol length associated with the corresponding interval also changes. This makes it difficult to determine the beginning and end of a symbol. Therefore, synchronization correction is applied to counteract this, complicating the decoding process [GN98, CRR98].

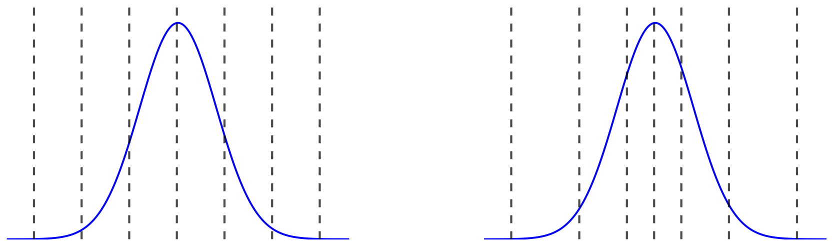


**Figure 5.7:** Equidistant (left) and equiprobable (right) quantization for six intervals.

In general, the quantization of noisy sources and the search for an optimal scheme is challenging [GN98]. In the following, I discuss the choice of different types of quantization considering thermal changes and the effect of attacks on the PUF-response.

### Binary Quantization and Wiretap Coding

When quantizing the PUF-response, two opposing goals have to be considered. First, a greater number of intervals increases accuracy and hence, the entropy of the PUF-key. However, secondly, if the number of quantization intervals is further increased, the quantized PUF-response becomes more susceptible to noise and can not be reliably reproduced. Hence, the quantization intervals have to be chosen such that the PUF-response is reliably reproducible while maximizing the resulting entropy.

Depending on the quantization scheme, the choice of intervals also determines the leakage of the PUF-response. In the case of equidistant quantization intervals, the varying symbol probabilities leak information about the PUF-response. On the other hand, equiprobable intervals leak information about the location of the PUF-node through the analog helper data. Hence, smaller shifts to the interval center indicate that the PUF-node resides within the center of the distribution. In both cases, leaking information about the PUF is unavoidable.

The quality of the chosen quantization can be estimated by calculating the distortion, which describes how well the original variable $x$ can be reproduced by its quantization $\hat{x}$ [GN98]. A simple and common distortion measure is the Mean Squared Error (MSE)

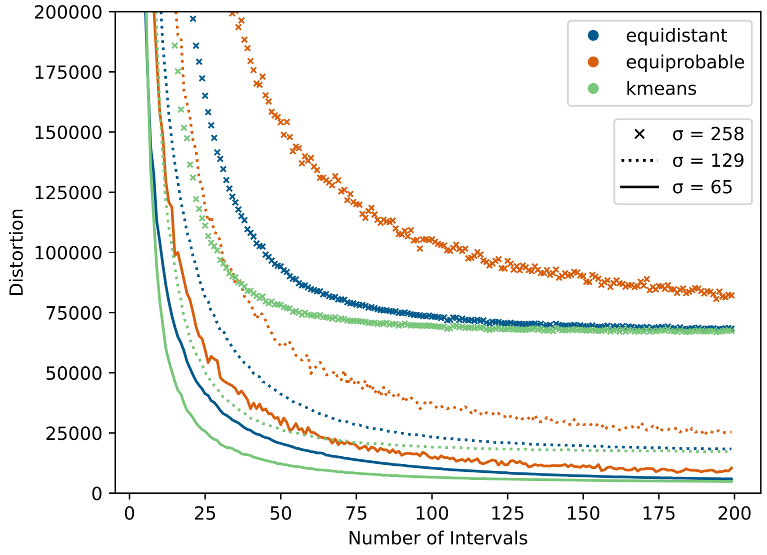


**Figure 5.8:** Simulation results for the distortion obtained from 1000 noisy PUF-responses for 5 to 200 intervals. The results were obtained for different noise distributions $\sigma_n = 65$, $\sigma_n = 129$, and $\sigma_n = 258$, and different types of quantization, i.e., equidistant (blue), equiprobable (red), and k-means (green).

$d(x, \hat{x}) = (x - \hat{x})^2$. For a sequence of length $n$, the MSE is defined as

$$d(x^n, \hat{x}^n) = \frac{1}{n} \sum_{i=1}^{n} d(x_i, \hat{x}_i) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{x}_i)^2$$

.

Figure 5.8 shows the distortion simulated for a set of 1000 PUF-responses influenced by different noise distributions for equidistant, equiprobable and k-means [kme20] quantization with 5 to 200 intervals. *k-means* is a common quantization method used in signal processing or machine learning. The noise is treated as additive Gaussian noise obtained from 200 consecutive measurements of the same enclosure at the same temperature. The noise standard deviation amounts to $\sigma_n = 129$ corresponding to 1.7 fF. The simulation was performed for $\sigma_n = 65$, $\sigma_n = 129$, or $\sigma_n = 258$ to observe the effect of different noise distributions. The distortion was calculated by considering PUF-responses of 128 symbols ranging between $[-10000, +10000]$ points. As depicted in Figure 5.8, after an initial decrease, the distortion of k-means and equidistant quantization approaches a constant value, which depends on the corresponding noise distribution. In case of equiprobable quantization, the decrease is less steep.

Figure 5.8 shows that the distortion of k-means and equidistant quantization has subsided at approximately 100 intervals. Hence, a quantization with 100 intervals will likely yield a good reproduction quality of the original PUF-response and high entropy. However, quantization with many intervals increases the number of errors to be corrected.

Figure 5.9 shows the number of interval shifts due to thermal changes and drilling attacks, simulated for 1000 PUF-responses. The parameters for all simulations were

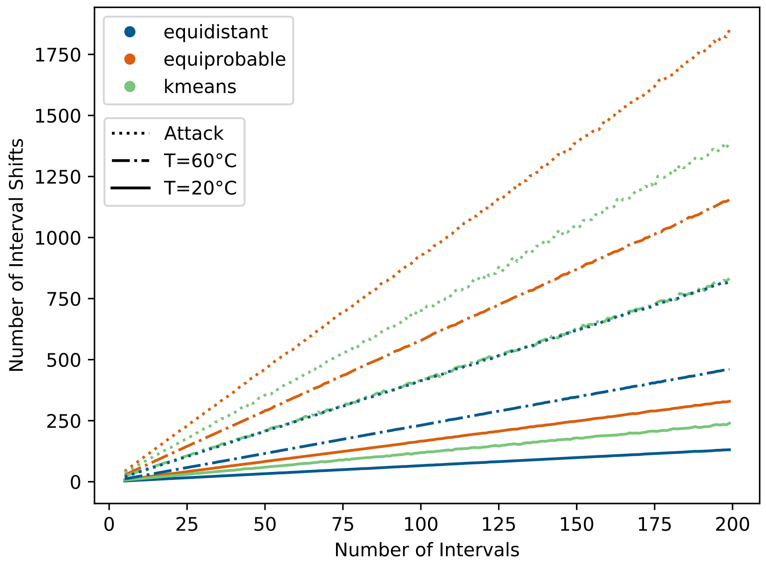


**Figure 5.9:** Simulation results for the number of interval shifts within a PUF-response. For each number of intervals, 1000 PUF-responses were simulated separately for $T = 20\,°\mathrm{C}$, $T = 60\,°\mathrm{C}$ and a drilling attack.

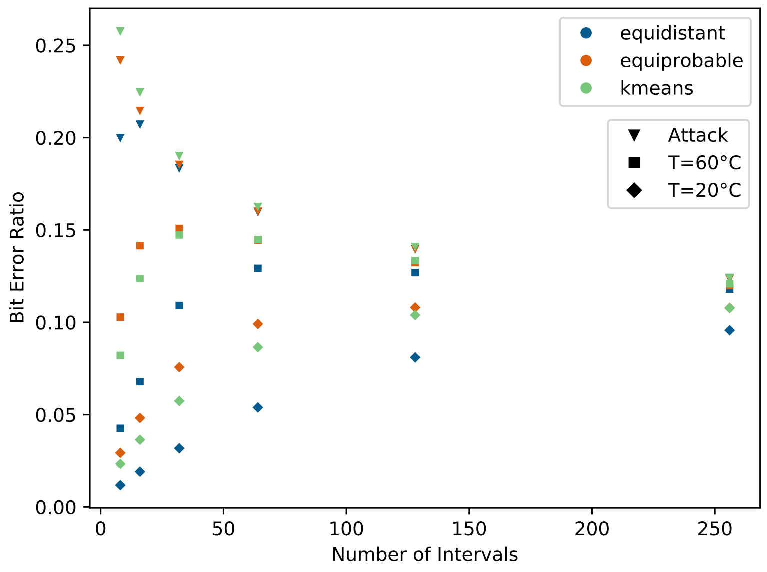


**Figure 5.10:** Simulation results for the Bit Error Ratio (BER) and Gray Code mapping, comparing equidistant (blue), equiprobable (red) and k-means (green) quantization for $T = 20\,°\mathrm{C}$ and $T = 60\,°\mathrm{C}$ to a drilling attack.

taken from the analysis in Subsection 5.2.1. This includes the distribution of the PUF-response at $T = 20°$ and $T = 60°$, considering Gaussian noise and changes due to an attack. Equidistant quantization has the lowest probability of interval shifts occurring. Since the innermost intervals of equiprobable or k-means quantization are very narrow, small shifts have a higher impact compared to equidistant intervals. The occurrence of interval shifts, i.e., changes in the quantized PUF symbols, depends on the interval width. The larger the size of the interval, the less likely an interval shift is about to occur. The interval width, however, depends on the number of chosen intervals. Figure 5.9 shows this linear relation, where the number of interval shifts increases — for both thermal changes and drilling attacks — as the number of intervals grows.

A natural representation of the quantized PUF-response through a binary encoding is a Gray code mapping, where the encoding minimizes the distance between intervals. The Bit Error Ratio (BER), i.e., the ratio of erroneous bits within the PUF-response, for Gray code mapping is depicted in Figure 5.10 simulated for a dataset of 1000 PUF-responses at $T = 20\,°\mathrm{C}$, $T = 60\,°\mathrm{C}$, and for a drilling attack. Figure 5.10 shows that as the number of intervals increases, distinguishing the BER of a PUF-response under thermal changes from a PUF-response influenced by a drilling attack becomes increasingly difficult since all curves approach the same BER. The Gray code maps each interval to a binary number of $\log_2(m)$ bits, where $m$ is the number of intervals. The reason why with an increasing amount of intervals, the difference in BER decreases is due to the Gray code mapping. Within the neighboring two intervals, an interval change, i.e., a bit flip, increases the Hamming distance by one. However, a shift of more than two intervals does not necessarily lead to a further increase of the Hamming distance, as depicted

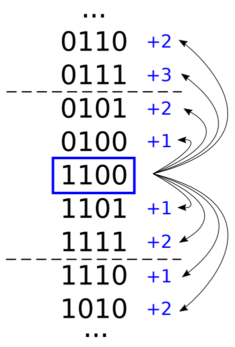


**Figure 5.11:** Example of a 4-bit Gray code. Shifts of more than two intervals do not increase the Hamming distance any further.

in Figure 5.11. This is unfavorable, especially for attacks that lead to large interval changes.

In previous publications 40 equidistant intervals were chosen for the enclosure with an interval width of 500 points [IHKS16, ION+19], corresponding to $3.9\sigma_n$. The interval width was chosen to entail 99.99% of the noise distribution.

Considering this recommendation and the above results for binary encoding, Figure 5.8 shows that for 40 intervals, a small distortion has not yet been reached. Furthermore, the difference in the number of interval shifts for different external influences, as depicted in Figure 5.9, is still relatively small for 40 intervals. Even though the distortion for equidistant and k-means quantization diminishes sufficiently at approximately 100 intervals, their BERs for 128 intervals can hardly be distinguished for different external influences. I observed the same behavior for consecutive numbering and different types of Gray codes, such as non-local or balanced Gray codes. .

These results show that binary encoding benefits the attacker when the number of intervals increases. However, a small number of intervals comes at the expense of accuracy and leads to entropy loss. Hence, in general, choosing a binary encoding for the quantization of the PUF-response is unfavorable for constructing a wiretap channel. Modeling the PUF through a $q-$ary channel is more suitable for the enclosure since the PUF-response is analog in nature. This ensures the distinction of both thermal influences and attack effects.

Considering the PUF post-processing, as discussed in Section 2.4, the quantization represents the final step. From the quantized PUF-response, the PUF-key is derived. In the following, I give an overview of different key generation schemes and their suitability for the PUF-based enclosure.

## 5.2.3 Key Derivation

Key generation schemes can include reliability data about the PUF (pointer-based schemes) or derive a key without considering the properties of the PUF (linear schemes). Pointer-based schemes [YD10, HMSS12, HWRL+13, YHD15] choose the most reliable PUF-bits for key derivation while discarding the remainder of the PUF-response. In the context of the capacitive enclosure, this constitutes a major disadvantage. The discarded PUF-nodes that are not considered in the codeword constitute "blind spots" that an attacker can easily target.

Linear schemes do not include additional information about the PUF or discard PUF-bits and hence, derive the key from the full PUF-response. One of the first key derivation schemes, published by Juels and Wattenberg, is the Fuzzy Commitment [JW99]. It was proposed in the context of biometrics for authentication of human fingerprints, whose repeated scans are prone to errors. A version of the Fuzzy Commitment that was extended by PUF post-processing as described in Chapter 3 is depicted in Figure 5.12.

As already discussed in Chapter 3, the key derivation process consists of two major steps:

(i) The *enrollment* is the initial generation of the PUF-key and helper data at the manufacturer. In the case of the Fuzzy Commitment, the secret $S$, which constitutes a true random number $R$, is encoded as the codeword $C$. Post-processing the raw PUF-response $X$ yields the analog helper data $W'$ that are stored in NVM. The codeword $C$ is masked by the quantized PUF-response $\widehat{X} = q(X)$, leading to the helper data $W$ that are also stored in NVM.

(ii) The *reproduction* is the continuous re-generation of the PUF-secret from the repeatedly measured PUF-response during regular operation. The PUF-response is influenced by noise $\epsilon_n$, thermal changes $\epsilon_t$, and drilling attacks $\epsilon_a$. After post-processing, the quantized PUF-response $\widehat{X}'$, together with the stored helper data $W$, generates the codeword, which, when decoded, results in the secret $S$.

Figure 5.12 shows that the repeated reproduction of the PUF-secret can be considered a "channel". Hence, a faulty codeword is "transmitted" over a noisy channel. As discussed in Subsection 5.2.2, a binary channel is not favorable for modeling the capacitive enclosure since it benefits the attacker. However, the Fuzzy Commitment can also be implemented for $q-$ary alphabets, e.g., through Reed Solomon codes [CS16]. This is extended to $q-$ary polar codes in Section 5.3.

The Fuzzy Extractor [DRS04] is a scheme similar to the Fuzzy Commitment. The main difference is that in the Fuzzy Extractor scheme, the PUF-response $\widehat{X}$ constitutes the secret $S$. This requires hashing of the secret to reduce helper data leakage.

Some schemes do not require the generation of a random number $R$, such as the Syndrome Construction [DRS04, DORS08] or Systematic Low Leakage Coding (SLLC) [HYP15]. In the Syndrome Construction, the PUF-response and the secret are equivalent. Hence, just as in the Fuzzy Extractor, the secret is hashed to obtain the PUF-key. The helper data $W = \widehat{X}\, H^{\mathrm{T}}$ are defined through the parity check matrix $H$. Reconstructing the correct PUF-response consists of minimizing the error $e$ in $W = (\widehat{X}+e)\, H^{\mathrm{T}}$. The Syndrome Construction was implemented in the context of polar codes [CIW+17]; however, since this scheme does not include additional randomness, it is not well-suitable for constructing a wiretap code. A wiretap channel scenario comprises two channels: The main communication channel and the attacker channel. The introduction of additional randomness is essential for constructing a wiretap code and ensures that the attacker

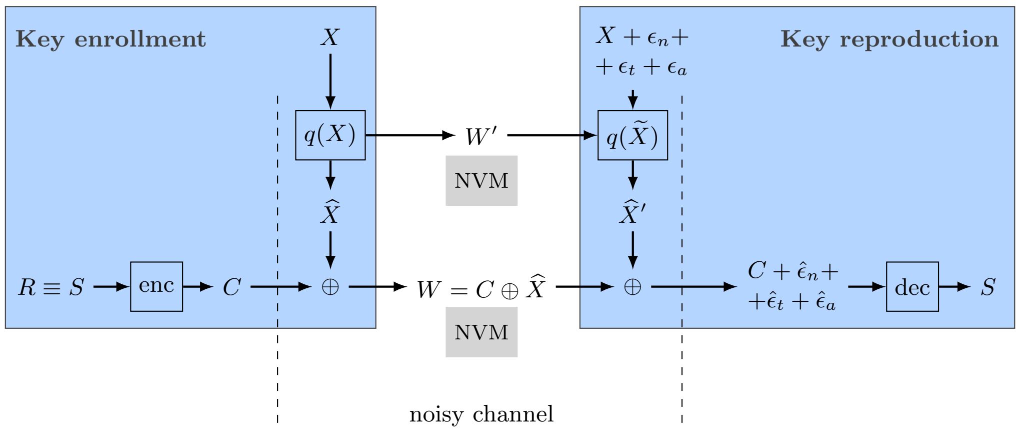


**Figure 5.12:** Fuzzy commitment scheme with post-processing and quantization $q(X)$ of the PUF-response $X$.

channel is significantly worse than the main channel. Hence, a key generation scheme has to be chosen that already considers an additional random number.

SLLC [HYP15] is another scheme that does not require an additional random number $R$ since it splits the PUF-response $X = X_\mathrm{S} + X_\mathrm{M}$ into the secret $X_\mathrm{S}$ and the mask $X_\mathrm{M}$. SLLC can only be applied to systematic codes that separate information and redundancy.

A major issue with many schemes for key derivation is helper data leakage. To overcome this issue, Müelich and Bossert [MB17] proposed a scheme based on the Secure Sketch that does not require additional helper data. Müelich and Bossert constructed the code such that the PUF-response corresponds to a codeword, omitting any helper data in the codeword generation. A drawback of their scheme is its complexity.

For a wiretap channel in the context of the capacitive enclosure, the Fuzzy Commitment, as depicted in Figure 5.12, is best suitable since:

(i) The PUF-secret is generated from a TRNG, enabling further enrollments and re-enrollments that are crucial for a secure system life cycle, as described in Chapter 3. Since the PUF-response only masks the secret, each enrollment generates a different KEK.

(ii) The wiretap channel requires additional randomness. However, in schemes such as the Syndrome Construction or SLLC, the helper data and secret depend solely on the PUF-response without considering additional randomness. Hence, key generation schemes that already include additional randomness by default are best suitable for constructing the wiretap code. It should be noted that randomness is also a necessary requirement for the secure life cycle introduced in Section 3.2.

In this section, I discussed the effects of thermal changes and drilling attacks on the PUF-response, followed by an investigation of quantization and key derivation. The next step is the construction of an encoder and decoder for the wiretap channel, which I discuss in the following section.

## 5.3 Construction of the Wiretap Code

A major requirement of the error correction code is the distinction between environmental changes and an attack. This distinction can be modeled through a wiretap channel [Wyn75, OW85, CK78]. In this section, I describe an adaption of the wiretap channel to the capacitive enclosure, which is implemented through polar codes.

### 5.3.1 The Wiretap Channel

The original wiretap channel, as published by Wyner [Wyn75], describes a wiretapper eavesdropping on a discrete, memoryless channel. Wyner assumed that the wiretapper eavesdrops on the transmission of the main channel via a second channel. The goal is to construct the encoder and decoder such that the level of confusion on the wiretapper's side is as high as possible. In choosing the equivocation as a measure of this level of confusion, Wyner found a trade-off between the transmission rate and the equivocation, as seen by the wiretapper. He showed that approximately perfect secrecy could be achieved for reliable transmission with a *secrecy capacity* $C_\mathrm{s} > 0$. The wiretap code is constructed to reliably transmit information on the main channel while hiding it on the second channel accessed by the wiretapper.

In the case of the capacitive enclosure, separately modeling the effects of thermal influences and drilling attacks is not feasible with a regular error correction code. To describe the complex error patterns and burst errors, as discussed in Section 5.2, a wiretap channel implemented based on polar codes is proposed in the following.

Figure 5.13 shows the wiretap channel adaptation for the capacitive enclosure. On the main channel, the codeword $C = \widehat{X} \oplus W$, affected by noise $\hat{\epsilon}_n$ and thermal changes $\hat{\epsilon}_t$, is "transmitted" with error probability $p_1$, resulting in the secret $S$. On the second channel, the PUF-response is additionally affected by drilling attacks $\hat{\epsilon}_a$, resulting in the error probability $p_2$ and the secret $S'$.

Previous wiretap channel implementations targeted leakage in binary PUFs caused by biased PUF-bits [HÖ17, BY19, BY21]. They chose their model such that the second

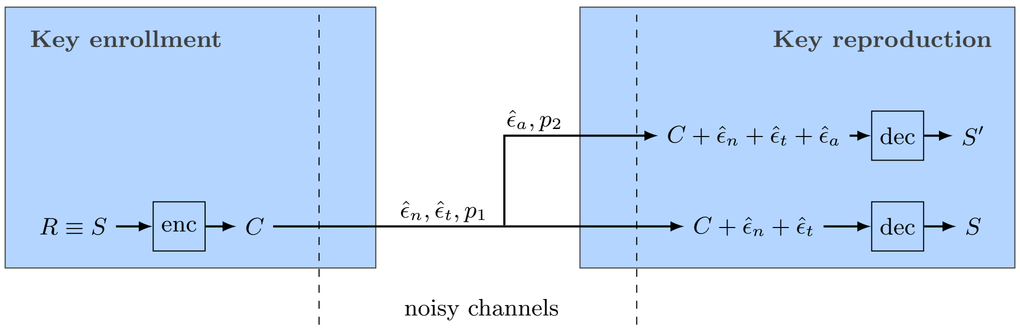

Key enrollment
Key reproduction
$R \equiv S$
enc
$C$
$\hat{\epsilon}_n, \hat{\epsilon}_t, p_1$
$\hat{\epsilon}_a, p_2$
$C + \hat{\epsilon}_n + \hat{\epsilon}_t + \hat{\epsilon}_a$
dec
$S'$
$C + \hat{\epsilon}_n + \hat{\epsilon}_t$
dec
$S$
noisy channels

**Figure 5.13:** The wiretap channel adapted to the capacitive enclosure considering measurement noise $\hat{\epsilon}_n$, thermal changes $\hat{\epsilon}_t$, and drilling attacks $\hat{\epsilon}_a$.

channel transmitted the distorted helper data. This is irrelevant in the case of the capacitive enclosure since the focus of the code construction is physical layer security. As depicted in Figure 5.13, the wiretap channel separates the regular transmission of the PUF-response on the main channel from an attack of the PUF-response on the second channel. Another major difference compared to previous implementations is that the code is designed for higher order alphabet PUFs since it is based on $q-$ary polar codes.

The attacker model itself was discussed in detail in Chapter 4, while the impact of fault attacks on the enclosure system is discussed in Chapter 6. Small holes of approximately 10 µm require additional countermeasures and can not be mitigated by adjusting the error correction code. Hence, the wiretap code construction discussed in the following has to reliably detect holes with a diameter of 250-300 µm.

### 5.3.2 $q-$ary Polar Code

Stolte was the first to introduce polar codes and investigate their similarity to Reed-Muller codes [Sto03]. A few years later, Arıkan reintroduced polar codes and showed that they achieved the capacity of a discrete memoryless channel (DMC) under successive cancellation (SC) decoding [Ari09]. This made the construction of capacity-achieving codes feasible and hence, drew much attention to polar codes. Arıkan's work was later extended to binary extension fields, for which Yuan and Steiner introduced a kernel construction [YS18]. In the context of the wiretap channel, polar code constructions with symmetric component channels have been shown to achieve capacity [MV11].

Basing the construction of the wiretap code on a binary channel is insufficient for the enclosure and benefits the attacker, as shown in Section 5.2. As the PUF-response is analog, a $q-$ary model is a more natural choice allowing for an increased tamper-sensitivity and higher extracted entropy. While previous polar code implementations [BY19, BY21] focused solely on binary PUFs, the generalized $q-$ary polar code for the enclosure extracts a longer PUF-secret and more entropy.

#### Encoder and Decoder

The binary polar transform, as published by Arıkan [Ari09], relies on a process called *channel polarization* that polarizes $n$ independent, identical copies of a binary-input DMC into better (noiseless) and worse (noisy) channels. This concept can be extended to binary extension fields. Figure 5.14 shows the polar code kernel defined over a finite field $\mathbb{F}_q$, as published by Yuan and Steiner [YS18]. The encoded $q-$ary symbols $(c_1, c_2)$

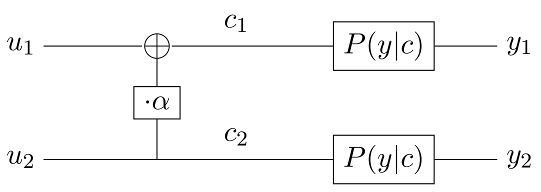


**Figure 5.14:** Depiction of the $2 \times 2$ kernel over a $q-$ary field, where $\alpha \in \mathbb{F}_q$.

relate to the $q-$ary source symbols $(u_1, u_2)$ via the polarization matrix $\mathrm{F}_2(\alpha)$

$$(c_1, c_2) = (u_1, u_2) \cdot \begin{pmatrix} 1 & 0 \\ \alpha & 1 \end{pmatrix} \triangleq (u_1, u_2) \cdot \mathrm{F}_2(\alpha) \tag{5.1}$$

with the optimization parameter $\alpha$ [YS18]. The first channel with input $u_1$ and output $(y_1, y_2)$ has a lower rate than the physical channel, defined by the channel law $P(y|c)$. The second channel with input $u_2$ and output $(y_1, y_2, u_1)$ has a higher rate than the physical channel [Ari09]. Figure 5.15 depicts the generalization for a code length $n > 2$. The relation between the information symbols $u = (u_1, u_2, u_3, u_4, \ldots, u_n)$ and the codeword $c = (c_1, c_2, c_3, c_4, \ldots, c_n)$ is defined as

$$c = u \cdot \mathrm{F}_2(\alpha)^{\otimes \log_2 n} \tag{5.2}$$

with the $\log_2 n-$fold Kronecker product $\mathrm{F}_2(\alpha)^{\otimes \log_2 n}$ of the polarization matrix $\mathrm{F}_2(\alpha)$ with itself. The received codeword is the channel output $y = (y_1, y_2, y_3, y_4, \ldots, y_n)$.

In his original work, Arıkan showed that in the limit $n \to \infty$, the channel polarization results in channels with either capacity 1 (perfect) or 0 (useless), which was later generalized to the $q-$ary case [PB12]. However, in practical applications with shorter code lengths $n$, the channel polarization also results in mediocre channels. The code construction, hence, aims at finding the best channels, and freezing the bad channels,

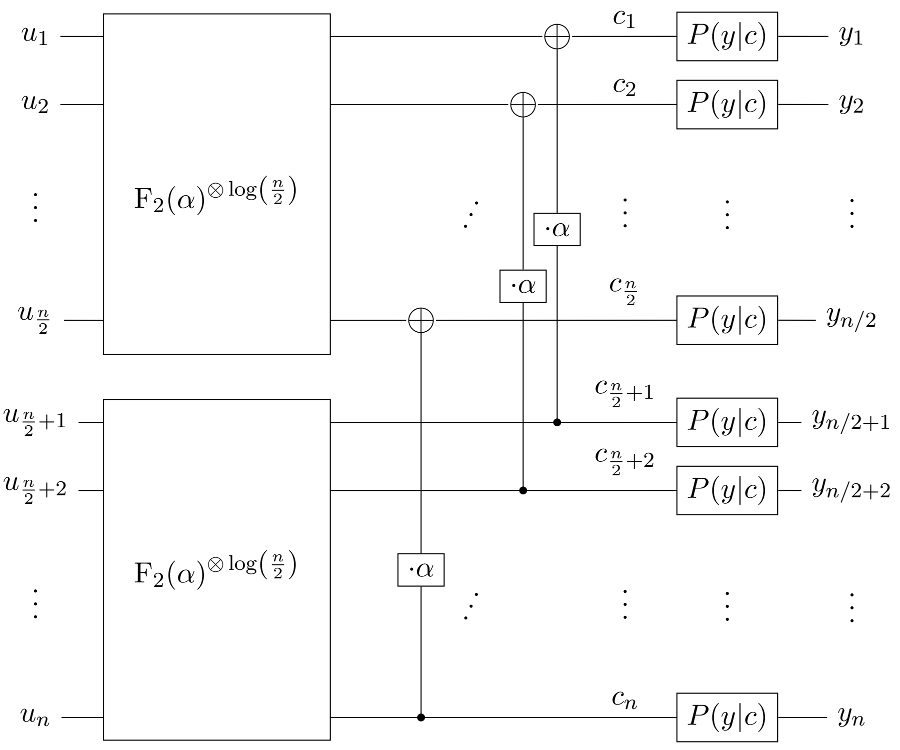


**Figure 5.15:** Depiction of the polar encoder.

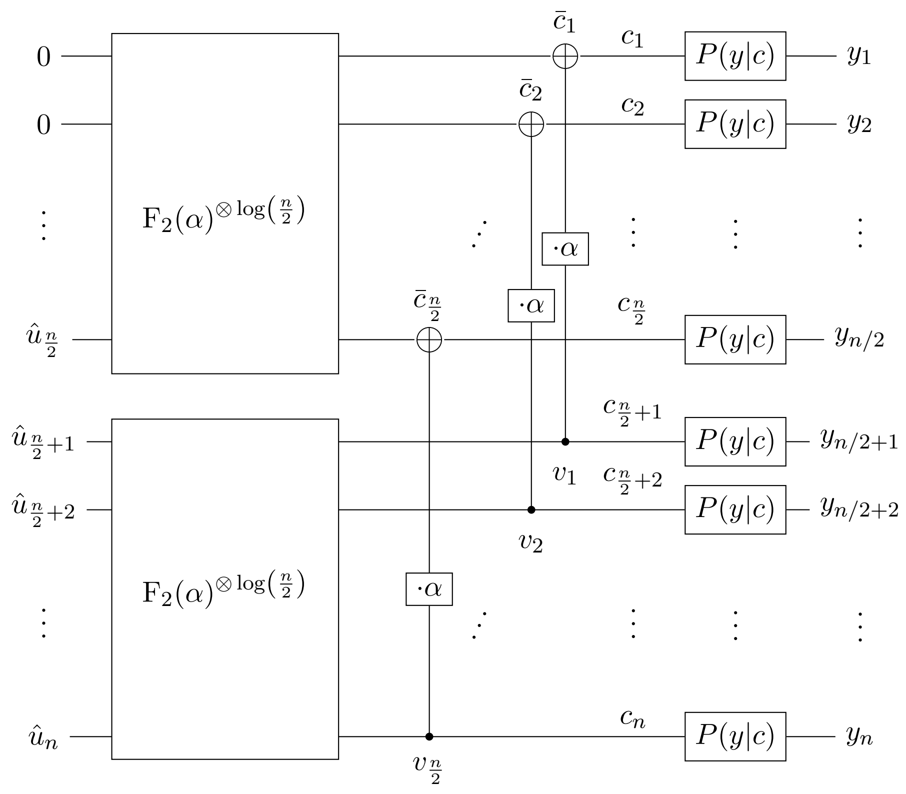


**Figure 5.16:** Depiction of the polar decoder.

i.e., transmitting the *frozen* value 0 that does not contain any information. The frozen positions are denoted by the set $\mathcal{F}$.

The decoder, as depicted in Figure 5.16, receives the erroneous codeword vector $y$ and the frozen positions $\mathcal{F}$ to estimate the source vector $\hat{u} = (\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \ldots, \hat{u}_n)$. Arikan originally proposed SC decoding, which allows for a recursive implementation and the use of soft information. The soft information propagates from right to left. *Check node* operations $\tilde{c}_i$ are depicted in Figure 5.16 through an XOR operation, and *variable node* operations $v_i$ by a bullet point. After the propagation of the soft information, a hard decision is made, which propagates from left to right, allowing for a successive decision on the most likely transmitted bits.

Hard decisions, however, can lead to the propagation of miscorrected symbols. To overcome this issue, Tal and Vardy proposed successive cancellation list (SCL) decoding [TV11], which allows pursuing multiple choices of information positions after a hard decision. For practical feasibility, however, the list size $L$ is limited. The most unlikely paths are discarded as soon as $L$ paths have been stored. Tal and Vardy showed that even a small list size of $L = 2$ already improves performance compared to SC decoding [TV11].

In the $q-$ary case, Yuan and Steiner modified the SCL decoding such that only paths within a reliability threshold $\delta$ are pursued [YS18]. This allows for the pruning of non-full lists and reduces the number of operations for transmissions over a good channel.

### Code Construction

The code construction determines the polarized channels that are "good" for the regular case — affected by noise and thermal changes — and "bad" for an attacker. Both types of channels are estimated by the Monte-Carlo code construction [Ari09]. However, as mentioned before, a full channel polarization is impossible due to the short code length $n$. For symmetric channel components, Mahdavifar and Vardy proposed a polar code construction achieving the secrecy capacity of the wiretap channel [MV11]. This construction freezes polarized symbol channels that are "very bad" for both the legitimate receiver and the eavesdropper. Information symbols are transmitted over polarized symbol channels that are "good" for the legitimate receiver and "bad" for the eavesdropper. The remainder of the polarized symbol channels that are "good" for both the legitimate receiver and the eavesdropper is used to transmit random symbols.

The channel model for the code construction is based on the analysis in Section 5.2. In an attack with a drill diameter of 250-300 µm, at least one TX and one RX electrode are destroyed. The natural choice for the enclosure is to quantize the PUF-response into $q-$ary symbols. Depending on the quantization intervals, the PUF-nodes subject to thermal changes can fall into the neighboring intervals. This behavior can be modeled through a $q-$ary channel with non-zero crossover probabilities $p_{i,j}$. I simulated this probability matrix with 100,000 PUF-responses considering all possible symbol changes for the legitimate and the attacker channel. The simulation yields the probabilities for each symbol changing into all other possible symbols, which serves as input for the code construction. I modeled the PUF-response based on the system analysis in Section 5.2, considering temperature changes, drilling attacks, and all post-processing steps, with and without analog helper data. The $8 \times 16$ differential PUF-nodes were quantized into 128 $q-$ary symbols. The $q-$ary polar code described in Subsection 5.3.2 is based on a discrete-time memoryless additive Gaussian white noise (AGWN) channel that requires independent and identically distributed input symbols [YS18]. Approaching this model, 8, 16, and 32 equiprobable intervals were chosen for quantizing the PUF-response to obtain uniformly distributed symbols. For the code construction, a bias in the input symbol distribution leads to additional leakage that has to be considered [CW19]. The simulation showed that the channel of the legitimate receiver is not symmetric since the $P(y_i|c_j) = p_{i,j}$ differ for $i, j \in \{0, 1, \ldots, q-1\}$.

For the code-offset construction, the polar code length amounts to $n = 128$, which is independent of the number of quantization intervals $q$. The code is constructed from a Monte Carlo simulation [Ari09]. The $n$ PUF-symbols are encoded into an $n$-symbol codeword, and noise is added according to the analysis in Section 5.2. For the Monte Carlo simulation, SC and SCL decoding were performed. For SC decoding, the received vector $y$ is decoded via the soft information of the channel model. After a hard decision, the decoder counts the occurrence of incorrect decisions to estimate the

**Table 5.1:** Extract of Monte Carlo simulation results for the $q-$ary polar code construction obtained with and without the consideration of analog helper data $W'$. The codeword consists of 128 symbols derived from $8 \times 16$ differential capacitances that are quantized into 8, 16, and 32 equiprobable intervals. Each interval, hence, corresponds to a dedicated $q$-ary symbol. The error correction reliability is indicated by the per-symbol error probability $d$. $n_s$ denotes the number of reliable symbols after error correction, of which the attacker can read $n_f$ symbols. The attacker entropy $H_{\text{att}}$ in bits is a measure for the achievable security level, while $H_{\text{secret}}$ corresponds to the bit-length of the PUF-secret.

| | Without $W'$ | | | | | With $W'$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $q$ | $d$ | $n_\text{s}$ | $n_\text{f}$ | $H_\text{att}$ | $H_\text{secret}$ | $d$ | $n_\text{s}$ | $n_\text{f}$ | $H_\text{att}$ | $H_\text{secret}$ |
| 8 | 0.0010 | 73 | 11 | 60.8 | 219 | 0.0010 | 119 | 22 | 151.9 | 357 |
| | 0.0005 | 71 | 11 | 55.6 | 213 | 0.0005 | 117 | 22 | 145.9 | 351 |
| | 0.0001 | 65 | 11 | 40.2 | 195 | 0.0001 | 112 | 22 | 130.9 | 336 |
| | - | | | | | $10^{-5}$ | 106 | 22 | 112.0 | 318 |
| | $10^{-6}$ | 56 | 11 | 22.1 | 168 | $\mathbf{10^{-6}}$ | **102** | **22** | **100.3** | **306** |
| | - | | | | | $< 10^{-6}$ | 101 | 22 | 98.0 | 303 |
| 16 | 0.0010 | 61 | 9 | 55.7 | 244 | 0.0010 | 82 | 17 | 92 | 328 |
| | 0.0005 | 58 | 9 | 47.8 | 232 | 0.0005 | 79 | 17 | 80.8 | 316 |
| | 0.0001 | 52 | 9 | 33.1 | 208 | 0.0001 | 74 | 17 | 63.9 | 296 |
| | - | | | | | $10^{-5}$ | 68 | 17 | 45.6 | 272 |
| | $10^{-6}$ | 45 | 9 | 15 | 180 | $10^{-6}$ | 64 | 17 | 34.9 | 256 |
| | - | | | | | $< 10^{-6}$ | 63 | 17 | 32.0 | 252 |
| 32 | 0.0010 | 68 | 11 | 111.4 | 340 | 0.0010 | 73 | 15 | 116.9 | 365 |
| | 0.0005 | 66 | 11 | 102.5 | 330 | 0.0005 | 72 | 15 | 112.4 | 360 |
| | 0.0001 | 62 | 11 | 89.1 | 310 | 0.0001 | 69 | 15 | 98.9 | 345 |
| | - | | | | | $10^{-5}$ | 62 | 15 | 74.0 | 310 |
| | $\mathbf{10^{-6}}$ | **55** | **11** | **57.3** | **275** | $10^{-6}$ | 58 | 15 | 58.9 | 290 |
| | - | | | | | $< 10^{-6}$ | 56 | 15 | 49.9 | 280 |

probability of an incorrectly decoded symbol $\hat{u}_i$ given correct previous hard decisions $(\hat{u}_1, \ldots, \hat{u}_{i-1}) = (u_1, \ldots, u_{i-1})$. The Monte Carlo simulation is run for both the legitimate and the attacker channel. For both, the "very bad" channels are frozen, where an erroneous decision is highly likely. Random symbols are transmitted over the "very good" channels, while information symbols are transmitted over the remaining channels. From the Monte Carlo simulation, the entropy for the attacker can be calculated: $s$ bits of entropy signify that the brute-force effort for an attacker amounts to $2^s$ options for the decoding. Hence, for the SCL decoder, the list size $L$ should be at least $L > 2^s$, including the correct codeword.

Table 5.1 shows the results from the Monte Carlo code construction for $q = 8$, 16, and 32 intervals, with and without analog helper data $W'$. The per-symbol error probability $d$ is a measure of the error correction reliability, which determines the number of reliable symbols $n_\text{s}$ after error correction. Of $n_\text{s}$ reliable symbols, $n_\text{f}$ can be read by an attacker.

**Table 5.2:** Simulation results for SC and SCL decoding at T=20 °C with and without analog helper data $W'$, for 8 and 32 intervals. $n_f$ of the $n_s$ correctly transmitted symbols are readable by the attacker. The decoding results yield the probability of the wrongly decoded PUF-secret (FER), the secret length $H_{\text{secret}}$ in bits, and the complexity of an attack $H_{\text{att}}$.

| Decoder | $W'$ | $q$ | FER | $n_{\text{s}}$ | $n_f$ | $H_{\text{secret}}$ | $H_{\text{att}}$ |
|---|---|---|---|---|---|---|---|
| SCD | yes | 8 | $4.0 \times 10^{-6}$ | 102 | 22 | 306 | 100 |
| SCL ($L = 8$) | yes | 8 | $1.0 \times 10^{-6}$ | 102 | 22 | 306 | 100 |
| SCD | no | 32 | $7.0 \times 10^{-6}$ | 55 | 11 | 275 | 57 |
| SCL ($L = 8$) | no | 32 | $3.3 \times 10^{-6}$ | 55 | 11 | 275 | 57 |

Choosing a maximum per-symbol error probability $d = 10^{-6}$, the parameter set with the largest entropies $H_{\text{att}}$ and $H_{\text{secret}}$ in both columns (with and without analog helper data $W'$) is selected. The entropy of the attacker channel $H_{\text{att}}$ is calculated from the legitimate symbol channels that have a probability for incorrect decoding below the threshold $d$, which is a reliability measure of the legitimate channel. The attacker entropy $H_{\text{att}}$ is related to the decoding complexity and the corresponding brute force effort considering the overall error correction scheme to be known by the attacker. Thus, $H_{\text{att}}$ determines the achievable security level of the code. In total, $n_{\text{s}}$ symbols, corresponding to the code dimension $k$, are selected that are good for the legitimate channel. The $n_{\text{s}}$ symbols contain an entropy of $H_{\text{secret}} = n_s \cdot \log_2(q)$ bits, corresponding to the bit length of the PUF-secret.

$n_{\text{f}}$ of the $n_{\text{s}}$ reliable symbols are randomized since they are "good" for the eavesdropper, which results in $n_{\text{s}} - n_{\text{f}}$ symbols of a high error rate received by the attacker. The physical layer security of the wiretap code is determined by the entropy of the attacker channel $H_{\text{att}} = -\sum_i^{n_s} p_{s,i} \log_2(p_{s,i})$, where $p_{s,i}$ denotes the symbol error rate after an attack. The choice of the threshold $d$ determines the security level since through a stricter selection, the number of reliable bits $n_s$ is reduced, together with the complexity for an attacker $H_{\text{att}}$.

For selected code construction parameters in Table 5.1, another Monte Carlo simulation was performed for SC and SCL decoding to obtain the Frame Error Rate (FER), i.e., the probability of the wrongly decoded PUF-secret. Table 5.2 shows the simulation results for $T = 20\,°\text{C}$. For a FER of $10^{-6}$ and 8 intervals, the PUF-secret length is 306 bits, and for 32 intervals, 275 bits. The complexity for an attacker amounts to 100 bits for 8 intervals, while for 32 intervals, the brute force effort reduces to $2^{57}$.

Polar decoding requires storing certain parameters in NVM, such as the indices of frozen bits. However, in case of a biased PUF-response, additional helper data can leak information about the PUF-key [WFP19]. An attacker could use this additional information to develop a strategy for guessing the key. The additional randomness of the wiretap code construction, however, increases the effort of guessing the correct key compared to a regular polar code implementation. Future work should investigate the general polar code related helper data leakage.

This polar code construction supports thermal changes in the range [+5 °C, +35 °C], which corresponds to the operating temperatures of state-of-the-art network HSMs [TG, Hew]. Hence, the herein presented code construction is a promising approach showing that sufficient entropy can be extracted with $q-$ary polar codes while achieving a high security level. It forms a basis for continued development, such as optimizing the PUF post-processing towards larger temperature ranges. Furthermore, it enables the consideration of additional environmental influences, such as changing humidity. The results could have been further improved by discarding the normalization that weakens the enclosure's tamper-sensitivity. However, this entails accepting significant information leakage about the PUF-response due to global manufacturing variations. Alternatively, the repeated calculation of TX group offsets could be prevented by storing the initial offsets in the enrollment phase as fixed helper data, which, however, leak information about the PUF-response. Hence, future work should focus on developing alternative methods to extract global manufacturing variations.

The suitability of binary polar codes for a low-cost implementation on a microcontroller was demonstrated by Kestel et al. [KFPW22]. The authors proposed a low-area (binary) polar decoder for PUF applications that requires a space of 4.200 µm$^2$ in a 22 nm technology. Compared to the binary case, the decoding complexity of $q-$ary polar codes increases depending on the number of quantization levels. In the binary case, SC decoding is performed in $\mathcal{O}(n \log n)$, whereas SCL decoding requires $\mathcal{O}(L \cdot n \log n)$ [TV11, BSPB15]. In the $q-$ary case, the check and variable node operations require an additional $\mathcal{O}(q \log q)$ and $\mathcal{O}(q)$. Hence, with a reasonably small number of intervals, the decoding can be performed on a common microcontroller. Future work should evaluate $q-$ary polar codes in terms of code size and performance optimizations to tailor them to the low-cost demands of Physical Unclonable Functions.

## 5.4 Conclusion

In this chapter, I presented a wiretap code for the capacitive PUF-based security enclosure implemented via $q-$ary polar codes.

I provided a data analysis of how thermal changes and drilling attacks affect the PUF-distribution and all post-processing steps. From this analysis, I derived a model of the enclosure serving as the foundation of the code construction. The constructed polar code incorporates the analog nature of the PUF-response through a $q-$ary channel model and equiprobable quantization of the higher order alphabet. The wiretap code enabled the selection of the best symbol channels for the legitimate receiver and minimized the good channels for the attacker. Through this, a physical layer security of 100 bits was achieved for a PUF-secret length of 306 bits, 8 intervals, and a temperature range of [+5 °C, +35 °C].

Future work should tackle the influence of further environmental influences on the enclosure, such as humidity or vibrations, and optimize the enclosure and code design to larger temperature changes. Furthermore, the implementation should be optimized for execution on a common microcontroller.

# Chapter 6

# Resistance of FORTRESS Against Fault Attacks

Chapter 4 focused on physical attacks in the context of the capacitive PUF-based security enclosure. This chapter covers the software side of FORTRESS by analyzing the effects of injected faults and discussing countermeasures to harden the system against fault attacks.

Section 6.1 provides an overview of basic terms and different types of fault attacks, with a focus on radiation faults. This is followed in Section 6.2 by an evaluation of how faults can affect critical algorithms through the injection of data and instruction faults.

To analyze the software of FORTRESS, I present ARCHIE, a QEMU-based fault injection framework for ARCHitecture-Independent Evaluation (ARCHIE) of faults, in Section 6.3. With the help of ARCHIE, I analyze the software side of FORTRESS to find weaknesses in the implementation in Section 6.4. This is followed in Section 6.5 by a discussion of various countermeasures. Finally, I conclude my results in Section 6.6.

## 6.1 Fault Injection

In this section, I introduce basic terms in the context of fault injection. Furthermore, I analyze different sources of faults considering their impact on the capacitive PUF-based enclosure.

### 6.1.1 Basic Terms

Embedded devices are subject to faults that affect their correct operation. Faults are defined as low-level changes in the physical state. Faults that propagate and affect the internal state are referred to as errors [Nyb18]. A failure occurs if, through the influence of the error, the system's intended behavior is no longer fulfilled.

In safety applications, failures are caused by either hard or soft errors [Lav08]. Hard failures originate from internal causes, e.g., defects in the circuit, while soft errors originate in external sources, e.g., particles affecting the stored and processed data.

Apart from accidental or random faults, for instance, through aging or environmental effects, deliberate fault injection through an attacker can severely impact cryptographic algorithms and leak CSPs.

Faults in a security context stem from invasive, semi-invasive, or non-invasive attacks, depending on the degree of hardware modifications during device preparation. Semi-invasive attacks, like, for instance, Laser Fault Injection (LFI) or other energy beams, were already discussed in Chapter 4. In general, the FORTRESS components within the enclosure are non-accessible. Even if holes with a small aspect ratio are technically feasible, countermeasures, such as increasing the casing thickness, adding a potting material, or reducing the enclosure's feature size hinder an attacker from probing critical components on the PCB. Furthermore, semi-invasive techniques require the preparation of the DuT through decapsulation or thinning of the substrate.

Non-invasive methods, such as electromagnetic pulses or deliberate temperature and voltage changes, do not require changes to the DuT's hardware. The envelope' shielding protects against alternating electric fields. Magnetic fields, however, permeate the enclosure as demonstrated in Chapter 4 but are unlikely to influence the capacitive measurement. They are counteracted by the narrow excitation frequency and canceled out through the meander structure of the Tx and RX electrodes. Strong EM-fields, in contrast, will most likely destroy the device and are unsuitable for repeated fault injection.

The only externally accessible component is the power supply. Deliberate changes in external temperature or voltage, leading to glitches, can be counteracted by sensor monitoring. Furthermore, voltage glitching is hampered through the electronic components of the power supply itself, such as large capacitors or voltage stabilizers. Faults induced by clock glitching [KHEB14, RGP22] can be ruled out since the enclosure protects the clock access.

There is, however, a non-invasive attack able to fault the enclosure system. Different types of radiation can permeate the envelope and cause faults in critical electronic components, as I will show in the following.

### 6.1.2 Faults Induced Through Radiation

Apart from glitching, physical fault injection attacks in the security context are often based on Laser Fault Injection (LFI) or Electromagnetic Fault Injection (EMFI). Laser Fault Injection requires the preparation of the DuT, which usually entails decapsulation or grinding and polishing. Laser-based faults can severely impact the execution of cryptographic algorithms, as I will show in Section 6.2. EMFI is a non-invasive method that does not require decapsulation of the DuT. An electromagnetic pulse generates variations in the magnetic fields surrounding the target device, which induces parasitic currents that disrupt the DuT's operation [DLM19]. However, the occurrence of faults in EMFI is not fully understood. Models of how faults are generated are difficult to verify since EMFI pollutes all measurements in the vicinity of several meters [DLM19]. Even though the fault causality is not fully clear, EMFI has been successful in attacking cryptographic algorithms [SH07, DDRT12, DDR+12, RNR+15]. Radiation-based energy probes could be the next step to combine the advantages of a focused laser beam with the non-invasive approach of EMFI. However, using other energy probes and radiation sources is also a financial topic since an EMFI setup is much more affordable than devices generating electron, neutron, proton, or $\gamma$ radiation. However, in reliability testing, ionizing radiation, such as neutron beams, has already been used to irradiate test devices in Accelerated Life Testing (ALT) [VPK+15]. Hence, applying these methods to a security-focused analysis could be the next possible step. An attacker targeting the capacitive PUF-based security enclosure could buy additional devices for reverse-engineering and subsequently launch an attack through a particle beam or other types of radiation. The question is if different radiation types can permeate the enclosure and reach critical components. The following analysis aims to provide an answer to this question.

The envelope layer stack has a thickness of 0.25 mm, with 8.5 µm copper, 150 µm polyimide, and 100 µm adhesive tape 3M VHB9460. Depending on the attacker model, the PCB components are protected by a metal or polymer casing with varying thickness. Assuming that the enclosure size is scalable, the casing thickness and material can be chosen so that the electronic components are shielded from different types of radiation. However, the ASIC resides on top of the casing and is covered only by the enclosure. Furthermore, to impede magnetic probing attacks, as conducted in Chapter 4, the design goal is to integrate the functionality of the main FORTRESS microcontroller into the ASIC. The ASIC, hence, is a potential weak point that could be reached by different types of radiation that have been shown to cause errors in memory.

Particle radiation can lead to data corruption and damage in electronic circuits [LaB04, Rau]. This was already known in 1979 and investigated by May and Woods, who showed that $\alpha$-particles penetrating the die surface could cause sufficient electron-hole pairs to induce errors in dynamic RAM [MW79]. Uranium and thorium, which are present to a certain degree in packaging materials, were found to be the source of $\alpha$-particles caused by radioactive decay. In 1997, Chou et al. investigated different causes of soft errors in Dynamic Random Access Memory (DRAM) induced by $\alpha$-particles (1997) [CCH97]. Electrically Erasable Programmable Read-Only Memory (EEPROM) can also be affected by $\alpha$- and $\gamma$-radiation. Carlo et al. exposed EEPROMs to a Co-60 gamma radiation source and an Am-241 alpha radiation source [CSS+11]. They observed soft errors induced by the $\alpha$-particles, while a long exposure to $\gamma$-rays led to severe damage causing device failure. As shown by Bagatin et al. [BGPFC12], flash memories are also susceptible to $\alpha$-particle induced soft errors. The authors decapsulated the target devices with an acid before the experiment to get access to flash memory. They showed that 5.4 MeV $\alpha$-particles induced raw bit errors in 50 nm NAND flash.

Also, protons and neutrons can cause soft errors. For example, in 1979, Wyatt et al. exposed dynamic memory devices to protons with energies ranging from 18 to 130 MeV [WMT+79]. They observed that soft error types and error location in memory depended on the beam energy. In a more recent experiment, Iwashita et al. determined the energy-dependent soft-error rate in SRAM caused by neutrons with an energy of 1-800 MeV [IFS+20]. The error rate gradually increased up to a neutron energy of 20 MeV. The authors also reported errors even below an energy of 2.5 MeV. Cellere et al. investigated neutron-induced soft errors in NAND and NOR flash memories [CPC+01]. They showed that neutrons trigger bit errors, particularly in multi-level cells (MLCs), where the number of bits per cell is higher than in single-level cell (SLC) devices. Also, devices with smaller feature sizes, in general, were more susceptible to neutron-induced bit errors.

Electrons inducing soft errors in a 28 nm SoC were investigated by Yang et al. [YLZ+20]. The electrons caused soft errors in the device's on-chip memory (OCM), D-Cache, register, and BRAM blocks.

X- and $\gamma$-rays are also known to cause errors or even device failure. Hence, inspecting semiconductor ICs with X-rays requires filtering and further precautions [Inf21]. Furthermore, $\gamma$-rays do not necessarily lead to the destruction of the device but can induce soft errors. This was shown by Fetahović et al., who observed bit-flips in Erasable Programmable Read-Only Memory (EPROM) and EEPROM caused by $\gamma$-radiation [FPV13]. Dolićanin investigated the long-term effects of $\gamma$-rays on four different NOR memory models [Dol12]. He showed that exposure to $\gamma$-rays for nine weeks led to a bit error percentage of 17% to 32%, which denotes the number of bit errors relative to the total memory size. A broader study on the susceptibility of flash memory to $\gamma$ radiation was conducted by Sharma et al. [SSF+16]. They targeted 250 nm NOR flash and 48 nm NAND flash technologies achieving the highest byte error rate for 12 keV X-ray irradiation. The authors performed experiments for different energies and exposure times, finding that data corruption is highest for long exposure time and that NAND flash memories were more susceptible to data corruption than the examined NOR flash memories.

Since these different types of radiation can cause soft errors, the question arises whether they would be able to reach the ASIC covered by the envelope layers. Table 6.1 shows the projected range for protons and $\alpha$-particles, the CSDA range for $\beta$-particles, and the HVL for X- and $\gamma$-rays. The absorber materials listed are copper, polyimide (Kapton), and silicon.

The CSDA range approximates the average path length that a charged particle travels as it slows down to rest [NIS22a]. However, the actual penetration depth differs from the CSDA range. This is expressed in the projected range; hence, the average penetration depth of a charged particle slowing down to rest.

For X- and $\gamma$-rays, the Half-Value Layer (HVL) represents the thickness of the absorber material, where 50% of the incident energy has been attenuated [NIS22b]. The HVL is inversely proportional to the attenuation coefficient $\mu$, defined through

$$I = I_0 \cdot \mathrm{e}^{-\mu t} = I_0 \cdot \mathrm{e}^{-\frac{\mu}{\rho}x}, \tag{6.1}$$

where $\mu$ is the attenuation coefficient, $t$ the thickness of the material, $I$ the radiation intensity, and $I_0$ the initial radiation intensity. For X-rays, the NIST database provides the mass attenuation coefficient $\mu/\rho$, with $\rho$ the absorber density and the mass thickness $x = \rho \cdot t$.

From the mass attenuation coefficient $\mu/\rho$, the HVL, i.e., where $I/I_0 = 0.5$, can be calculated as follows

$$t = -\frac{\ln(0.5)}{\frac{\mu}{\rho} \cdot \rho}. \tag{6.2}$$

In general, the distinction between $\gamma$- and X-rays is not always clearly defined. This is because classification within the electromagnetic spectrum varies, and both types of

**Table 6.1:** Overview of different radiation types, their projected range (protons and $\alpha$-particles) [NIS22a], CSDA range (electrons) [NIS22a] and HVL (X- and $\gamma$-rays) [NIS22b].

| **Radiation** | **Energy** | **Projected/CSDA Range, HVL** |
|---|---|---|
| Protons | 70 MeV | 7.1 mm (Copper)<br>2.2 cm (Silicon)<br>3.0 cm (Kapton) |
| | 200 MeV | 4.3 cm (Copper)<br>13.8 cm (Silicon)<br>19.2 cm (Kapton) |
| $\alpha$-particles | 2.0 MeV | 3.6 µm (Copper)<br>7.2 µm (Silicon)<br>8.0 µm (Kapton) |
| | 10 MeV | 26.5 µm (Copper)<br>69.1 µm (Silicon)<br>84.5 µm (Kapton) |
| $\beta$-particles | 0.1 MeV | 24.9 µm (Copper)<br>78.0 µm (Silicon)<br>0.1 mm (Kapton) |
| | 1.0 MeV | 0.7 mm (Copper)<br>0.2 cm (Silicon)<br>0.3 cm (Kapton) |
| | 20 MeV | 1.2 cm (Copper)<br>4.3 cm (Silicon)<br>6.9 cm (Kapton) |
| X- and $\gamma$-rays | 0.01 MeV | 3.6 µm (Copper)<br>87.5 µm (Silicon) |
| | 0.1 MeV | 1.6 mm (Copper)<br>1.6 cm (Silicon) |
| | 1.0 MeV | 1.3 cm (Copper)<br>4.7 cm (Silicon) |
| | 10 MeV | 2.5 cm (Copper)<br>12.1 cm (Silicon) |

radiation are either defined by their source of creation or by their photon energy. Hence, for this analysis, X- and $\gamma$-rays are grouped in the same category.

Table 6.1 shows that all types of radiation could, in general, reach the ASIC that is covered by the envelope with a thickness of 0.25 mm, including 8.5 µm copper, 150 µm polyimide (Kapton), and 100 µm adhesive tape 3M VHB9460. In general, the penetration depth within the absorber material also depends on the radiation energy. A further aspect that has to be considered is that the smaller the fabrication technology, the greater the radiation's effect on electronic components [CPG+19].

Faults induced through radiation are, hence, a potential threat to the enclosure system and, specifically, the ASIC. In the following, I describe how transient faults can impact the execution of cryptographic algorithms.

‘

## 6.2 Impact of Fault Injection on Cryptographic Algorithms

Faults can lead to changes in the control flow or manipulate calculation results that can leak CSPs. To demonstrate the effects that faults have on cryptographic algorithms, I herein present three experiments injecting transient faults into flash memory with data *and* instructions manipulations.

### 6.2.1 Previous Work and Contributions

Attacks on microcontrollers can be categorized into non-invasive, invasive, and semi-invasive attacks depending on the level of hardware alterations performed on the target device. Semi-invasive attacks require the decapsulation of the microcontroller as attack preparation but do not directly alter the hardware of the target device. A prominent example in the semi-invasive attack category is Laser Fault Injection (LFI).

Traditionally, publications in the field of LFI primarily focus on faulting processor registers or Static Random-Access Memory (SRAM) to manipulate data and, thus, break algorithm implementations [CLMFT14, SBHS16, RSDT13].

So far, attacks on non-volatile memory have been vastly omitted in fault injection research, with a few exceptions. UV light has been used for many years to disable security fuses in EPROM [Sko12]. Furthermore, UV light attacks enable reverse-engineering of address scrambling in smart card chips [FLM10]. Another exception is the work of Obermaier and Tatschner [OT17], who permanently manipulated data in flash memory through UV-C light and mentioned that they observed transient faults on the same device injected via a pulse laser. One of the first transient fault attacks against flash memory read-out was performed by Skorobogatov, targeting the microcontroller's backside [Sko10]. Skorobogatov considered optical fault injection via the front side as impractical for targets with a feature size $\leq 0.35\,\mu\text{m}$. However, in the following I, will show that even a 180 nm feature-size flash memory is susceptible to *front-side* LFI.

In 2018, Colombier et al. performed LFI on the flash memory of a Cortex-M3 processor [CMD+18]. To analyze the targeted instructions, Colombier et al. surrounded them with multiple NOP operations. A NOP instruction is usually implemented through an ADD operation that can be altered through a laser pulse leading to a fault, which leaves the target instruction unchanged. The use of NOPs in close vicinity to the target instruction can, hence, falsify the attack analysis.

Colombier et al. mentioned that the characterization of laser parameters might take months [CMD+18]. They also stated that pipeline behaviors and timing inconsistencies hampered the fault analysis. In the following, I will discuss a calibration method based on

Direct Memory Access (DMA) that resolves these difficulties. Furthermore, Colombier et al. targeted instruction fetches while mentioning the difficulty of data faulting since the data resides in SRAM. The fault attacks in this section show that data faulting is indeed possible, which I demonstrate through a differential fault attack on AES targeting the S-box access and the *indirect* faulting of registers by altering instructions.

Even though the Cortex-M3 operates at a frequency in the two-figure to the three-figure range, Colombier et al. reduced the Cortex-M3 frequency to 7.4 MHz. Through this frequency reduction, possible disturbances induced by the laser can not be assessed for the practical use case. In the LFI experiments that I present in this section, the targeted Cortex-M0 processor was run at its maximum frequency of 48 MHz while enabling the cache/prefetch buffer. Through this, I analyze LFI attacks on flash memory under realistic conditions. Apart from the aspects mentioned above, Colombier et al. also significantly reduced the feature size such that as little chip area as possible was affected by the laser spot. To demonstrate that LFI attacks are feasible irrespective of the DuT's feature size, the LFI experiments described in this section were conducted with coarse laser settings and a rectangular spot size with dimensions of 40-60 µm, hence, covering multiple flash cells.

Another group that targeted the flash memory of a microcontroller was Kumar et al., who performed an LFI attack on the ATmega128p [KBB$^{+}$19]. Like Colombier et al., they surrounded the target instruction with multiple NOPs and reduced the spot size to the single-digit micrometer range.

They also ran the ATmega at a reduced clock frequency of 13 MHz instead of the maximum of 20 MHz. Kumar et al. stated that manual reverse engineering of executed instructions was "extremely time-consuming“. Furthermore, they suspected affecting read-out circuitry shared between several bits. As I will show in the following, the effect of the LFI attacks in this section can be directly attributed to the flash cells instead of the read-out logic.

In the following section, I show how fault injection threatens the execution of cryptographic algorithms. This is demonstrated in a front-side attack on an ARM Cortex-M0 targeting instruction *and* data fetches from flash memory via LFI and transient faults. The repeatable and reliable results were obtained by calibrating the laser and adjusting its position and spot size. I show that the DuT was attacked under realistic conditions, e.g., at the maximum clock frequency, with Phase-Locked Loop (PLL) and caching/prefetch enabled and compiler optimizations. Furthermore, I explain the attack background and discuss why the 180 nm flash memory is still reliably faulted even with extreme mismatches between structure and laser spot size. I presented the attacks described in this section at the 26th IEEE International Symposium on On-Line Testing and Robust System Design. The results are published in the corresponding IEEE Proceedings [GO20]. Supplementary material is available at `https://github.com/Fraunhofer-AISEC/laserflash`.

### 6.2.2 Background

Transient fault injection into flash memory is a topic that has only been sparsely covered so far. To understand the mechanism behind transient faults in more detail, I address the attack background in the following. However, it should be noted that even though the proposed model explains the observed behavior, further influences and different fault mechanisms can not be ruled out.

#### Flash Memory Cell

Microcontrollers equipped with flash memory usually contain NOR memory, which has a relatively short read-access time and is more reliable than NAND flash. A single cell of NOR flash memory comprises a floating gate transistor, which is depicted in Figure 6.1. It can basically be seen as a metal–oxide–semiconductor field-effect transistor (MOSFET) with an additional floating gate. The cell can either store the value **1** or **0**, depending on the number of charge carriers on the floating gate (FG). Without additional charge carriers and an active control gate (G), the current flow from the drain (D) to source (S) translates to a binary **1**. Whereas, with a charged floating gate, no current between source and drain will be measured, resulting in a binary **0**. When data or instructions in flash memory are read out, the read word is addressed through the corresponding bit and word lines. The bit lines (BL) act as source and drain electrodes, while the word lines (WL) are attached to the control gate.

According to the observed behavior of the target chip, the injected faults into NOR flash are expected to be unidirectional, i.e., bits can only be flipped in one direction, e.g., they are flipped to **1**. This can only be assumed if the faults are injected into the actual flash memory cells. If, however, the faults are injected into the read-out circuitry as suggested by Kumar et al. [KBB$^+$19], an asymmetric channel model, i.e., unidirectional errors, can no longer be assumed.

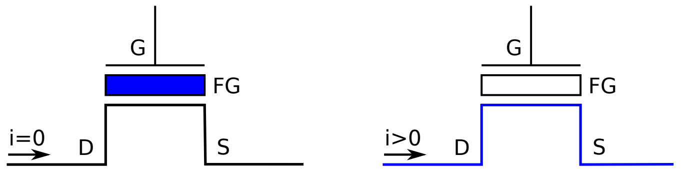


**Figure 6.1:** Binary **0** (charged cell) on the left and reading of a NOR flash cell, with a binary **1** (uncharged cell) on the right.

#### Transient Faults

There are different ways to manufacture NOR flash. The flash cell assembly of the target device is depicted in Figure 6.2.

In a word line, several words can be stored. The $i$-th bits of all words are stored close to each other. For example, in Figure 6.2, the last bits of all words are stored in the

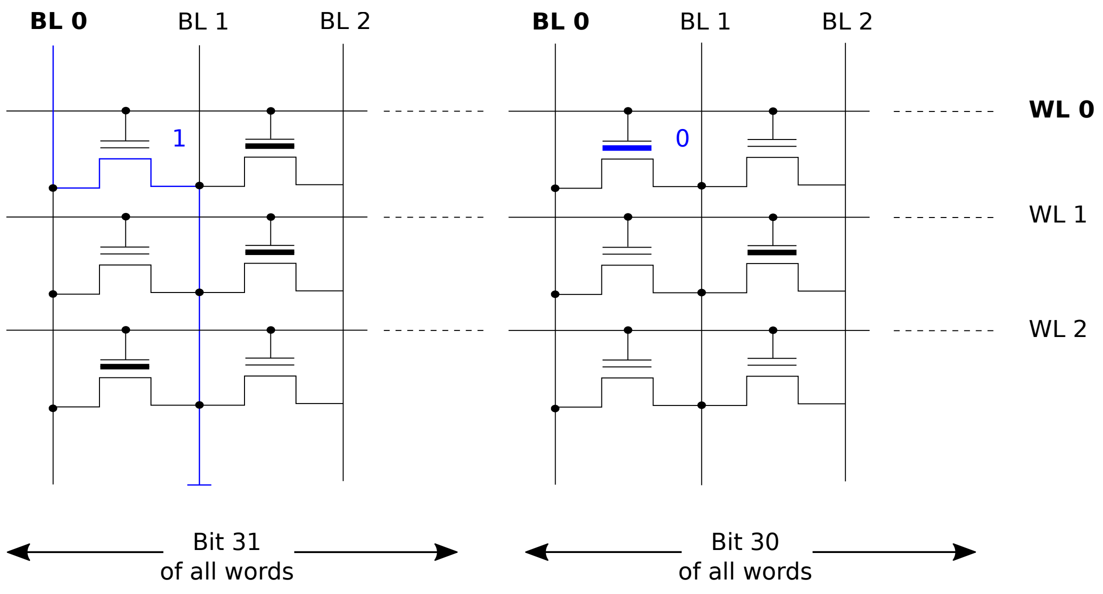


**Figure 6.2:** Reading out a data word in NOR flash. Addressing the read out of a binary **1** (31st bit) and binary **0** (30th bit) via (BL0, WL0). All bit lines BL0 are read out in parallel.

leftmost part of the flash memory. The next-to-last bits of all words can be found going from left to right, followed by the next group of bits and so forth.

All leftmost bit lines (BL0) are read in parallel to obtain the leftmost word in the first word line. The neighboring bit line (BL1) is pulled to GND to read a bit of BL0 and hence, acts as source electrode potential for the respective floating gate transistor. This way, space can be saved; however, not all words can be read out simultaneously.

When shooting laser light onto flash memory, the change can be temporary, permanent, or there can be no impact at all that is detectable. Which of these three options actually occurs depends on the wavelength and intensity of the laser beam. I observed *transient faults* in the conducted attacks. These transient faults could be timed to a specific word read-out.

During the execution of a program, data or instructions are fetched. In the following, I assume that the relevant bits are stored in the uppermost word line. Figure 6.3 shows the read-out of the memory cell associated with BL0 and WL0. With no current between source and drain, the 30th bit associated with BL0 and WL0 corresponds to a binary **0** during regular read-out. However, suppose the flash cells are targeted with a laser beam of sufficient energy (highlighted green box in Figure 6.3) during the read-out of a word. In that case, the memory cell's value is temporarily changed since a laser beam with sufficient energy creates electron-hole pairs in the targeted memory cells. This induces a current from the source to drain, flowing through the gate transistor depicted by the highlighted cells at the bottom of Figure 6.3. Hence, the original **0** will *temporarily* change to a binary **1**, implying that the bit line is pulled to GND.

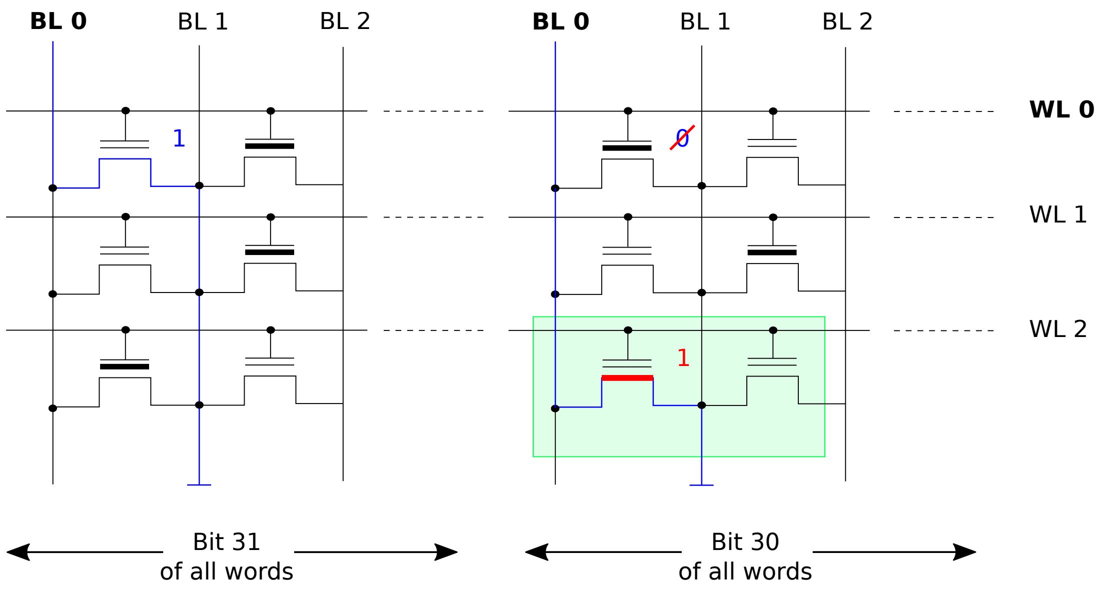


**Figure 6.3:** Targeting BL0 through laser fault injection. The laser beam (highlighted green box) induces a *temporal* change in the targeted bit line, faulting the upper memory cell to **1** by pulling the bit line to GND.

The advantage of this setup is that only the corresponding bit line has to be hit to change specific bits of a word. The attack affects all word lines and hence, does not have to be spatially finetuned to target a specific word line. This allows for successful *temporal* manipulation of flash memory cells with coarse laser settings and simplifies the laser adjustment, which makes the attack quite powerful.

In practice, when shedding light on flash memory, it is difficult to determine how much energy is absorbed by the material. Furthermore, in most cases, the laser will not hit the bare flash cells since, after chip decapsulation, small structures will remain on top of the cells. Hence, in practice, more energy will be necessary to trigger the formation of electron-hole pairs in the flash cells.

## 6.2.3 Experimental Setup

The experimental setup for the performed attacks and the details for the attack execution are described in the following.

### Overview of Performed Experiments

Transient fault injection, which was discussed above, enables the execution of several experiments:

**Calibration via bus master data transaction:**
For efficient experimentation, the laser position, spot size, and timing are adjusted to obtain the desired number and position of faulted bits. I will refer to this process as

*calibration.* The laser calibration can be performed on any bus master of the microcontroller that has flash memory access. The large advantage of this calibration procedure is that the CPU is not involved, which enables accurate timing. By reading out the flash memory and checking which bits have been affected by the current laser position, the reproducibility of the results can be ensured. Furthermore, the calibration method provides insight into the duration of injected faults.

**Manipulation of data fetch:**
I demonstrate the possibilities of transient fault injection through a data fetch manipulation, for which I implemented the AES diagonal fault attack (fault model M0) described by Saha et al. [SMRC09]. The manipulation occurs with enabled prefetching and at the maximum specified CPU clock frequency. The attack is, therefore, quite powerful, as it works reliably irrespective of the CPU behavior.

**Manipulation of instructions:**
Transient fault injection can also be applied to instructions. For the demonstration of the above-described principle, *mov* (register and immediate) and *branch* instructions are manipulated via transient faults. Moreover, another instruction manipulation is shown by successfully skipping the last AES round. Both attacks were reliably reproduced.

### Overview of Fault Injection Setup

Figure 6.4 shows the experimental setup for the injection of *transient* faults. The laser source is a QuikLaze-50 ST2 pulse laser equipped with a Mitutoyo microscope. The DuT (left side) and the attack board (right side) are both STM32F051R8T6 Discovery Boards with an ARM Cortex-M0. The internal chip of the DuT was exposed through chemical etching. The attack board resets the DuT and provides the external clock. A Raspberry Pi orchestrates the attack and collects the results via a UART interface. It also controls the overall timing of the laser pulse injection, while the attack board takes care of the low-level timing and triggering.

As a preparation step, the upper layers of the microcontroller were removed via chemical etching and exposure to high-power laser light [OT17]. The flash memory layout, as depicted in Figure 6.5, was reverse engineered by Obermaier and Tatschner. They successively covered it with different masks, analyzing the effects of UV-light exposure on the microcontroller [OT17]. The total flash memory size amounts to 64 KiB, 512 word lines, and 1024 bit lines.

Apart from the firmware flash region, flash memory is also reserved for settings and calibration, which is likely implemented via additional word lines. This leads to an extra 3 KiB for system memory and additional word lines for option bytes. The overall number of word lines amounts to 537. The reverse engineering also revealed an estimate of the width of a bit column. In fact, 32 bits correspond to approximately 23 $\mu$m. This is a typical flash memory layout, as other reverse engineering efforts have shown [CSW17]. The feasibility of the attack, hence, is not limited to a specific microcontroller.

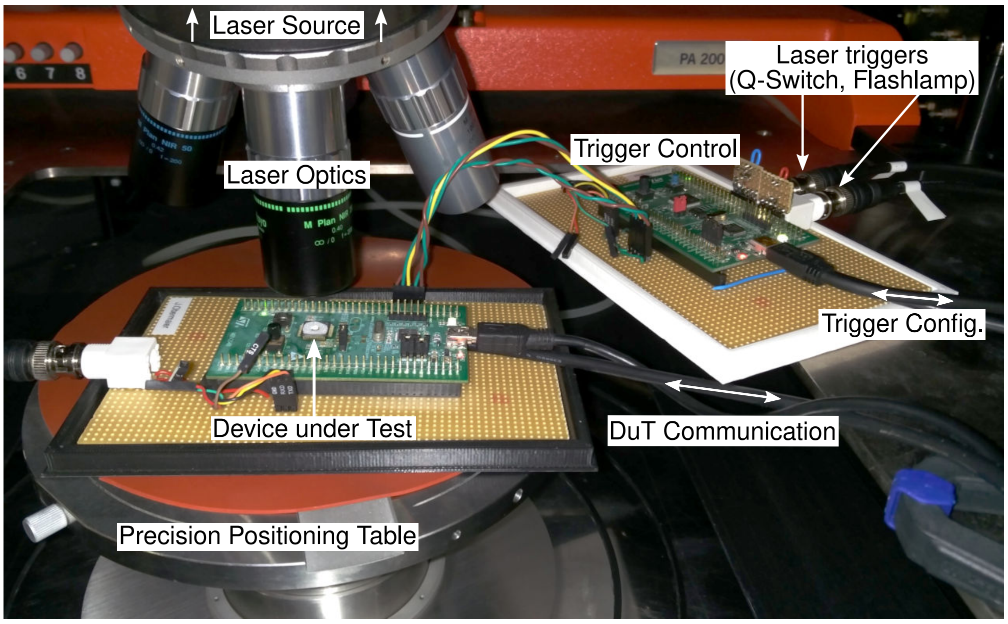


**Figure 6.4:** The experimental setup for laser fault injection, with the DuT on the left, the Trigger Control board on the right, and the QuikLaze-50 ST2 equipped with a Mitutoyo microscope.

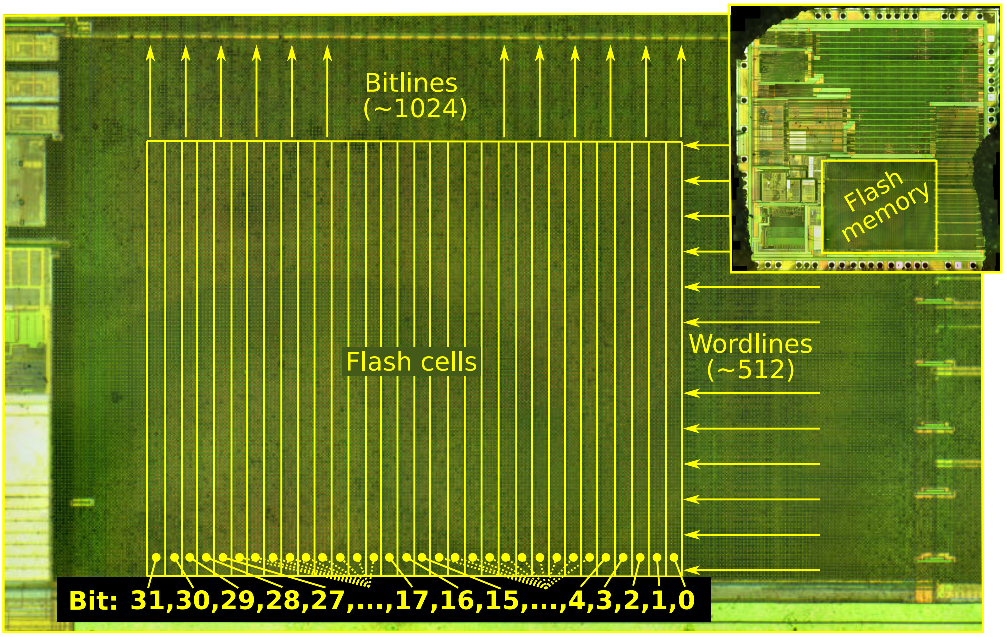


**Figure 6.5:** Flash memory layout of the DuT with 1024 bit lines and 512 word lines.

Furthermore, fetching a word implies obtaining two stored 16-bit instructions in the ARM Thumb instruction set. Hence, a single laser pulse may affect only one of the instructions.

### Attack Execution

The QuikLaze-50 ST2 laser has three possible wavelengths: infrared (IR) (1064 nm), green (532 nm), and ultraviolet (UV) (355 nm). *Transient* faults were caused by the green laser in a *front* side attack with a pulse energy of approximately 0.4 mJ. Tests with other samples of the same DuT gave insight into suitable energy settings. A pulse energy of approximately 0.4 mJ seemed to be the lower limit for the successful injection of transient faults. Lower pulse energies led to no faults at all, while a significant increase in the pulse energy destroyed the microcontroller's memory cells. The pulse duration was set to 10 ns. These parameter settings allowed for thousands of experiments without permanent damage to the flash memory region.

The laser's rectangular shutter was adjusted by setting its width to 50 % and the height to a value between 30 % and 40 %, corresponding to a laser spot size of 58 µm × 42 µm, which was measured by removing flash material from a second microcontroller with a high power laser pulse. This means that the shutter setting affected 2-3 bits of an 32-bit word, where one bit of a word corresponds to approximately 23 µm. During the measurement of the spot size, the laser pulse distribution was found to be irregular, as the structure of the remaining material was also irregular. Nevertheless, even with these very coarse laser settings, all experiment outcomes were reliably and accurately reproduced. This is a significant advantage over other setups with, for instance, small spot sizes [KBB+19, CMD+18], as an approximate laser placement is already sufficient to fault the bit line.

Previous attack setups were run at a reduced clock frequency, with prefetch buffer and optimization disabled [KBB+19, CMD+18]. All of the following attacks were conducted with enabled prefetch buffer and compiler optimizations `-O2` to provide realistic attack conditions. What is more, the DuT was run at the maximum clock frequency of 48 MHz generated by the DuT's PLL. Furthermore, the code can either be partially run from SRAM or fully run from flash memory.

## 6.2.4 Calibration Method

One of the main difficulties of LFI experiments is the adjustment of laser settings. Colombier et al. mentioned that this process could even take months [CMD+18]. In the following, I will show a simple and fast method to calibrate the laser setup and adjust the laser position, timing, and spot size, to determine the number and position of affected flash bits. Apart from this, the calibration method allows determining the *temporal* duration of the injected faults.

In previous LFI setups [KBB+19, CMD+18], tedious reverse engineering of the injected faults was necessary, for instance, by analyzing all possible register changes. However, even then, determining the exact location and number of flipped bits is difficult. The number of bit-flips, in general, is unknown since it is hard to tell whether bits that are already **1** were affected by the laser pulse. Furthermore, since certain instructions produce the same result [SO19], for instance, $r1 = r2 + r3$ and $r1 = r3 + r2$, reverse engineering will not necessarily determine which instruction has been faulted.

To overcome these drawbacks, I propose a simple and effective calibration method that I describe in the following. This method is based on the fact that access to flash memory does not exclusively occur via the CPU, but that other bus masters, like DMA, can also access the flash memory [STM22b]. By copying data to SRAM via DMA while adjusting the laser settings, each bit can be observed directly to check whether it was affected by the laser pulse. This drastically reduces the number of required test patterns since a zero-only pattern is sufficient to check the sensitivity of every bit. Another advantage of the calibration is that it can be performed on non-executable parts of the flash memory, such as interrupt tables. Furthermore, the calibration via the bus master allows for a precise evaluation of temporal and spatial effects of the laser pulse since, in contrast to accesses via the CPU, DMA transfers have fixed timing.

In total, the proposed calibration method has many advantages over the usual reverse-engineering approach:

- *Any* desired pattern and *each* bit is tested directly.
- The affected instructions are determined from the calibration data.
- One test pattern in flash memory is sufficient, i.e., multiple tests are no longer necessary.
- This method works on all parts of the flash memory that are accessible to the bus master.
- The timing effects of the laser pulse are well visible due to the high timing reliability of this low-level method.

It should be noted that this method is only used for calibration pre-attack, while during the experiment, the flash memory is accessed directly by the CPU. To exclude any influence by the processor, the `WFI` instruction of the Cortex-M0 stops the CPU during calibration.

The basic idea behind the calibration is simple and easily implementable: Repeatedly copy the 32-bit word `0x00000000` from flash memory to SRAM. The content copied to SRAM shows the affected bits in flash memory. The reason for this is attributed to the flash memory layout (Figure 6.5). A word in flash memory spans its entire width; hence, if one bit column is hit, all read-out words will be affected. The offset of the affected address is equivalent to the temporal component of the laser trigger. This method allows easily calibrating the laser spot, position, and power to a specific fault pattern of single or multiple neighboring bit-flips while analyzing the temporal component.

The analysis of the temporal component not only helps set the trigger but also provides valuable insight into possible *smearing* effects in the case of *transient* faults. Smearing, in this case, refers to multiple DMA transactions that are affected by one laser pulse, as depicted by the red curve in Figure 6.6, since it can, in general, not be ruled out that the recombination of charge carriers could take several DMA transactions. Only an analysis of the temporal component can exclude such effects that could falsify the attack results.

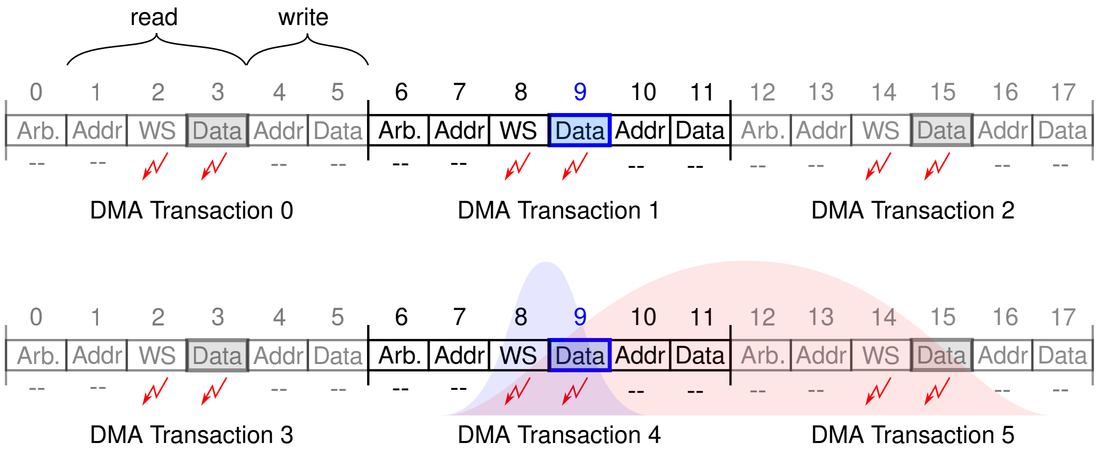


**Figure 6.6:** Calibration of the QuikLaze-50 ST2 via DMA transactions from flash memory to SRAM. The injected single- and multi-bit faults are temporarily limited to two clock cycles (WS and Data). The hypothetical duration of charge carrier recombination is depicted by the blue and red curve.

The minority carrier lifetimes in direct semiconductors, such as silicon, can be in the µs range [Sch97]. Since the laser pulse and clock period are in the ns range, the recombination of electron-hole pairs could take several clock cycles, which could affect multiple transactions (data or instruction) from flash memory. This means that it is, in general, not clear whether adding additional energy to semiconductor structures will lead to smearing effects.

An analysis of the temporal component through the calibration method provides insight into how many data transactions are affected by the laser pulse. One DMA transaction on the ARM Cortex-M0 — copying one word from flash memory to SRAM — can be modeled by six clock cycles, as depicted in Figure 6.6. The DMA arbiter selects the DMA channel associated with the corresponding memory-to-memory transfer [STM22b]. This is followed by a read operation accessing the `Data` at `Addr` in flash memory, including the latency (wait state) required to maintain the control signals of the flash memory for the specified clock frequency. The `Data` are then written to the corresponding `Addr` in SRAM. The flash access occurs in two out of the six clock cycles (Wait State and Data Access), which are vulnerable to LFI. The copied data in SRAM revealed that a fault is limited to a single DMA transaction, even at a clock frequency of 48 MHz. This is depicted exemplarily by the blue curve in Figure 6.6. The analysis demonstrated that even a single-bit is flipped precisely in one transaction. This shows that charge carrier recombination indeed takes place within a single transaction and that even if multiple flash accesses occur within a certain time frame, only a single data/instruction transaction will be affected. Hence, the proposed calibration method not only easily determines the spatial parameters of the experimental setup but also provides a deeper insight into the temporal behavior of the target device.

The calibration enables a reliable and repeatable injection of transient faults into flash memory, as I will show in three experiments: A data fetch manipulation with a differential fault attack on the AES and two instruction fetch manipulations that change the execution of the AES and modify specific instructions.

### 6.2.5 Data Fetch Manipulation

The first attack demonstrating the effect of transient faults is a data fetch manipulation on the Advanced Encryption Standard (AES). As mentioned before, the code was adapted to run both in flash memory and partially in SRAM. To provide realistic conditions, the TinyAES-128 was adjusted to run in the DuT's SRAM. However, as a larger constant, the S-box is stored in flash memory. The S-box lookup (*SubBytes*) in the 8th round is the target of the differential fault attack [SMRC09], where a single injected fault propagates through the AES-state. A set of equations describes this fault propagation relating the faulty ciphertexts with the 10th round key. Since three equations contain four key bytes, the underconstrained set of equations requires trying all possible key bytes. A few faulty ciphertexts, and the correct ciphertext, are sufficient to determine the secret AES key.

To provide even more realistic conditions, at system startup, the DuT runs a set of test vectors to ensure the correct behavior of the AES implementation. This serves as a protection against *permanent* changes in flash memory. The experiment results showed that the injection of *transient* faults was unnoticed by the self-check routine. In the experiment, the DuT obtained sample data for encryption with the TinyAES-128 consisting of the key and plaintext from the first test vector of NIST SP 800-38A. With the calibration method, the execution of the attack took only a few minutes, injecting faults with one to four bit-flips into a single S-box lookup. From the faulty ciphertexts, I calculated the secret AES key and, hence, broke the encryption. Further information on the attack results is published at `https://github.com/Fraunhofer-AISEC/laserflash`.

### 6.2.6 Instruction Fetch Manipulation

The calibration method allows for not only the manipulation of data fetches and transactions but also the reliable and repeated manipulation of *instruction* fetches, as I will show in the following via two experiments: An instruction manipulation of the AES and the modification of specific instructions through injected *transient* faults.

#### AES Round Skip

The target of the first experiment is again the TinyAES-128 implementation of Subsection 6.2.5, however, with two distinctions. First, the AES implementation in this experiment runs entirely from the DuT's flash memory, and second the target of the attack is not the S-box fetch but the manipulation of the AES loop counter. For this experiment, the previous AES test vectors were used. However, setting the trigger timing to a specific word-fetch is difficult. The precise moment of an instruction fetch can only be reduced to a certain interval since the prefetch buffer causes an offset, and bus contention may introduce jitter. Thus, to affect the intended instruction, the moment

```
Program Counter     Opcode     Instruction        Registers

ROUNDSTART: .....     b                   r7       0x20000050
    0x08000938      469a      mov   r10, r3       r10: 0x10
    0x0800093a      9b00      ldr   r3, [sp]      r3: 0x20001FB0
    0x0800093c      44d1      add   r9, r10       r9: 0x20001FA0
    0x0800093e      454b      cmp   r3, r9         0x40001FF0
    0x08000940      d300      blo   AESFINISHED
    0x08000942      e777      b     ROUNDSTART
AESFINISHED: .....
```

**Figure 6.7:** Assembly code of the loop counter check for the second-last AES round, faulting the loop's upper limit `r3` by replacing it with the pointer `r7`. The faulted instructions are depicted in blue color.

of fault injection is gradually shifted over time. Even with these timing limitations, the experiment still took only a few minutes. The experiments were performed on the ARM Thumb instruction set, where one word-fetch contains two 16-bit instructions.

Through *temporary* instruction faulting, the code execution terminates in the 9th AES round. The assembly code of the loop counter check is shown in Figure 6.7. The pointer `r9` represents the loop counter, with the loop's upper limit `r3` and the loop counter increment `r10`. The pointer `r7` is, in general, unrelated to the increment of the loop counter. Through the calibration, the injected fault pattern is set to `0x00000020` and injected into the `mov r10, r3` instruction. This replaces the loop's upper limit `r3` with the (large) pointer `r7`. The fault propagates through the code, altering the loop counter `r9`, which leads to premature termination of the AES execution. Knowing the correct ciphertext, I derived the AES main key from the faulty ciphertext. This instruction manipulation demonstrates that *transient* faults can easily break the AES implementation even with self-check functions enabled.

### Manipulation of Specific Instructions

Instructions can be faulted not only by manipulating immediate values and registers but by modifying the instruction itself. I will show both instruction manipulations through transient fault injection into the assembly code in Listing 6.1, which is executed in an infinite loop.

The corresponding ARM Thumb instructions are added on the right. Since with each word-fetch, two 16-bit instructions are read, only one of the instructions can be faulted with a single laser pulse. Hence, only half of the operations are listed. The target instructions are move-immediate-to-register (`mov`), addition (`add`), compare (`cmp`), and branch-if-equal (`beq`) instructions, which are the basic building blocks of algorithms. The registers are initialized by successive powers of two, which are added up and compared to the expected sum. Altering the register initialization or summation leads to a failed compare, which leaves the loop and dumps the register content.

```
1:
    movs r0, #0         00100 000 00000000
    movs r1, #1
    movs r2, #2         00100 010 00000010
    movs r3, #4
    movs r4, #8         00100 100 00001000
    movs r5, #16
    movs r6, #64        00100 110 01000000
    movs r7, #128
    adds r0, r0         0001100 000 000 000
    adds r0, r1
    adds r0, r2         0001100 010 000 000
    adds r0, r3
    adds r0, r4         0001100 100 000 000
    adds r0, r5
    adds r0, r6         0001100 110 000 000
    adds r0, r7
    cmp  r0, 0xDF       00101 000 00001111
    beq  1b
```

**Listing 6.1:** Targeted assembly code with the corresponding 16-bit ARM Thumb instructions.

The goal of the first experiment is to fault immediate values and registers. For this purpose, the fault pattern is calibrated to `0x00000020`. Targeting the `mov` instruction alters the immediate value copied into the specified register. The

`movs r2, #2` (`00100 010 00000010`)

instruction is changed to

`movs r2, #34` (`00100 010 00100010`).

Another example is a change of `add r0, r0, r1` to `add r0, r4, r1`. I observed similar behavior for the `cmp` instruction. All manipulations via the calibrated transient faults lead to a termination of the loop. By changing the shutter width and hence, the spot size, I observed single and multi-bit faults.

For the second experiment, the fault pattern was set to `0x00000100` through the calibration method. The target of the experiment was to manipulate the branch-if-equal `beq` instruction itself. The (temporary) fault pattern modified the

`beq 1b` (`1101 0000 11111101`)

to

`bne 1b` (`1101 0001 11111101`),

executing the branch only if the compared values are *not* equal, which led to a termination of the loop. Further information and details of the attack result are available at `https://github.com/Fraunhofer-AISEC/laserflash`.

### 6.2.7 Attack Limitations and Summary

In this section, I demonstrated the reliable and repeated manipulation of data *and* instructions via transient faults and a fast calibration method.

I discussed data *and* instruction manipulations targeting the AES execution and extracting its secret key. Furthermore, I demonstrated instruction manipulations by targeting the instruction itself and faulting of registers and immediate values. All attacks were performed under realistic conditions running at the maximum clock frequency and enabling the prefetch buffer and compiler optimizations. I showed that transient faults can circumvent the AES self-check routines and that even coarse laser settings lead to reliable and repeatable results. This demonstrates the high future relevance of transient fault injection into flash memory, even for decreasing structure sizes.

Various countermeasures against fault injection are discussed in Section 6.5. Apart from these countermeasures, some other mechanisms and implementations pose an obstacle to the attack execution. For example, obfuscation and scrambling complicate the calibration of the laser, which impedes targeting specific bits in flash memory. However, obfuscation alone will not suffice to fully mitigate the risk since reverse engineering efforts can reveal the particular obfuscation. Besides obfuscation and scrambling, memory encryption poses another obstacle to the attack execution. Although, depending on the chosen cipher, certain algorithms might leak information about the targeted bit positions. Furthermore, microcontrollers might apply wear leveling to balance the flash memory usage. This hampers the identification of the correct laser position, which affects the calibration procedure. If microcontrollers implement all or several of these techniques, conducting laser fault injection attacks might be severely impeded. However, many low-cost devices do not implement these mechanisms and are hence, expected to be susceptible to transient fault injection.

This section demonstrated the reliable and repeated injection of transient faults into a microcontroller's flash memory under realistic conditions. In a front side attack, transient instruction and data faults were injected with minimal preparation time due to efficient laser calibration. Even though, the calibration method significantly reduced the time for experiment preparation, the search for fault candidates, i.e., weaknesses in the target firmware, required a cumbersome manual analysis of assembly instructions. To speed up the process of finding possible fault candidates, automated tool-based testing can be applied. However, the available fault injection tools only supported specific fault types and architectures. Furthermore, many of these tools were not publicly available. To overcome these drawbacks, in the following, I introduce ARCHIE, an architecture-independent QEMU-based framework for the evaluation of faults, which I later on use for an automated analysis of the EKMS implementation (Section 6.4).

## 6.3 A QEMU-Based Framework for Fault Emulation

As I showed in the previous section, injected faults can disturb the execution of cryptographic algorithms and even leak CSPs. However, finding possible weaknesses in the code, which I will refer to as *fault candidates*, can be quite time-consuming. Even if no black box testing occurs, hence, when the source code is available, the resulting binary is affected by the compiler version and settings, leading to varying binary files and different outcomes of the source code analysis. In general, the cumbersome manual examination of the binary and assembly instructions is necessary to determine potential fault candidates. The propagation of errors is difficult to predict, and even after a time-consuming manual analysis of the assembly code, the most critical fault candidates might not even be found. Since the calibration method, described in Section 6.2, greatly facilitated the adjustment of the laser settings; indeed, most of the time during the LFI experiments was spent on finding suitable fault targets.

However, the automated simulation of faults through a software tool could greatly facilitate the process of finding suitable fault candidates and hence, conducting LFI experiments. In this section, I introduce ARCHitecture-Independent Evaluation (ARCHIE), an ARCHitecture-Independent QEMU-based framework for fault Evaluation that provides:

- The automatic fault injection into the running binary,
- The possibility of parallelized execution,
- The simulation results in text file format,
- Open source availability,
- And most importantly, an easy implementation on different boards and architectures.

I presented ARCHIE and the results discussed in this section at the 2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC) together with my co-author Florian Hauschild. This work was published in the IEEE conference proceedings [HGA+21].

### 6.3.1 Previous Work

The amount of tools for fault simulation is vast. However, when I searched this large zoo of tools and checked them against the above-listed requirements, I was left disappointed. None of them fulfilled all requirements, let alone the ability to support different processor architectures without considerable modifications to the tool itself. In the following, I will discuss the results of this search and give an overview of different fault injection categories, considering physical and virtual fault injection. Furthermore, since ARCHIE is based on QEMU — an open source machine emulator and virtualizer — I also discuss QEMU-based fault simulation tools.

#### Fault Injection Categories

Due to a large number of device architectures, fault types, and fault analysis objectives, many fault injection and simulation tools have been published. For safety applications, a high-level approach alone is not sufficient to determine whether the system remains in a safe state. Safety-critical systems are influenced by arbitrary faults caused by environmental changes or natural radiation phenomena. However, in the field of security, faults are injected on purpose through an attacker, either through physical access or manipulation of software.

Depending on the abstraction layer, there are different possibilities of injecting faults into the target device. These faults can either be injected physically or implemented in hardware or software.

I already showed an example of physical fault injection in Section 6.2. Laser fault injection requires much preparation, for instance, decapsulation of the target device and the assembly and configuration of the attack setup. The system's susceptibility to faults is often determined by injecting faults in predefined patterns and analyzing the observed outcome. However, this approach is quite time-consuming and can lead to unintended faults — due to thermal drifts, jitter, or vibrations — or latch-ups that can even destroy the target device [HHH08]. Furthermore, specific faults may be impossible to provoke due to the limitations of setup parameters, such as laser focus or wavelength [SZK+18]. In general, the investigation method and sample preparation limit the types of injected faults. Considering all of these aspects requires a lot of manual adjustments, which slows down the experimentation. Counteracting this, for instance, through parallelization of tests, requires additional hardware and preparation and hence, does not scale well.

Apart from physical fault injection, faults can also be injected into hardware at the logic level. Hardware-based methods include, for instance, fault injection on the pin-level [AAA+90, MRMS01]. In this case, probes or other hardware modules set the pins of the DuT to faulty values. Another approach injects faults through on-chip debugging techniques [FSK98, AVFK03, YAL+03, CMS98, FSMA99], using already built-in logic in microcontrollers, and hence, not requiring additional hardware modules. Another hardware-based approach at a higher abstraction layer is fault injection at Register Transfer Level (RTL). One exemplary implementation is ETISS-ML [MGDW+18], an instruction set simulator for soft errors. Moreover, Civera et al. speed up the fault injection into VLSI circuits via FPGA-based emulation [CMR+01]. A security-based approach for fixing design bugs and testing countermeasures on the circuit level was proposed by Nyberg [Nyb18]. He focused on maximizing the configurability and performance of FPGA-based fault emulation, modeling fault attacks in gate level netlists.

Model-based fault injection is implemented at the hardware level via VHDL models [JAR+94, STB97, GBGG03, GSBC+08], SCADE models [VBRE07], or Matlab / Simulink models [SVET10, FMMJ21]. These approaches aim at improving system models in a safety context.

Apart from hardware-based approaches, there are also numerous fault injection tools implemented in software. Many implementations focus on OS validation [Hil02, KKA95, TI95, SVS+88, WTSS13], or the evaluation of OS applications [MRL02, MR00, LVY12, WTLP14]. Some tools in this category are based on virtualization [SB02, PSC07]. Software fault injection into embedded systems can be directly implemented on the DuT [BBC+14, ALMT17] or virtualized. In recent years, virtualized approaches have become more frequent, such as ARMORY [HSP21], a fault injection tool for ARM-M binaries based on an emulator for ARMv6-M and ARMv7-M architectures.

In recent years, many tools based on virtualizers and emulators for fault injection into embedded devices have been developed. Many of these tools are based on QEMU, which I discuss in the following.

### Fault Simulation Based on QEMU

Many more recent fault analysis tools use QEMU, which is an open-source emulator and virtualizer that emulates the target device through dynamic binary translation [Bel05]. In contrast to physical fault injection, the advantage of simulating or emulating faults is that many faults can be tested within a certain time frame without damaging hardware or spending time preparing or adjusting an experimental setup.

One of the oldest fault simulation tools based on QEMU is called QEFI [CG12]. QEFI configures and triggers the fault injection into ARM CPU registers and RAM via QEMU's GNU Debugger (GDB) interface. The authors modified the source code to implement the fault injection points. They attribute this drawback to a lack of hook mechanisms in QEMU. Since QEFI comes with minimal log management, implementing logging mechanisms is left to the user. QEFI is tailored to ARM processors.

Another QEMU-based tool that does not inject faults via GDB is XEMU [BBK$^+$12]. Prior to the fault injection through mutation testing, XEMU analyzes the control flow graph of the disassembled target device. The authors enhanced QEMU's performance by combining the execution with and without faults. For this modification, they extended QEMU's lifetime by saving a backup of the CPU and memory state, followed by a reset of QEMU and subsequent fault injection. XEMU was tested on car motor management software specifically for the ARM instruction set, saving the simulation results in a metrics report.

Adelt et al. presented another QEMU-based framework tailored to the RISC-V architecture. The authors injected transient and permanent single n-bit-flips targeting CPU registers and instructions. They also limited the number of possible fault variants to reduce the simulation time.

Geissler et al. published a QEMU-based approach for soft errors targeting the x86 architecture [GKS14]. The fault location, type, and clock cycle define a fault that is injected into the CPU registers. The authors tested their implementation for the RTEMS OS.

In 2013, Li et al. introduced BitVaSim [LXW13, XX12], a QEMU-based tool for build-in tests of PowerPC and ARM processors. However, according to the authors, to support different fault modules, QEMU was "heavily" modified, impairing the maintainability of the tool.

In the same year, Ferraretto and Pravadelli presented an approach combining RTL fault modeling with fault injection in QEMU by modifying the Dynamic Binary Translation (DBT) [DGFFP13]. Their approach is tailored to an ARM instruction set. To improve the emulator performance, Ferraretto and Pravadelli modified the DBT fetching mechanism and the Translation Block (TB) caching [FP15, FP16]. They injected transient and permanent faults into CPU registers and tailored their implementation to

ARM and x86 processors. In their implementation, Ferraretto and Pravadelli focused on instruction faults without considering faults in flash memory or RAM.

Traditionally, many fault injection tools are implemented with an emphasis on safety. An open-source QEMU-based approach that focuses on security was presented by Höller et al. [HSK+14, HKR+15, HMR+15]. Their tool FIES injects permanent and transient faults into RAM and CPU registers. FIES is tailored to ARM CPUs and does not cover fault injection into flash memory. The authors modified QEMU in a similar fashion as in the program execution statistics collection published by Chyłek [Chy09]. The faults are configured via XML files, defining the fault types, number of injected faults, and target components. FIES records the simulation results, such as accessed memory, register addresses, and program counter.

A simpler QEMU-based tool was developed for the security verification of MCUboot [Git20]. The fault analysis, in this case, is limited to instruction skips injected via the GDB interface. QEMU has also been customized to the fault analysis of the Linux kernel [JKNM19] and its applications [GDBF14]. Another use case is the investigation of faults in the context of distributed systems [HKB+10].

The reader is referred to surveys and further literature to obtain additional information about the large ”zoo“ of fault injection tools [KDN14, LT15, GBGG03, JH11, ACK+03, KBBT15, PBR17, BP03, NCDM13].

The search of different fault injection tools showed that, despite promising approaches, the implementations do not fulfill the requirements defined for ARCHIE. The reviewed tools were designed for specific architectures, and adding new devices is problematic since most QEMU-based tools were strongly modified. Most implementations were also limited to a particular fault type, e.g., faulting only instructions or a specific memory type, and hardly any tool was open source.

ARCHIE was developed for the fault emulation of embedded devices and provides a simple interface for data analysis. Since QEMU was extended with only minimal modification, new machines can be easily added to ARCHIE, which makes it architecture-independent. Furthermore, ARCHIE enables the injection of different fault types, such as transient and permanent data *and* instruction faults, into CPU registers, RAM, and flash memory. ARCHIE is open source and available at `https://github.com/Fraunhofer-AISEC/archie`.

### 6.3.2 The Architecture of ARCHIE

In the following, I provide an overview of ARCHIE's essential components and their interaction with one another. I also discuss the input configuration specified by the user and the data processing.

QEMU is the foundation of ARCHIE, which enables emulating faults in embedded devices (e.g., ARM or RISC-V) on a standard computer system (e.g., x86-64). QEMU is independent of the operating system and supports 22 architectures. It also allows simulating peripherals and makes adding new architectures relatively easy. The performance of QEMU is based on its Tiny Code Generator (TCG), enabling the fast translation of guest to host instructions. The TCG groups instructions into TBs and caches them after translation.

Until November 2019, QEMU did not provide an API for low-level guest space access, making interfacing with the TCG difficult. This changed with QEMU 4.2, which was released in November 2019. However, the API still did not provide access to the memory and registers of the guest space. Our fault-injection plugin was added to overcome this drawback, thereby extending the TCG plugin interface. This minimal extension enabled memory and register read/write access. Through the fault-injection plugin, adding new architectures became feasible. This architecture independence is the most important aspect of ARCHIE since previous fault tools required heavy implementation adjustments to add new architectures.

### Input and Output Data

The input to ARCHIE is a binary firmware image of the DuT and fault and QEMU configurations in JSON format. The location of the binary image is specified in the QEMU configuration. The fault configuration describes a fault *campaign*, containing multiple experiments, while one or multiple faults make up an *experiment*. Faults are implemented through bit-flips in virtual memory, defined by the register number or memory address of their occurrence. A fault is defined through seven parameters:

- The *fault address*, hence, a memory address or register, as defined by GDB numbering.
- The *fault type*, which in ARCHIE's case, is an instruction, data, or register fault.
- The *fault model* specifies how the bits set in the "fault mask" will be altered. It is defined as a bit-flip to `0`, to `1`, a toggle, or overwrite. The "overwrite" flag allows replacing the instruction at the "fault address" with a `NOP` instruction.
- The *fault lifespan* defines the lifetime of transient faults through several instructions. A lifespan of 0 marks a permanent fault which remains active throughout the experiment.
- The *fault mask* defines which bits at the "fault address" are to be replaced with the specified "fault model".
- The *trigger address* after which the fault is injected.
- The number of times the trigger address is executed before fault injection is defined as the *trigger counter*.

The simulation output is stored in an HDF5 file containing:

- The pre-golden run, hence, the instructions before the defined starting point of the experiment.
- The golden run, i.e., the results without fault injection.

- The fault experiments documenting only TB changes compared to the golden run.

The HDF5 file contains the TBs' execution order and their content in the form of assembly instructions. ARCHIE also logs the register and memory dumps and memory accesses (read or write).

### Basic Building Blocks

Figure 6.8 shows the building blocks of ARCHIE. Provided with the configuration files and the firmware image, the Controller launches one or multiple Worker Tasks. Experiments are distributed among the Worker Tasks and run independently. To emulate the faults, each Worker Task starts an instance of QEMU and communicates with the fault (injection) plugin via a UNIX FIFO. A more detailed view of the Worker Task's interaction with QEMU is depicted in Figure 6.9. The Worker Task collects data from the fault plugin, removes redundant data and artifacts, and compares the simulation results to the golden run, i.e., the code execution without faults. Finally, the processed data is forwarded to the Logger, which stores the output results in an HDF5 file. ARCHIE allows the user to connect via the GDB interface to observe the fault propagation in detail. However, this slows down the execution and should be avoided for larger data sets.

After the initialization and start of a QEMU subprocess, the Worker Task activates the logging of the guest's control flow and memory accesses in parallel to the experiment execution. Before the fault address is approached, the Worker Task switches to single-stepping mode, where the Translation Blocks TBs — comprising one or several DuT assembly instructions — are reduced to single instructions. This enables a fine-grained injection of the fault. In the case of transient faults, the single-stepping mode is disabled after the specified fault lifespan, while for permanent faults, single-stepping

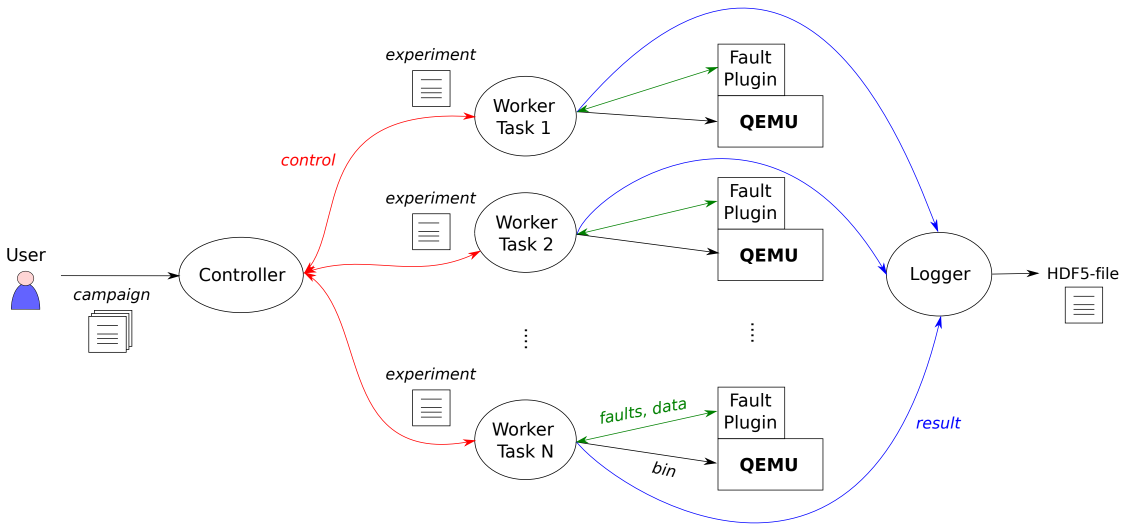


**Figure 6.8:** The components of ARCHIE, comprising the Controller, Worker Tasks, and Logger.

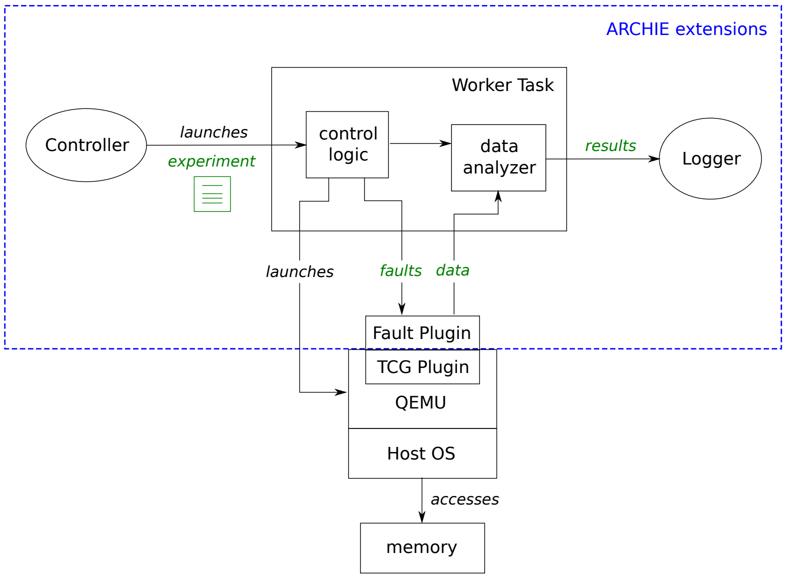


**Figure 6.9:** The Worker Task and its interaction with QEMU.

is deactivated after the fault injection. The experiment execution terminates either at a specified instruction or — if the control flow was diverted — after a maximum number of instructions specified by the user. After the termination, the processed data is forwarded to the Logger, which is the final step of the Worker Task.

A significant advantage of ARCHIE over previous QEMU-based fault emulation tools is that ARCHIE simplifies the integration of different devices and architectures. If the user wants to run tests for, e.g., both RISC-V and ARM, using ARCHIE is straightforward. After compiling the binary file and configuring QEMU, the user adjusts the corresponding fault addresses in the fault configuration and starts ARCHIE. Even though, ARCHIE is easily extensible it is currently restricted to embedded devices with single-core processors. Extending ARCHIE to multicore processors is a potential goal for future improvement.

### Multiple Worker Tasks

The fault emulation can be parallelized through multiple Worker Tasks. However, ARCHIE inhibits the spawning of new Worker Tasks when the monitoring of memory usage and the moving average runtime of Worker Tasks indicate a bottleneck.

A test campaign for the TinyAES implementation from Section 6.2 was run on different host devices to determine the performance of ARCHIE and the impact of multiple Worker Tasks. The focus of the test campaign was to inject single-bit faults into the TinyAES implementation used for experimentation in Section 6.2. The test campaign comprises the S-Box from Section 6.2 together with the last round skip. It serves as verification of a successful ARCHIE installation [ARC21]. Reaching full fault coverage is not the aim of this test but rather injecting a sufficiently large number of faults to investigate the behavior of multiple worker tasks.

In total, 24,794 faults were injected, with host systems ranging from standard laptops to resource-extensive servers with 4 to 64 parallel Worker Tasks. The execution time on the most resource-extensive server, a Dell Poweredge Rack Server with 32 Cores and 192 GB RAM, took 6 min 50 s. A LenovoP15v laptop with 6 cores, 32 GB RAM, and 12 worker tasks ran the test campaign in 24 min 32 s, while a Fujitsu TX140 S2 server with 4 cores, 28 GB RAM, and, in total, 8 worker tasks finished the campaign after 33 minutes 12 s. However, even on the least resource-extensive device, a Lenovo T470s with 2 cores and 20 GB RAM, the execution terminated in 80 min 19 s. This shows that even a common laptop speeds up the fault analysis compared to manual inspection. In general, the execution time will not scale linearly with the amount of spawned Worker Tasks since the Logger acts as a bottleneck and hence, limits the data processing.

### 6.3.3 Simulation Results

Campaigns for two exemplary applications were performed as a test of ARCHIE's capabilities. The first experiment entails an analysis of the TinyAES implementation from Section 6.2, including a differential fault attack on the AES and the AES round skip. The second practical experiment tested the implementation of a secure bootloader, which was verified in an LFI experiment on an Infineon XMC1400. All campaigns were run on ARM and RISC-V architectures.

The following campaigns focused entirely on single-bit faults since multi-bit faults are less likely to occur and significantly increase the computational effort of the campaign. Chatzidimitriou et al. investigated several microprocessors and their susceptibility to spatial multi-bits faults via the GeFIN fault injection framework to get an idea about the occurrence of multi-bit faults [CPG$^+$19]. They injected single and multi-bit faults into microprocessors with fabrication technologies between 250 nm to 22 nm targeting registers and cache. Smaller fabrication technologies are more susceptible to multi-bit faults due to a higher density of memory cells. Chatzidimitriou et al. observed that from 250 nm down to 90 nm, $\leq 6\%$ of failures could be attributed to multi-bit faults, which increased from 65 nm with 8% to 21% for 22 nm technologies. Hence, single-bit faults were, in general, more likely to occur even for smaller fabrication technologies. Furthermore, including multi-bit faults would increase the computational complexity due to the large number of possible fault combinations. The overall goal of the following campaigns is not a full fault coverage for the entire firmware but an analysis of code sections that are of particular interest to the attacker and the corresponding attack scenario.

#### Analysis of TinyAES

The first practical experiment is a differential fault attack on the AES S-box, described in Section 6.2. In this case, transient faults were injected into the AES `SubBytes` function, resulting in multiple faulty ciphertexts that led to the leakage of the secret key.

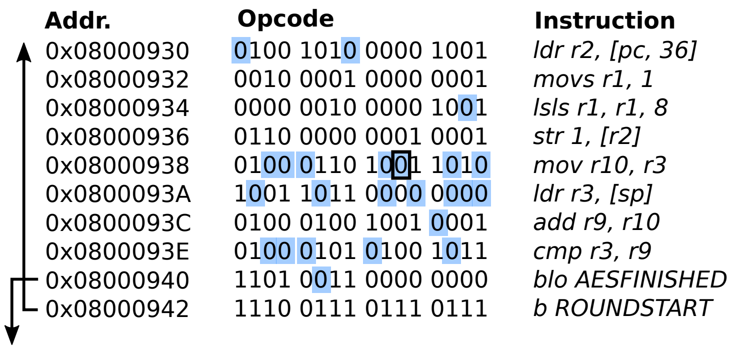
The campaign was run for the STM32F051R8T6 Discovery board with an ARM Cortex-M0 processor by injecting transient bit-flips from `0` to `1` into the S-box stored in flash memory. For the identification of possible fault candidates, i.e., weaknesses in the code, the emulated faulty ciphertexts were compared to the ciphertexts from the practical experiment.

The simulation results showed that the faulty ciphertexts originated from one to four (neighboring) bit-flips into flash memory. However, no fault candidates were found for some faulty ciphertexts since they were possibly generated by modifying instructions instead of data fetches from the S-box.

A second simulation was launched, where — for matters of simplicity — one-bit instruction faults were injected. In total, 588 different fault candidates were identified that led to exploitable faulty ciphertexts. These results show that the fault occurrence and propagation are better understood through a simulation tool than through error-prone and cumbersome manual verification of the binary and assembly instructions.

The second practical experiment related to the AES was a control flow manipulation of the TinyAES implementation described in Section 6.2. The experiment aimed to fault the AES execution in the 9th round to enforce premature termination and calculate the secret key.

Analyzing the assembly instructions manually to determine a possible fault candidate for the practical experiment was cumbersome. Hence, a campaign was launched targeting transient one-bit faults injected into flash memory to find additional fault candidates that lead to a control flow manipulation. As depicted in Figure 6.10, ARCHIE found 23 additional faults leading to a premature termination of the AES algorithm in the 9th round. The black rectangle in Figure 6.10 marks the original fault from the LFI experiment in Section 6.2, which was also found through the campaign. In total, the alteration of ten different bytes in the vicinity of the loop control contained possible fault candidates, including multiple fault possibilities for certain bytes. Finding them through manual research would have been error-prone and time-consuming. In contrast



```
Addr.         Opcode                 Instruction
0x08000930    0100 1010 0000 1001    ldr r2, [pc, 36]
0x08000932    0010 0001 0000 0001    movs r1, 1
0x08000934    0000 0010 0000 1001    lsls r1, r1, 8
0x08000936    0110 0000 0001 0001    str 1, [r2]
0x08000938    0100 0110 1001 1010    mov r10, r3
0x0800093A    1001 1011 0000 0000    ldr r3, [sp]
0x0800093C    0100 0100 1001 0001    add r9, r10
0x0800093E    0100 0101 0100 1011    cmp r3, r9
0x08000940    1101 0011 0000 0000    blo AESFINISHED
0x08000942    1110 0111 0111 0111    b ROUNDSTART
```

**Figure 6.10:** ARM Thumb assembly instructions for skipping the last AES round, as discussed in Section 6.2. The potential fault candidates obtained with ARCHIE are depicted in blue.

to manual analysis, the simulation on a common laptop terminated after only a few minutes.

### Further Campaigns

The second practical use case to test ARCHIE's capabilities was a secure bootloader implementation. A cryptographic signature ensures that the correct software image is loaded. To ensure long-term security in practice, this "signature" is often implemented through symmetric cryptographic primitives, for instance, a Keyed-Hash Message Authentication Code (HMAC) with SHA-256. The verification flag was stored in SRAM.

The main advantage of ARCHIE is that the fault emulation can be easily adjusted to different devices and architectures while remaining its maintainability. A campaign was launched to identify possible vulnerabilities, targeting different DuTs with ARM and RISC-V architectures. ARCHIE was provided with an invalid software image, and permanent one-bit faults were injected into the SRAM region of the bootloader. After successful fault injection, the software image with the invalid signature was accepted as verified. The signature verification was successfully faulted between the write of the comparison result and its read-out to determine if the software image should be booted. This vulnerability was verified on both systems (ARM and RISC-V).

The fault injection into the secure bootloader implementation was successfully reproduced in an LFI experiment with a 1064 nm Solid State Laser (single mode) targeting an Infineon XMC1400 with an ARM Cortex-M0. The DuT was prepared through decapsulation and additional thinning of the substrate. In a backside attack, the verification flag for the bootloader stored in SRAM was successfully faulted. Hence, the vulnerability was verified in the practical experiment.

One of ARCHIE's merits is that different architectures can be easily added to the fault emulation. All campaigns were run for an ARM Cortex-M0 and a 32-bit RV32IMC processor to compare our simulation results for different architectures. For the secure bootloader, the same vulnerability was discovered for both architectures. The simulation results for the differential fault attack targeting the AES `SubBytes` function identified 632 possible one-bit faults leading to 526 unique *exploitable* ciphertexts for the RV32IMC processor. For the ARM Cortex-M0, 588 different fault candidates were found.

Only two possible fault candidates for the RISC-V platform and the AES round skip campaign were found compared to 24 exploitable faults for the ARM Cortex-M0. In the RISC-V example, the relevant branch instruction is `bne t6, t4, 0x80000190`. Hence, the program counter jumps to address `0x80000190`, resuming regular operation if `t6` $\neq$ `t4`. For one-bit faults, the control flow is diverted only if the comparison fails, hence, either if the instruction is changed to `ble` (branch if less than or equal) or if the second operand is set to `t6`.

Hence, the discrepancies in the simulation results are attributed to differences in the compiled code. This shows that different compiler versions and settings affect the potential fault candidates. Thus, a source code analysis alone is not sufficient to determine altered assembly instructions.

### 6.3.4 Summary

The practical results from various use cases demonstrate that automated fault testing significantly reduces the time to analyze the code compared to manual inspection and yields significantly more fault candidates that were overlooked in the manual inspection of the assembly instruction. With ARCHIE, transient and permanent faults were injected into flash memory, RAM, and registers, covering data and instruction faults. Different architectures are easily added through only minimal modifications of the TCG plugin interface, hence, providing architecture independence. The results show that ARCHIE's execution time yields good performance even on a common laptop. ARCHIE is open-source and freely available on GitHub at `https://github.com/Fraunhofer-AISEC/archie`.

## 6.4 Analysis of the Embedded Key Management System

This section discusses vulnerable parts of the EKMS that can be compromised through fault attacks. Furthermore, with the help of ARCHIE, I investigate whether the re-enrollment capabilities of FORTRESS can be exploited such that the EKMS does not detect partial removal or deliberate aging.

### 6.4.1 The Critical Layers of the EKMS

Figure 6.11 depicts the structure of the FORTRESS software, the second generation EKMS discussed in Chapter 3. The FORTRESS operating system was hardened and extended through a secure life cycle with re-enrollment capabilities. The EKMS includes 128- and 256-bit AES (ECB and CBC) for encryption, SHA256 for hashing, and ECC secp256k1 [SEC09, SEC10] for signing. The EKMS is built modularly, such that it can easily be extended through different cryptographic algorithms that are added to *Crypto-Func* and made accessible through the *Crypto-API*. When adding new cryptographic algorithms to the EKMS, the available hardware resources have to be considered.

The critical layers of the EKMS core with a greater susceptibility toward fault attacks are highlighted in yellow in Figure 6.11. The *Crypto-Functions* rely on the implementation of cryptographic algorithms that are hardened against fault attacks and side-channel analysis. Since this is a broad and universal topic, I will exclude it from further analysis. Hence, the hardened OS, drivers, measurement, and specific user applications are excluded from further analysis.

As shown in Section 6.2, injected faults can leak secret keys and alter the control flow of cryptographic algorithms. Assuming the injecting of faults through radiation without disassembly of the enclosure, the attacker can derive information about the secret key by collecting ciphertexts. A possible use case is FORTRESS serving as an HSM for confidential communication, where data is encrypted and sent over the external UART interface. An attacker could intercept the encrypted data and hence, gain access to ciphertexts. Thus, the *Crypto-Func* layer has to be hardened against fault injection.

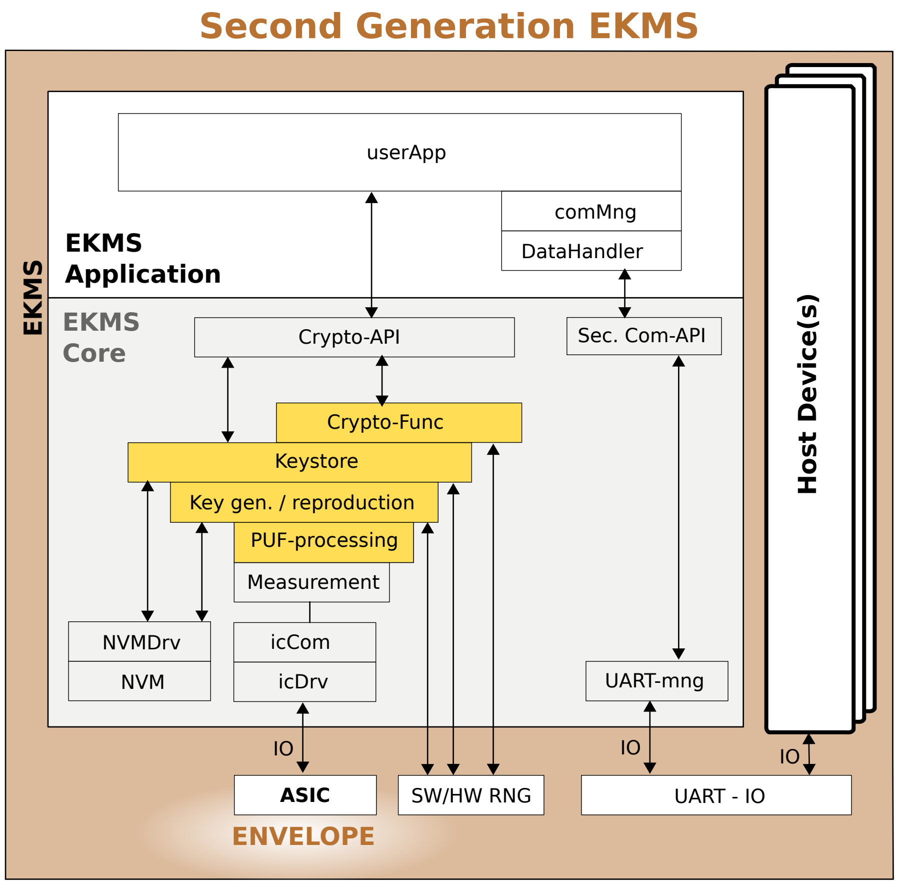


**Figure 6.11:** Layers of the second generation EKMS (as depicted in Figure 3.8), with critical layers highlighted in yellow.

The *Keystore* manages the key chain, verifies the CSPs for the secure system life cycle, and checks the enrollment state. Hence, an attacker could target the re-enrollment process, faulting the verification of the enrollment state. Through this, the attacker could manipulate the maximum number of possible re-enrollments, potentially allowing for the partial removal of the enclosure. Fault injection can support the partial removal of the enclosure. An attacker could use the re-enrollment function in field mode to first remove a very small portion of the enclosure and then trigger re-enrollment to generate a new PUF-key that incorporates the minuscule alterations of the enclosure. The partial removal can be, in general, accompanied by re-soldering, i.e., re-connecting severed traces. In the next step, the attacker again removes a portion of the enclosure — small enough to not be recognized through the measurement — and then re-triggers re-enrollment to generate a new PUF-key. Through this, the attacker could gradually generate a hole in the enclosure, large enough to insert probing needles to read out critical signals of the PCB.

During re-enrollment, the original KEK, i.e., the hashed PUF-secret, is compared against the newly created KEK. This comparison could be faulted, theoretically leading to the acceptance of a slightly modified PUF-key. In this case, the gradual partial removal described above could be conducted without re-enrollment. However, since this attack is a lot more challenging than provoking an additional re-enrollment, for the following analysis, I focus on verifying the enrollment state implemented in the *Keystore*.

### 6.4.2 Attacking Re-Enrollment

As an exemplary code analysis, I investigate the fault susceptibility of the EKMS' re-enrollment extension.

The *Keystore* verifies and sets the correct enrollment state during each stage of the secure life cycle. To account for changes in the PUF-response through aging effects, the EKMS accepts one re-enrollment in field mode over the device's lifetime. This number was chosen to counteract possible helper data manipulation attacks, as described in Section 3.2.

If additional re-enrollments are allowed, an attacker could gradually alter the PUF-response, subsequently removing parts of the enclosure. An additional number of re-enrollments could be achieved by altering the re-enrollment counter, which is the focus of the following analysis. If this remains undetected by the EKMS, an attacker could gradually remove small portions of the enclosure through re-enrollment and potentially probe signals on the PCB.

In Section 6.1, I explained that the enclosure, in general, has to be considered susceptible to radiation faults. Radiation faults, however, can lead to unidirectional bit-flips, for instance, from `0` to `1`. For the following analysis, I focus on single-bit faults since multi-bit faults are less likely to occur even for small fabrication technologies [CPG$^+$19]. Just as for the campaigns in Section 6.3, I focus on the EKMS code sections related to the described attack scenario instead of analyzing the full FORTRESS software. I inject all possible single-bit fault patterns into the code sections that are of particular interest for the attack.

The target of the attack herein is to fault the verification of the correct enrollment state; more precisely, I focus on the counter increment after each re-enrollment. This is implemented in the `keystoreCheckNextState` function of the EKMS. The number of possible re-enrollments in field mode is restricted to a maximum number of 1.

To emulate the injection of unidirectional faults in the EKMS enrollment verification with ARCHIE, I set the configuration to permanent single **`1`**-bit faults injected into flash memory. I tested the emulation exemplarily for the STM32F0 Discovery board, which has an ARM Cortex-M0 processor. Figure 6.12 depicts the assembly instructions executed before the verification of the re-enrollment counter. At address `0x080000de`, the current re-enrollment counter is loaded (`ldr r0, [r2, 0x34]`) into register `r0` and checked against the specified maximum number of re-enrollments (`cmp r0, 0`). Hence, if the counter is `0`, a re-enrollment is allowed (`beq REENROLLMENT`). Before loading the current re-enrollment counter, the EKMS checks if the system is in a valid state that allows re-enrollment.

In total, I found 15 single-bit permanent faults leading to the acceptance of an enrollment count that exceeds the maximum number of permitted re-enrollments. The faults

| **Addr.** | **Opcode** | **Instruction** |
|---|---|---|
| 0x080000cc | 0100 0010 1000 0100 | *cmp r4, r0* |
| 0x080000ce | 1101 0000 0000 1101 | *beq STATECHECK* |
| 0x080000d0 | 0100 1000 0001 1011 | *ldr r0, [STATE]* |
| 0x080000d2 | 0100 0010 1000 0100 | *cmp r4, r0* |
| 0x080000d4 | 1101 0000 0000 1010 | *beq STATECHECK* |
| 0x080000d6 | 0100 0010 1010 1100 | *cmp r4, r5* |
| 0x080000d8 | 1101 0001 0000 0100 | *bne INVALIDMODE* |
| 0x080000da | 0100 0010 1010 0011 | *cmp r3, r4* |
| 0x080000dc | 1101 0001 0010 0011 | *bne SWITCHSTATE* |
| 0x080000de | 0110 1011 0101 0000 | *ldr r0, [r2, 0x34]* |
| 0x080000e0 | 0010 1000 0000 0000 | *cmp r0, 0* |
| 0x080000e2 | 1101 0000 0001 1011 | *beq REENROLLMENT* |

**Figure 6.12:** Potential fault candidates targeting the re-enrollment counter check (address `0x080000e0`).

range from instruction changes to altered registers and immediate values. Particularly, changes from `cmp` to `ldr` operations as induced by faults during the state verification are difficult to anticipate. In this case, instead of a `cmp`, a `ldr` instruction is executed that influences the result of the re-enrollment counter verification.

This shows that even single-bit faults can lead to control flow deviations, which can lead to additional re-enrollments that severely impact the overall security of the enclosure. In the next section, I discuss possible countermeasures against the fault injection attacks presented in this chapter.

## 6.5 Countermeasures Against Fault Attacks

This section focuses on countermeasures against fault attacks, discussing redundancy, memory error correction, secure coding, and architectural changes.

### 6.5.1 Redundancy

To counteract fault attacks on block ciphers, redundant calculation or hardware redundancy, such as duplication of functions, modules, or even CPUs, was proposed. Countermeasures come in the form of time, information, or spatial redundancy. The most common redundant countermeasure is recomputing the encryption and comparing the calculation results [MSY06]. However, since this comes with a considerable performance overhead, less time-consuming variations were proposed, such as duplicate calculation using both clock edges [ML08] or recomputing with shifted operands [CML$^+$11].

Information redundancy based on parity techniques aims at validating certain bits of the input message after the computation, which usually comes with an increased hardware overhead [BBK$^+$03]. To reduce the overhead, Wu et al. limited the parity to one bit per 128-bit output, which is verified in each round of the AES calculation [WKKG04]. Furthermore, parity implementations based on logic gates were proposed [MKRM10]. An implementation-independent parity-based scheme for S-box protection was published by

Mozaffari-Kermani and Reyhani-Masoleh [MKRM10, MKRM11]. However, countermeasures based on parity bits do not detect an even number of faults.

To overcome this drawback, a more robust scheme proving uniform fault coverage was proposed, which, however, comes with a significant hardware overhead [KKT04, KKT07]. Furthermore, cyclic redundancy check codes were proposed to counteract fault analysis [YW06].

The calculation is followed by its inverse in a different approach by Karri et al., comparing the computation results [KWMK02]. This scheme was accelerated through pipelining optimizations to decrease the performance overhead [SSHA08, RBMK10].

In 2018, Zhang et al. demonstrated that persistent fault attacks on block ciphers defeat fault attack countermeasures, where the calculation was performed with two redundant modules [ZLZ+18]. In this case, the fault was injected before the encryption stage, altering a stored constant of the algorithm through persistent faults. The authors obtained multiple faulty ciphertexts and calculated the secret key. Pan et al. showed that even masking attempts to counteract persistent fault attacks can be broken through fault injection [PZRB19].

Another attack proposed by Saha et al. demonstrated that redundancy-based countermeasures could be broken [SJB+18]. They combined laser fault injection with power side-channel measurements, assuming a random fault model.

In the same year, Dobraunig et al. attacked the simple time redundancy of an AES software implementation [DEK+18] via Statistical Ineffective Fault Analysis (SIFA). SIFA can be seen as an intersection between Statistical Fault Analysis (SFA) [FJLT13] and Ineffective Fault Analysis (IFA) [Cla07]. The authors exploited the non-uniform distribution of *fault-free* ciphertexts stemming from faults in intermediate values that were induced by clock glitching. Unlike IFA-based attacks, the proposed approach does not rely on stuck-at-faults that are difficult to achieve in practice but only requires ineffective faults that lead to a biased distribution.

These attacks show that countermeasures based on hardware redundancy do not provide sufficient security for block ciphers. Furthermore, redundancy-based measures do not necessarily detect permanent faults in all rounds of the block cipher calculation [CML+11].

## 6.5.2 Error Detection

In the following, I focus on error detection in microcontrollers and discuss a flash-aware error detection scheme counteracting permanent and transient fault injection into flash memory. I presented the results herein at the 26th IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS) [GO20].

### Error Detection in Microcontrollers

Error detection in microcontrollers is usually implemented through error correction codes, which can correct one error and detect a maximum of two errors (SECDED). In many cases, manufacturers apply Hamming codes to verify SRAM and flash integrity of microcontrollers, such as the MPC55xx [Tos15], STM32H7 [STM22a], or TMSx70 [Noh11]. Error correction codes are usually applied to enhance system reliability in safety-critical or harsh environments, such as in aerospace applications, e.g., for the GR712RC [Fro23]. However, the integration of error correction requires additional check bits compared to simple error detection. In safety-critical devices, the focus is on correcting errors to maintain system availability, while in high-security applications, error correction weakens the detectability of an attack. Although error correction codes might mark detected errors through flags, this does not prevent the code's execution and exploitation by an attacker. Hence, the additional check bits of the error correction code should be traded for a more robust error detection scheme.

### Flash-Aware Error Detection Scheme

Fault injection into flash memory can occur either through the permanent erasure of bits, for instance, via UV-C light [OT17], or through transient faults, as shown in Section 6.2. Both attacks produce unidirectional faults; hence, bits are only flipped from 0 to 1. This aspect has to be considered by the error detection scheme.

Unidirectional errors, where bits are solely flipped into one direction, were investigated by Berger [Ber61] in 1961, who proposed sum codes for their detection. Sum codes detect *all* unidirectional errors in an $(n + m)$ bit wide codeword $F$ without correcting them. The codeword $F$ is the concatenation $[D\ C]$ of a data word $D$ of $n$ bits length, and the $m$ bit check word $C$, where $m = \lfloor \log_2 n \rfloor + 1$. The check word is computed as $C = \sum_{i=0}^{n-1} (1 \oplus D_i)$, where $D_i$ denotes bit $i$ in the data word $D$. For the microcontroller in Section 6.2, where bits are flipped from `0` to `1`, this is equivalent to counting all zero-bits in the data word $D$. Alternatively, a similar approach is to count all one-bits, and invert the result, i.e., $C' = \overline{\sum_{i=0}^{n-1} D_i}$.

Eq. (6.3) shows an example for the codeword $F = [D\ C]$, comprised of the 32-bit data word $D$ with ten zero-bits and the 6-bit check word $C$.

$$F = [D\ C] = [\underbrace{1110\ 0101\ 1111\ 1010\ 1100\ 1110\ 0111\ 0111}_{D} \quad \underbrace{001010}_{C}] \tag{6.3}$$

Sum codes enforce an identical lower and upper bound for $D$; hence, any injected fault can be detected. Flipping any bit in $D$ to `1` reduces the *present* number of zero-bits, while any bit-flip to `1` in the check word $C$ increases the *expected* number of zero-bits, hence creating a mismatch. This mismatch also occurs if bits are simultaneously flipped in $D$ and $C$.

Depending on the flash architecture, the unidirectional errors can either be injected zero-bits or injected one-bits. In the case of injected one-bits, the argumentation for the upper and lower bound is mirrored, but the scheme is still valid. Furthermore, the check word size scales well with $\mathcal{O}(\log_2(n))$. Hence, doubling the length of the above data word to 64 bits requires only one additional check bit.

### 6.5.3 Secure Coding and Architectural Choices

Apart from the redundant execution of cryptographic algorithms and implementing a suitable error detection scheme, a further countermeasure against fault attacks is the application of secure coding principles.

Coding standards, such as MISRA C [MIS13], the CERT C Coding Standard [Sea08], or the MITRE Common Weakness Enumeration [MIT22], provide basic guidelines to enhance the safety and security of the written code by reducing arithmetic, memory access, and design vulnerabilities. However, they are not necessarily sufficient to protect against fault injection. Therefore, additional measures should be integrated into the code to detect injected faults and trigger a tamper response, such as secure booleans, the repeated verification of loop counters and branch conditions, or handling additional checkpoints to verify the program flow.

Additionally, architectural changes could be beneficial to the security of FORTRESS. Instead of further developing and hardening the EKMS, the functionality of FORTRESS could be integrated into already existing secure elements [Ram21, STM19] that have undergone extensive testing. Secure enclaves [ARM22, Gue16] can also provide benefits through isolated execution or memory partition. However, secure enclaves are not physically isolated from other parts of the system. Hence, they, in general, are vulnerable to hardware attacks [MOG$^+$20, TMS$^+$13] and thus require further hardening.

The goal in hardening a system against fault attacks should be to not rely on a single countermeasure but to combine different countermeasures in a multi-layered approach.

## 6.6 Conclusion

In this chapter, I discussed the effects of fault attacks on the capacitive PUF-based enclosure and cryptographic algorithms in particular.

In Section 6.1, I analyzed the susceptibility of FORTRESS to different types of fault injection. I explained why, even though FORTRESS is enveloped by the enclosure, it is still vulnerable to radiation faults.

In Section 6.2, I demonstrated how transient fault injection could severely impact the execution of cryptographic algorithms and even lead to the leakage of secret keys. I showed that even coarse laser settings lead to exploitable faults through data *and* instruction fetch manipulations. The laser position, spot size, and timing were easily adjusted through a simple calibration technique.

To reduce the effort of finding possible fault candidates, I introduced ARCHIE in Section 6.3. ARCHIE, as an architecture-independent framework for fault evaluation, enables the automated search for possible fault candidates on different processor architectures. I investigated the potential weaknesses of the FORTRESS software in Section 6.4. This was followed in Section 6.5 by a discussion of countermeasures, including a flash-aware error detection scheme.

In addition to the discussion in Chapter 4, I analyzed a further class of attacks targeting the capacitive enclosure.

Future work should focus on implementing additional countermeasures and extensive testing of FORTRESS. Moreover, different trusted execution environments and secure elements should be evaluated to determine whether they are suitable to extend or even replace the existing EKMS.

# Chapter 7

# Conclusion and Outlook

In this dissertation, I presented solutions to various open issues in the context of the capacitive enclosure, focusing on tamper-sensitivity. This includes its integration into a full HSM prototype (Chapter 3), the further development of its error correction (Chapter 5), and an analysis of its behavior under physical attacks (Chapter 4) and fault injection (Chapter 6).

## 7.1 Contributions

In Chapter 3, I described how the enclosure, the ASIC, and the EKMS are combined and further developed into FORTRESS, a PUF-based prototype HSM. The FORTRESS software — the second generation EKMS — includes:

- a hardened OS,
- a full key generation tailored to the capacitive enclosure, and
- a secure life cycle, incorporating supply chain aspects.

This was followed in Chapter 4 by an analysis of the three most relevant physical attacks threatening the security and tamper-sensitivity of the capacitive enclosure. As an extension to the previously investigated 300 µm drilling attacks, I discussed the feasibility of micro-drilling attacks and evaluated different countermeasures. Furthermore, I proposed a countermeasure to the bypassing attack published by Obermaier [Obe19] that can be integrated into the measurement circuit or the enclosure itself. As a third attack, I demonstrated that the communication between the FORTRESS microcontroller and the ASIC could be intercepted via magnetic probing of the SPI interface. To restore the tamper-sensitivity of FORTRESS, I evaluated various countermeasures.

In Chapter 5, I presented a wiretap code for the capacitive enclosure implemented via $q-$ary polar codes. In a first step, I analyzed the PUF-distribution of the capacitive enclosure and its behavior under thermal changes and drilling attacks. From this, I derived an enclosure model, considering all post-processing steps. The proposed polar code construction based on a higher order alphabet provides a separate model for the legitimate and the attacker channel. It achieves a physical layer security of 100 bits for a PUF-response of 306 bits length.

To extend the attack analysis of Chapter 4, I investigated the effects of fault injection on the capacitive enclosure, focusing on cryptographic algorithms in Chapter 6. I identified radiation faults as a possible threat and demonstrated how injected faults impact the security of cryptographic algorithms. Through ARCHIE, an architecture-independent framework for fault analysis, I showed that the cumbersome search for fault candidates is simplified and sped up through automatic firmware analysis. With ARCHIE, I investigated critical sections of the EKMS and discussed possible countermeasures.

By proposing schemes and countermeasures as a solution to various security issues, I improved the tamper-sensitivity of the capacitive PUF-based enclosure, thereby taking the next steps toward its commercial deployment.

## 7.2 Future Work

Despite all the proposed solutions and progress made, there is still a prospect for further development of the enclosure system.

The assembly of FORTRESS would be facilitated by increasing the flexibility of the envelope. This could be achieved by reducing the layer stack thickness and testing the suitability of different materials within the production process. This also entails enlarging the masks within the lithographic process to achieve larger envelope sizes. Furthermore, materials and concepts of wearable electronics could be applied to the enclosure design to enhance flexibility.

The measurement IC integrated into the envelope is optimized in terms of size. Hence, the ASIC's resolution has to be improved to achieve an accuracy comparable to the discrete measurement circuit. This could be achieved in further revisions of the ASIC, tailoring the resolution to the specific femtofarad scale of the enclosure.

The most obvious measure to counteract magnetic probing attacks of the SPI communication between the ASIC and the microcontroller is an elimination of the interface itself. Hence, manufacturing a custom ASIC incorporating both the EKMS and IC functionality should, therefore, be attempted. In general, a multi-layered approach aiming at an in-depth defense is most promising to counteract physical attacks (Chapter 4) and, in particular, fault attacks (Chapter 6). Thus, further proposed countermeasures should be implemented and tested to harden the FORTRESS system.

Regarding the PUF post-processing, the wiretap code construction, discussed in Chapter 5, should be optimized toward larger temperature scales. So far, mainly thermal changes were analyzed and modeled. However, commercial deployment of FORTRESS requires an investigation of various environmental effects and physical quantities, such as humidity or vibrations. Therefore, the obtained measurement results should be incorporated into the polar code construction.

Advancing all these open issues sets the agenda for future work in the context of the capacitive PUF-based security enclosure.

# Bibliography

[AAA$^+$90] Jean Arlat, Martine Aguera, Louis Amat, Yves Crouzet, Jean-Charles Fabre, Jean-Claude Laprie, Eliane Martins, and David Powell. Fault Injection for Dependability Validation: A Methodology and Some Applications. *Software Engineering, IEEE Transactions on*, 16(2):166 – 182, March 1990.

[AAR$^+$18] Nikolaos Athanasios Anagnostopoulos, Tolga Arul, Markus Rosenstihl, André Schaller, Sebastian Gabmeyer, and Stefan Katzenbeisser. Low-temperature data remanence attacks against intrinsic sram pufs. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pages 581–585, 2018.

[ACK$^+$03] Jean Arlat, Yves Crouzet, Johan Karlsson, Peter Folkesson, Emmerich Fuchs, and Günther H Leber. Comparison of physical and software-implemented fault injection techniques. *IEEE Transactions on Computers*, 52(9):1115–1133, 2003.

[ADG96] Nabil Mahmoud Amer, David Peter DiVincenzo, and Neil Gershenfeld. Tamper detection using bulk multiple scattering, 1996. US Patent 5790025, filed in 1996, published in 1998.

[Adv12] Advanced Cryptographic Hardware Development IBM Poughkeepsie and IBM Research. *IBM 4765 cryptographic coprocessor security module – security policy.* Zürich, December 2012.

[ALMT17] Nejmeddine Alimi, Younes Lahbib, Mohsen Machhout, and Rached Tourki. An rtos-based fault injection simulator for embedded processors. *International Journal of Advanced Computer Science and Applications*, 8, June 2017.

[Ame] American Probe & Technologies, Inc. 71T Tungsten Wire Probe. `https://www.americanprobe.com/71t-tungsten-wire-probe.html`. Online, accessed July 28, 2021.

[ARC21] ARCHIE AES Example. `https://github.com/tibersam/archie-aes-example`, 2021. Online, published on December 7, 2021.

[Ari09] Erdal Arikan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on information Theory*, 55(7):3051–3073, 2009.

[ARM22] TRUSTZONE FOR CORTEX-M. `https://www.arm.com/technologies/trustzone-for-cortex-m`, 2022. Online, accessed July 24, 2022.

[AVFK03] Joakim Aidemark, Jonny Vinter, Peter Folkesson, and Johan Karlsson. GOOFI: generic object-oriented fault injection tool. In *2003 International Conference on Dependable Systems and Networks, 2003. Proceedings.*, pages 668–668, July 2003.

[BB19] Elaine Barker and William C. Barker. *Recommendation for Key Management: Part 2 – Rev. 1 – Best Practices for Key Management Organizations.* National Institute of Standards and Technology (NIST), Gaithersburg, MD, May 2019.

[BBC$^+$14] Maël Berthier, Julien Bringer, Hervé Chabanne, Thanh-Ha Le, Lionel Rivière, and Victor Servant. Idea: Embedded fault injection simulator on smartcard. In Jan Jürjens, Frank Piessens, and Nataliia Bielova, editors, *Engineering Secure Software and Systems*, pages 222–229, Cham, 2014. Springer International Publishing.

[BBK$^+$03] Guido Marco Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. *IEEE Transactions on Computers*, 52(4):492–505, 2003.

[BBK$^+$12] Markus Becker, Daniel Baldin, Christoph Kuznik, Mabel Joy, Tao Xie, and Wolfgang Mueller. XEMU: An efficient QEMU based binary mutation testing framework for embedded software. In *EMSOFT'12 - Proceedings of the 10th ACM International Conference on Embedded Software 2012, Co-located with ESWEEK*, pages 33–42, October 2012.

[Bec15a] Georg T. Becker. On the pitfalls of using arbiter-PUFs as building blocks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(8):1295–1307, 2015.

[Bec15b] Georg T. Becker. The gap between promise and reality: On the insecurity of XOR arbiter PUFs. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 535–555. Springer, 2015.

[Bel05] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, page 41, USA, 2005. USENIX Association.

[Ber61] Jay M Berger. A note on error detection codes for asymmetric channels. *Information and control*, 4(1):68–73, 1961.

[Ber68] Elwyn R. Berlekamp. Nonbinary BCH decoding. *IEEE Transactions on Information Theory*, 14(2):242–242, 1968.

[BF78] Jorgen Brosow and Erik Furugard. Method and system for verifying authenticity safe against forgery, 1978. US Patent 4218674, filed in 1978, published in 1980.

[BGPFC12] M. Bagatin, S. Gerardin, A. Paccagnella, and V. Ferlet-Cavrois. Alpha-induced soft errors in floating gate flash memories. In *2012 IEEE International Reliability Physics Symposium (IRPS)*, pages 3C.2.1–3C.2.7, 2012.

[BGS$^+$08] Christoph Bösch, Jorge Guajardo, Ahmad-Reza Sadeghi, Jamshid Shokrollahi, and Pim Tuyls. Efficient Helper Data Key Extractor on FPGAs. In *Proceeding Sof the 10th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 181–197. Springer Berlin Heidelberg, 2008.

[Bit21] Bitkom. Ausgaben für IT-Sicherheit in Deutschland in den Jahren 2017 bis 2020 und Prognose bis 2025 (in Milliarden Euro) [Graph]. `https://de-statista-com.eaccess.ub.tum.de/statistik/daten/studie/1041736/umfrage/ausgaben-fuer-it-security-in-deutschland/`, 2021. In Statista. Online, accessed February 8, 2022.

[BK14] Georg T. Becker and Raghavan Kumar. Active and passive side-channel attacks on delay based PUF designs. *Cryptology ePrint Archive*, 2014.

[BNCF13] Lilian Bossuet, Xuan Thuy Ngo, Zouha Cherif, and Viktor Fischer. A PUF based on a transient effect ring oscillator and insensitive to locking phenomenon. *IEEE Transactions on Emerging Topics in Computing*, 2(1):30–36, 2013.

[Boa81] David G. Boak. A History of U.S. Communications Security (U). `https://www.nsa.gov/portals/75/documents/news-features/declassified-documents/cryptologic-histories/history_comsec_ii.pdf`, 1981. National Security Agency. Online, accessed February 22, 2022.

[Boc21] René Bocksch. Wachstumsmarkt IT-Sicherheit [Digitales Bild]. `https://de-statista-com.eaccess.ub.tum.de/infografik/26033/ausgaben-fuer-it-sicherheit-in-deutschland/`, 2021. In Statista. Online, accessed February 8, 2022.

[BP03] Alfredo Benso and Paolo Prinetto. *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation.* Springer Science + Business Media, New York, January 2003.

[Bri04] Gary A. Brist. Design optimization of single-ended and differential impedance PCB transmission lines. In *PCB West Conference Proceedings*, 2004.

[BSPB15] Alexios Balatsoukas-Stimming, Mani Bastani Parizi, and Andreas Burg. LLR-Based Successive Cancellation List Decoding of Polar Codes. *IEEE Transactions on Signal Processing*, 63(19):5165–5179, 2015.

[BY19] Yonghong Bai and Zhiyuan Yan. A Secure and Robust Key Generation Method Using Physical Unclonable Functions and Polar Codes. In *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, pages 254–259, 2019.

[BY21] Yonghong Bai and Zhiyuan Yan. A Secure and Robust PUF-based Key Generation with Wiretap Polar Coset Codes. *Journal of Electronic Testing*, 37, June 2021.

[CCH97] H.P. Chou, T.C. Chou, and T.H. Hau. Evaluation of high density DRAMs as a nuclear radiation detector. *Applied Radiation and Isotopes*, 48(10):1601–1604, 1997.

[CCL$^+$11] Qingqing Chen, György Csaba, Paolo Lugli, Ulf Schlichtmann, and Ulrich Rührmair. The bistable ring PUF: A new architecture for strong physical unclonable functions. In *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 134–141. IEEE, 2011.

[CG12] Sławomir Chyłek and Marcin Goliszewski. Qemu-based fault injection framework. *Studia Informatica*, 33:25–42, January 2012.

[Cha84] David Chaum. Design concepts for tamper responding systems. In *Advances in Cryptology*, pages 387–392. Springer, 1984.

[Chi64] Robert Chien. Cyclic decoding procedures for Bose- Chaudhuri-Hocquenghem codes. *IEEE Transactions on Information Theory*, 10(4):357–363, 1964.

[Chy09] Sławomir Chyłek. Collecting program execution statistics with qemu processor emulator. *2009 International Multiconference on Computer Science and Information Technology*, pages 555–558, 2009.

[CIW$^+$17] Bin Chen, Tanya Ignatenko, Frans M. J. Willems, Roel Maes, Erik van der Sluis, and Georgios Selimis. A Robust SRAM-PUF Key Generation Scheme Based on Polar Codes. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–6, 2017.

[CK78] Imre Csiszár and Janos Korner. Broadcast Channels with Confidential Messages. *IEEE transactions on Information Theory*, 24(3):339–348, 1978.

[Cla07] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 181–194, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[CLMFT14] Franck Courbon, Philippe Loubet-Moundi, Jacques JA Fournier, and Assia Tria. Increasing the efficiency of laser fault injections using fast gate level reverse engineering. In *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 60–63, 2014.

[CMD+18] Brice Colombier, Alexandre Menu, Jean-Max Dutertre, Pierre-Alain Moëllic, Jean-Baptiste Rigaud, and Jean-Luc Danger. Laser-induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-bit Microcontroller. *IACR Cryptology ePrint Archive*, 2019:1024, 2018.

[CMH20] Durba Chatterjee, Debdeep Mukhopadhyay, and Aritra Hazra. Interpose PUF can be PAC Learned. *Cryptology ePrint Archive*, 2020.

[CML+11] Gaetan Canivet, Paolo Maistri, Régis Leveugle, Jessy Clédière, Florent Valette, and Marc Renaudin. Glitch and laser fault attacks onto a secure AES implementation on a SRAM-Based FPGA. *Journal of Cryptology*, 24:247–268, April 2011.

[CMR+01] Pierluigi Civera, Luca Macchiarulo, Maurizio Rebaudengo, Matteo Sonza Reorda, and A. Violante. Exploiting FPGA for accelerating fault injection experiments. In *Proceedings Seventh International On-Line Testing Workshop*, pages 9–13. IEEE, 2001.

[CMS98] Joao Carreira, Henrique Madeira, and Joao Gabriel Silva. Xception: a technique for the experimental evaluation of dependability in modern computers. *IEEE Transactions on Software Engineering*, 24(2):125–136, 1998.

[Col02] Elliott Cole. *Application Report SLLA030C – Reducing Electromagnetic Interference (EMI) With Low Voltage Differential Signaling (LVDS).* Texas Instruments, June 2002.

[CPC+01] G. Cellere, P. Pellati, A. Chimenton, J. Wyss, A. Modelli, L. Larcher, and A. Paccagnella. Radiation effects on floating-gate memory cells. *IEEE Transactions on Nuclear Science*, 48(6):2222–2228, 2001.

[CPG+19] Athanasios Chatzidimitriou, George Papadimitriou, Christos Gavanas, George Katsoridas, and Dimitris Gizopoulos. Multi-bit upsets vulnerability analysis of modern microprocessors. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*, pages 119–130, 2019.

[CRR98] Ramamurti Chandramouli, N Ranganathan, and Shivaraman J Ramadoss. Adaptive quantization and fast error-resilient entropy coding for image transmission. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(4):411–421, 1998.

[CS16] Sonam Chauhan and Ajay Sharma. Fuzzy commitment scheme based on reed solomon codes. In *Proceedings of the 9th International Conference on Security of Information and Networks*, pages 96–99, 2016.

[CSS$^+$11] Luiz H. Claro, A. A. Silva, Jose A. Santos, Suzy F.L. Nogueira, and Ary G. Barrios Junior. Evaluation of soft errors rate in a commercial memory EEPROM. In *International Nuclear Atlantic Conference*, July 2011.

[CSW17] Franck Courbon, Sergei Skorobogatov, and Christopher Woods. Reverse Engineering Flash EEPROM Memories Using Scanning Electron Microscopy. In Kerstin Lemke-Rust and Michael Tunstall, editors, *Smart Card Research and Advanced Applications*, pages 57–72. Springer, 2017.

[CW19] Bin Chen and Frans M. J. Willems. Secret Key Generation Over Biased Physical Unclonable Functions With Polar Codes. *IEEE Internet of Things Journal*, 6:435–445, 2019.

[DDR$^+$12] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, P. Orsatelli, Philippe Maurine, and Assia Tria. Injection of transient faults using electromagnetic pulses -practical results on a cryptographic system-. Cryptology ePrint Archive, Paper 2012/123, 2012. `https://eprint.iacr.org/2012/123`.

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

[DEK$^+$18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: Exploiting ineffective fault inductions on symmetric cryptography. Cryptology ePrint Archive, Paper 2018/071, 2018. `https://eprint.iacr.org/2018/071`.

[DGFFP13] Giuseppe Di Guglielmo, Davide Ferraretto, Franco Fummi, and Graziano Pravadelli. Efficient fault simulation through dynamic binary translation for dependability analysis of embedded software. In *Proceedings - 2013 18th IEEE European Test Symposium, ETS 2013*, pages 1–6, May 2013.

[DGSV15] Jeroen Delvaux, Dawu Gu, Dries Schellekens, and Ingrid Verbauwhede. Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 34(6):889–902, 2015.

[DLM19] Mathieu Dumont, Mathieu Lisart, and Philippe Maurine. Electromagnetic fault injection : How faults occur. In *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 9–16, 2019.

[Dol12] Edin Dolićanin. Gamma ray effects on flash memory cell arrays. *Nuclear Technology and Radiation Protection*, 27:284–289, September 2012.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM journal on computing*, 38(1):97–139, 2008.

[DPDC93] Stuart Denenberg, Robert Petersen, John Densberger, and John J. Christensen. System for registration, identification and verification of items utilizing unique intrinsic features, 1993. US Patent 5521984, filed in 1993, published in 1996.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *IACR Eurocrypt*, pages 523–540. Springer, 2004.

[DV13] Jeroen Delvaux and Ingrid Verbauwhede. Side channel modeling attacks on 65nm arbiter PUFs exploiting CMOS device noise. In *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 137–142. IEEE, 2013.

[DV14a] Jeroen Delvaux and Ingrid Verbauwhede. Attacking PUF-based pattern matching key generators via helper data manipulation. In *Cryptographers' Track at the RSA Conference*, pages 106–131. Springer, 2014.

[DV14b] Jeroen Delvaux and Ingrid Verbauwhede. Fault injection modeling attacks on 65 nm arbiter and RO sum PUFs via environmental changes. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(6):1701–1713, 2014.

[DV14c] Jeroen Delvaux and Ingrid Verbauwhede. Key-recovery attacks on various RO PUF constructions via helper data manipulation. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2014.

[EFK$^+$12] Thomas Esbach, Walter Fumy, Olga Kulikovska, Dominik Merli, Dieter Schuster, and Frederic Stumpf. A new security architecture for smartcards utilizing pufs. In *Information Security Solutions Europe (ISSE 2012)*. Vieweg Verlag, 2012.

[EL] Randall J. Easter and Ken Lu. FIPS 140-3 Section 5 - Physical Security. `https://csrc.nist.gov/CSRC/media/Presentations/FIPS-140-3-Section-5-Physical-Security/images-media/physecpre18.pdf`. Online, accessed March 24, 2021.

[ES05] Halit Eren and Lucas D Sandor. Fringe-Effect Capacitive Proximity Sensors for Tamper Proof Enclosures. In *2005 Sensors for Industry Conference*, pages 22–26, 2005.

[Eur09] European Parliament and the Council of the European Union. *REGULATION (EC) No 595/2009 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 18 June 2009 on type-approval of motor vehicles and engines with respect to emissions from heavy duty vehicles (Euro VI) and on access to vehicle repair and maintenance information and amending Regulation (EC) No 715/2007 and Directive 2007/46/EC and repealing Directives 80/1269/EEC, 2005/55/EC and 2005/78/EC*, June 2009.

[Eur14] European Parliament and the Council of the European Union. *RGULATION (EU) No 165/2014 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 4 February 2014 on tachographs in road transport, repealing Council Regulation (EEC) No 3821/85 on recording equipment in road transport and amending Regulation (EC) No 561/2006 of the European Parliament and of the Council on the harmonisation of certain social legislation relating to road transport*, February 2014.

[Eur16a] European Parliament and the Council of the European Union. *COMMISSION DELEGATED REGULATION (EU) 2017/654 of 19 December 2016 supplementing Regulation (EU) 2016/1628 of the European Parliament and of the Council with regard to technical and general requirements relating to emission limits and type-approval for internal combustion engines for non-road mobile machinery*, December 2016.

[Eur16b] European Parliament and the Council of the European Union. *COMMISSION IMPLEMENTING REGULATION (EU) 2017/656 of 19 December 2016 laying down the administrative requirements relating to emission limits and type-approval of internal combustion engines for non-road mobile machinery in accordance with Regulation (EU) 2016/1628 of the European Parliament and of the Council*, December 2016.

[Fed08] Federal Office for Information Security. Common Criteria Protection Profile – Cryptographic Modules, Security Level "Enhanced" (BSI-CC-PP-0045), 2008.

[Fed11] Federal Office for Information Security, Bonn, Germany. *Evaluation of random number generators – Version 0.10*, September 2011. `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_20_AIS_31_Evaluation_of_random_number_generators_e.pdf`.

[FF06] Markus Fischer and Günther Froschermeier. Electronic safety module, 2006. EU Patent 1804557A1, filed in 2006, published in 2007.

[FIU$^+$18] Elischa Ferres, Vincent Immler, Alexander Utz, Alexander Stanitzki, Reneé Lerch, and Rainer Kokozinski. Capacitive Multi-Channel Security Sensor IC for Tamper-Resistant Enclosures. In *IEEE SENSORS*, pages 1–4, 2018.

[FJLT13] Thomas Fuhr, Eliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on aes with faulty ciphertexts only. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 108–118, 2013.

[FLM10] Jacques J.A. Fournier and Philippe Loubet-Moundi. Memory address scrambling revealed using fault attacks. In *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 30–36, 2010.

[FMMJ21] Tagir Fabarisov, Ilshat Mamaev, Andrey Morozov, and Klaus Janschek. Model-based fault injection experiments for the safety analysis of exoskeleton system, 2021.

[For65a] George David Forney. Concatenated codes - technical report 440. Technical report, Massachusetts Institute of Technology – Research Laboratory of Electronics, December 1965.

[For65b] George David Forney. On decoding BCH codes. *IEEE Transactions on Information Theory*, 11(4):549–557, 1965.

[FP15] Davide Ferraretto and Graziano Pravadelli. Efficient fault injection in QEMU. *2015 16th Latin-American Test Symposium, LATS 2015*, May 2015.

[FP16] Davide Ferraretto and Graziano Pravadelli. Simulation-based Fault Injection with QEMU for Speeding-up Dependability Analysis of Embedded Software. *Journal of Electronic Testing*, 32, February 2016.

[FPV13] Irfan Fetahović, Milić Pejović, and Miloš Vujisić. Radiation Damage in Electronic Memory Devices. *International Journal of Photoenergy*, 2013, 2013.

[Fro23] Frontgrade Gaisler AB. *GR712RC Dual-Core LEON3FT SPARC V8 Processor User's Manual – Version 2.16*, November 2023.

[FSK98] Peter Folkesson, Sven Svensson, and Johan Karlsson. A comparison of simulation based and scan chain implemented fault injection. In *Digest of Papers. Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing (Cat. No.98CB36224)*, pages 284–293, 1998.

[FSMA99] Jean-Charles Fabre, Frédéric Salles, M. Rodríguez Moreno, and Jean Arlat. Assessment of COTS microkernels by fault injection. In *Dependable Computing for Critical Applications 7*, pages 25–44, 1999.

[Gar21a] Gartner. Ausgaben für IT-Sicherheit weltweit im Jahr 2020 nach Marktsegment und Prognose für 2021 (in Milliarden US-Dollar) [Graph]. `https://de-statista-com.eaccess.ub.tum.de/statistik/daten/studie/151720/umfrage/it-sicherheitsumsatz-nach-marktsegmenten-weltweit/`, 2021. In Statista. Online, accessed February 8, 2022.

[Gar21b] Gartner. Ausgaben für IT-Sicherheit weltweit im Jahr 2020 und Prognose für 2021 (in Milliarden US-Dollar) [Graph]. https://de-statista-com.eaccess.ub.tum.de/statistik/daten/studie/1038510/umfrage/ausgaben-fuer-it-sicherheit-weltweit/, 2021. In Statista. Online, accessed February 8, 2022.

[Gar22] Gartner. Prognose zu den weltweiten IT-Ausgaben von 2012 bis 2023 nach Segment (in Milliarden US-Dollar) [Graph]. https://de-statista-com.eaccess.ub.tum.de/statistik/daten/studie/160458/umfrage/weltweite-it-ausgaben-nach-ausgewaehlten-bereichen/, 2022. In Statista. Online, accessed February 8, 2022.

[Gas03] Blaise Laurent Patrick Gassend. *Physical Random Functions.* PhD thesis, Massachusetts Institute of Technology (MIT), MA, USA, 2003.

[GBGG03] Daniel Gil, Juan Carlos Baraza, Joaquín Gracia, and Pedro Joaquín Gil. *VHDL Simulation-Based Fault Injection Techniques*, pages 159–176. Springer US, Boston, MA, 2003.

[GCvDD02] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Silicon Physical Random Functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, page 148–160, New York, NY, USA, 2002. Association for Computing Machinery.

[GCvDD03] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Delay-Based Circuit Authentication and Applications. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, SAC '03, page 294–301, New York, NY, USA, 2003. Association for Computing Machinery.

[GDBF14] Qiang Guan, Nathan DeBardeleben, Sean Blanchard, and Song Fu. F-sefi: A fine-grained soft error fault injection tool for profiling application vulnerability. In *Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS*, May 2014.

[GGB] GGB Industries Inc. Tungsten Probe Tips T-4 series. https://ggb.com/wp-content/uploads/2017/06/t-4.pdf. Online, accessed July 31, 2021.

[Git20] GitHub. Add travis-ci tests for testing sw counter measures against hw level fault injection. https://github.com/mcu-tools/mcuboot/pull/789, 2020. Accessed (Online): May 24, 2021.

[GKS14] Filipe Geissler, Fernanda Kastensmidt, and Jose Souza. Soft error injection methodology based on qemu software platform. In *LATW 2014 - 15th IEEE Latin-American Test Workshop*, pages 1–5, March 2014.

[GKST07] Jorge Guajardo, Sandeep S Kumar, Geert-Jan Schrijen, and Pim Tuyls. FPGA intrinsic PUFs and their use for IP protection. In *International workshop on cryptographic hardware and embedded systems*, pages 63–80. Springer, 2007.

[GKST15] Fatemeh Ganji, Juliane Krämer, Jean-Pierre Seifert, and Shahin Tajik. Lattice basis reduction attack against physically unclonable functions. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1070–1080, 2015.

[GM89] Sidney N. Graybeal and Patricia Bliss McFate. Getting Out of the STARTing Block. *Scientific American*, 261(6):61–67, 1989.

[GN98] Robert M. Gray and David L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44(6):2325–2383, 1998.

[GO20] Kathrin Garb and Johannes Obermaier. Temporary Laser Fault Injection into Flash Memory: Calibration, Enhanced Attacks, and Countermeasures. In *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2020.

[GOFK21] Kathrin Garb, Johannes Obermaier, Elischa Ferres, and Martin König. FORTRESS: FORtified Tamper-Resistant Envelope with Embedded Security Sensor. In *2021 18th International Conference on Privacy, Security and Trust (PST)*, pages 1–12, 2021.

[GOR] W.L. GORE & Associates, Inc., Electronic Products Division, Newark. *GORE$^{\text{TM}}$ Tamper Respondent Surface Enclosure.*

[GRAS$^+$16] Yansong Gao, Damith C. Ranasinghe, Said F. Al-Sarawi, Omid Kavehei, and Derek Abbott. Emerging Physical Unclonable Functions With Nanotechnology. *IEEE Access*, 4:61–80, 2016.

[GSBC$^+$08] Joaquin Gracia, Luis Saiz, Juan-Carlos Baraza-Calvo, Daniel Gil, and Pedro Gil-Vicente. Analysis of the influence of intermittent faults in a microcontroller. In *Proceedings - 2008 IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, DDECS*, pages 1–6, May 2008.

[GSHO21] Kathrin Garb, Marc Schink, Matthias Hiller, and Johannes Obermaier. Attacks and countermeasures for capacitive puf-based security enclosures. In *2021 IEEE Physical Assurance and Inspection of Electronics (PAINE)*, pages 1–8, 2021.

[GTFS16] Fatemeh Ganji, Shahin Tajik, Fabian Fäßler, and Jean-Pierre Seifert. Strong machine learning attack against PUFs with no mathematical model. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 391–411. Springer, 2016.

[GTS15] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. Why attackers win: on the learnability of XOR arbiter PUFs. In *International Conference on Trust and Trustworthy Computing*, pages 22–39. Springer, 2015.

[GTS16] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. PAC learning of arbiter PUFs. *Journal of Cryptographic Engineering*, 6(3):249–258, 2016.

[Gue16] Shay Gueron. A memory encryption engine suitable for general purpose processors. Cryptology ePrint Archive, Paper 2016/204, 2016. `https://eprint.iacr.org/2016/204`.

[GXKF22] Kathrin Garb, Marvin Xhemrishi, Ludwig Kürzinger, and Christoph Frisch. The Wiretap Channel for Capacitive PUF-Based Security Enclosures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(3):165–191, 2022.

[HAP09] Ryan Helinski, Dhruva Acharyya, and Jim Plusquellic. A physical unclonable function defined using power distribution system equivalent resistance variations. In *2009 46th ACM/IEEE Design Automation Conference*, pages 676–681. IEEE, 2009.

[HBNS13] Clemens Helfmeier, Christian Boit, Dmitry Nedospasov, and Jean-Pierre Seifert. Cloning physically unclonable functions. In *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 1–6. IEEE, 2013.

[Hew] Hewlett-Packard Company. Hewlett-Packard — Atalla Security Products Ax160 PCI HSM Security Policy. `https://www.pcisecuritystandards.org/ptsdocs/HP%20Atalla%20Ax160%20PCI%20HSM%20Security%20Policy%201_1.pdf`. Online, accessed November 14, 2021.

[Hew15] Hewlett-Packard Company. *Atalla Cryptographic Subsystem (ACS) Non-Proprietary Security Policy – Version 0.6*, August 2015. `https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp2452.pdf`.

[HGA$^+$21] Florian Hauschild, Kathrin Garb, Lukas Auer, Bodo Selmke, and Johannes Obermaier. ARCHIE: A QEMU-Based Framework for Architecture-Independent Evaluation of Faults. In *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, pages 20–30. IEEE, 2021.

[HHH08] Sun-Mook Hwang, Joo-Il Hong, and S. Huh. Characterization of the susceptibility of integrated circuits with induction caused by high power microwaves. *Progress in Electromagnetics Research-pier - PROG ELECTROMAGN RES*, 81:61–72, January 2008.

[Hil02] Martin Hiller. *A Software Profiling Methodology for Design and Assessment of Dependable Software.* PhD thesis, Chalmers University of Technology, Göteborg, Sweden, 2002.

[Hil16] Matthias Hiller. *Key derivation with physical unclonable functions.* PhD thesis, Technische Universität München, 2016.

[HKB$^+$10] Toshihiro Hanawa, Hitoshi Koizumi, Takayuki Banzai, Mitsuhisa Sato, Shin'ichi Miura, Tadatoshi Ishii, and Hidehisa Takamizawa. Customizing virtual machine with fault injector by integrating with specc device model for a software testing environment d-cloud. In *2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing*, pages 47–54, 2010.

[HKR$^+$15] Andrea Höller, Armin Krieg, Tobias Rauter, Johannes Iber, and Christian Kreiner. QEMU-Based Fault Injection for a System-Level Analysis of Software Countermeasures Against Fault Attacks. In *2015 Euromicro Conference on Digital System Design*, pages 530–533, 2015.

[HKS20] Matthias Hiller, Ludwig Kurzinger, and Georg Sigl. Review of error correction for PUFs and evaluation on state-of-the-art FPGAs. *Journal of Cryptographic Engineering*, 10:229–247, 2020.

[HMR$^+$15] Andrea Höller, Georg Macher, Tobias Rauter, Johannes Iber, and Christian Kreiner. A virtual fault injection framework for reliability-aware software development. In *2015 IEEE International Conference on Dependable Systems and Networks Workshops*, pages 69–74, 2015.

[HMSS12] Matthias Hiller, Dominik Merli, Frederic Stumpf, and Georg Sigl. Complementary IBS: Application specific error correction for PUFs. In *2012 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 1–6. IEEE, 2012.

[HNT$^+$13] Clemens Helfmeier, Dmitry Nedospasov, Christopher Tarnovsky, Jan Starbug Krissler, Christian Boit, and Jean-Pierre Seifert. Breaking and Entering through the Silicon. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 733–744, New York, NY, USA, 2013. Association for Computing Machinery.

[HÖ17] Matthias Hiller and Aysun Gurur Önalan. Hiding secrecy leakage in leaky helper data. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 601–619. Springer, 2017.

[HOSB16] Matthias Hiller, Aysun Gurur Önalan, Georg Sigl, and Martin Bossert. Online Reliability Testing for PUF Key Derivation. In *Proceedings of the 6th International Workshop on Trustworthy Embedded Devices*, pages 15–22, New York, NY, USA, 2016. Association for Computing Machinery.

[HSK$^+$14] Andrea Höller, Gerhard Schönfelder, Nermin Kajtazovic, Tobias Rauter, and Christian Kreiner. Fies: A fault injection framework for the evaluation of self-tests for cots-based safety-critical systems. In *2014 15th International Microprocessor Test and Verification Workshop*, pages 105–110, 2014.

[HSP21] Max Hoffmann, Falk Schellenberg, and Christof Paar. ARMORY: Fully Automated and Exhaustive Fault Simulation on ARM-M Binaries. *IEEE Transactions on Information Forensics and Security*, 16:1058–1073, 2021.

[Hub92] Gerardus Tarcisius Maria Hubert. Device with protection against access to secure information, 1992. EU Patent 509567A2, filed in 1992, published in 1992.

[HWGH18] Robert Hesselbarth, Florian Wilde, Chongyan Gu, and Neil Hanley. Large scale RO PUF analysis over slice type, evaluation time and temperature on 28nm Xilinx FPGAs. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 126–133. IEEE, 2018.

[HWRL$^+$13] Matthias Hiller, Michael Weiner, Leandro Rodrigues Lima, Maximilian Birkner, and Georg Sigl. Breaking through Fixed PUF Block Limitations with Differential Sequence Coding and Convolutional Codes. In *Proceedings of the 3rd International Workshop on Trustworthy Embedded Devices*, TrustED '13, pages 43–54, New York, NY, USA, 2013. Association for Computing Machinery.

[HYKS10] Yohei Hori, Takahiro Yoshida, Toshihiro Katashita, and Akashi Satoh. Quantitative and statistical performance evaluation of arbiter physical unclonable functions on FPGAs. In *2010 International conference on reconfigurable computing and FPGAs*, pages 298–303. IEEE, 2010.

[HYP15] Matthias Hiller, Meng-Day Yu, and Michael Pehl. Systematic Low Leakage Coding for Physical Unclonable Functions. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 155–166, New York, NY, USA, 2015. Association for Computing Machinery.

[HZJ17] Mahadi Hasan, Jingwei Zhao, and Zhengyi Jiang. A review of modern advancements in micro drilling techniques. *Journal of Manufacturing Processes*, 29:343–375, 2017.

[IFS$^+$20] Hidenori Iwashita, Gentaro Funatsu, Hirotaka Sato, Takashi Kamiyama, Michihiro Furusaka, Stephen A. Wender, Eric Pitcher, and Yoshiaki Kiyanagi. Energy-Resolved Soft-Error Rate Measurements for 1–800 MeV Neutrons by the Time-of-Flight Technique at LANSCE. *IEEE Transactions on Nuclear Science*, 67(11):2363–2369, 2020.

[IHKS16] Vincent Immler, Maxim Hennig, Ludwig Kürzinger, and Georg Sigl. Practical aspects of quantization and tamper-sensitivity for physically obfuscated keys. In *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems*, CS2 '16, pages 13–18, New York, NY, USA, 2016. Association for Computing Machinery.

[IHL$^+$19] Vincent Immler, Matthias Hiller, Qinzhi Liu, Andreas Lenz, and Antonia Wachter-Zeh. Variable-length bit mapping and error-correcting codes for higher-order alphabet pufs—extended version. *Journal of Hardware and Systems Security*, 3(1):78–93, 2019.

[IHOS17] Vincent Immler, Matthias Hiller, Johannes Obermaier, and Georg Sigl. Take a moment and have some t: Hypothesis testing on raw PUF data. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 128–129. IEEE, 2017.

[IMJFC13] Phil Isaacs, Thomas Morris Jr, Michael J Fisher, and Keith Cuthbert. Tamper proof, tamper evident encryption technology. In *Pan Pacific Symposium*, 2013.

[Imm19] Vincent Charles Immler. *Higher-Order Alphabet Physical Unclonable Functions.* PhD thesis, Technische Universität München, 2019.

[Inf21] Infineon Technologies AG. *Application Note AN98527 Rev. D – X-Ray Inspection Test Conditions for NOR/SPI/NAND Flash*, March 2021.

[Int12] International Organization for Standardization (ISO), Geneva, Switzerland. *ISO/IEC 19790:2012 Information technology – Security techniques – Test requirements for cryptographic modules*, August 2012.

[Int15] International Organization for Standardization (ISO), Geneva, Switzerland. *ISO/IEC TS 30104:2015 Information Technology — Security Techniques — Physical Security Attacks, Mitigation Techniques and Security Requirements*, May 2015.

[Int17] International Organization for Standardization (ISO), Geneva, Switzerland. *ISO/IEC 24759:2017 Information technology – Security techniques – Test requirements for cryptographic modules*, March 2017.

[Int19] International Electrochemical Commission (IEC), Geneva, Switzerland. *IEC 62443-4-2:2019 Security for industrial automation and control system - Part 4-2: Technical security requirements for IACS components*, February 2019.

[Int20] International Organization for Standardization (ISO), Geneva, Switzerland. *ISO/IEC 20897-1:2020 Information security, cybersecurity and privacy protection - Physically unclonable functions - Part 1: Security requirements*, December 2020.

[Int22a] International Organization for Standardization (ISO), Geneva, Switzerland. *ISO/IEC 15408-1: Information security, cybersecurity and privacy protection – Evaluation criteria for IT security – Part 1: Introduction and general model*, August 2022.

[Int22b] International Organization for Standardization (ISO), Geneva, Switzerland. *ISO/IEC 15408-2: Information security, cybersecurity and privacy protection – Evaluation criteria for IT security – Part 2: Security functional components*, August 2022.

[Int22c] International Organization for Standardization (ISO), Geneva, Switzerland. *ISO/IEC 15408-3: Information security, cybersecurity and privacy protection – Evaluation criteria for IT security – Part 3: Security assurance components*, August 2022.

[Int22d] International Organization for Standardization (ISO), Geneva, Switzerland. *ISO/IEC 15408-4: Information security, cybersecurity and privacy protection – Evaluation criteria for IT security – Part 4: Framework for the specification of evaluation methods and activities*, August 2022.

[Int22e] International Organization for Standardization (ISO), Geneva, Switzerland. *ISO/IEC 15408-5: Information security, cybersecurity and privacy protection – Evaluation criteria for IT security – Part 5: Pre-defined packages of security requirements*, August 2022.

[Int22f] International Organization for Standardization (ISO), Geneva, Switzerland. *ISO/IEC 18045: Information security, cybersecurity and privacy protection – Evaluation criteria for IT security – Methodology for IT security evaluation*, August 2022.

[Int22g] International Organization for Standardization (ISO), Geneva, Switzerland. *ISO/IEC 20897-2:2022 Information security, cybersecurity and privacy protection - Physically unclonable functions - Part 2: Test and evaluation methods*, May 2022.

[IOK$^+$18] Vincent Immler, Johannes Obermaier, Martin König, Matthias Hiller, and Georg Sigl. B-TREPID: Batteryless tamper-resistant envelope with a PUF and integrity detection. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 49–56. IEEE, 2018.

[ION$^+$19] Vincent Immler, Johannes Obermaier, Kuan Kuan Ng, Fei Xiang Ke, JinYu Lee, Yak Peng Lim, Wei Koon Oh, Keng Hoong Wee, and Georg Sigl. Secure Physical Enclosures from Covers with Tamper-Resistance. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 51–96, 2019.

[IU19] Vincent Immler and Karthik Uppund. New Insights to Key Derivation for Tamper-Evident Physical Unclonable Functions. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 30–65, 2019.

[JAR$^+$94] Eric Jenn, Jean Arlat, Marcus Rimen, Joakim Ohlsson, and Johan Karlsson. Fault injection into VHDL models: the MEFISTO tool. In *Proceedings of IEEE 24th International Symposium on Fault- Tolerant Computing*, pages 66–75, 1994.

[JH11] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.

[Jil98] David Jiles. *Introduction to Magnetism and Magnetic Materials.* Chapman & Hall, London, 1998.

[JIW22] Joint Interpretation Working Group (JIWG). *Joint Interpretation Library - Application of Attack Potential to Smartcards – Version 3.2*, November 2022.

[JKNM19] Hideyuki Jitsumoto, Yuya Kobayashi, Akihiro Nomura, and Satoshi Matsuoka. MH-QEMU: Memory-State-Aware Fault Injection Platform. In David Abramson and Bronis R. de Supinski, editors, *Supercomputing Frontiers*, pages 71–85, Cham, 2019. Springer International Publishing.

[JL12] Myeongwoon Jeon and Jungwoo Lee. On codes correcting bidirectional limited-magnitude errors for flash memories. In *2012 International Symposium on Information Theory and its Applications*, pages 96–100, 2012.

[JW99] Ari Juels and Martin Wattenberg. A fuzzy commitment scheme. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, CCS '99, page 28–36, New York, NY, USA, 1999. Association for Computing Machinery.

[KBB$^+$19] Dilip S. V. Kumar, Arthur Beckers, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. An In-Depth and Black-Box Characterization of the Effects of Laser Pulses on ATmega328P. In *Smart Card Research and Advanced Applications*, pages 156–170. Springer, 2019.

[KBBT15] Maha Kooli, Alberto Bosio, Pascal Benoit, and Lionel Torres. Software testing and software fault injection. In *2015 10th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6, 2015.

[KDN14] Maha Kooli and Giorgio Di Natale. A survey on simulation-based fault injection tools for complex systems. In *2014 9th IEEE International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6, 2014.

[KFPW22] Claus Kestel, Christoph Frisch, Michael Pehl, and Norbert Wehn. Towards more secure puf applications: A low-area polar decoder implementation. In *2022 IEEE 35th International System-on-Chip Conference (SOCC)*, pages 1–6, 2022.

[KHEB14] Thomas Korak, Michael Hutter, Baris Ege, and Lejla Batina. Clock Glitch Attacks in the Presence of Heating. In *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 104–114, 2014.

[KKA95] Ghani A. Kanawati, Nasser A. Kanawati, and Jacob A. Abraham. FERRARI: a flexible software-based fault and error injection system. *IEEE Transactions on Computers*, 44(2):248–260, 1995.

[KKT04] Mark Karpovsky, Konrad J Kulikowski, and Alexander Taubin. Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard. In *International Conference on Dependable Systems and Networks, 2004*, pages 93–101. IEEE, 2004.

[KKT07] Konrad Kulikowski, Mark Karpovsky, and Alexander Taubin. Robust codes and robust, fault-tolerant architectures of the Advanced Encryption Standard. *Journal of Systems Architecture*, 53:139–149, February 2007.

[kme20] kmeans1d 0.3.1 - a python package for optimal 1d k-means clustering. `https://pypi.org/project/kmeans1d/`, 2020. Online, published on August 24, 2020.

[KS10] Deniz Karakoyunlu and Berk Sunar. Differential template attacks on PUF enabled cryptographic devices. In *2010 IEEE International Workshop on Information Forensics and Security*, pages 1–6, 2010.

[KS11] Wolfgang Killmann and Werner Schindler. *A proposal for: Functionality classes for random number generators.* Federal Office for Information Security, Bonn, Germany, September 2011.

[KWMK02] Ramesh Karri, Kaijie Wu, Piyush Mishra, and Yongkook Kim. Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(12):1509–1517, 2002.

[LaB04] Kenneth A. LaBel. Radiation Effects on Electronics 101: Simple Concepts and New Challenges. `https://nepp.nasa.gov/DocUploads/392333B0-7A48-4A04-A3A72B0B1DD73343/Rad_Effects_101_WebEx.pdf`, 2004. National Aeronautics and Space Administration. Online, accessed April 15, 2022.

[Lav08] Kevin Lavery. *Application Report SPNA109 – Discriminating Between Soft Errors and Hard Errors in RAM.* Texas Instruments, Dallas, Texas, July 2008.

[LCNR02] Paul Arthur Layman, Samir Chaudhry, James Gary Norman, and Thomson J. Ross. Electronic fingerprinting of semiconductor circuits, 2002. US Patent 6738294, filed in 2002, published in 2004.

[LDT00] Kevin Lofstrom, W. Robert Daasch, and Donald Taylor. IC identification circuit using device mismatch. In *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.00CH37056)*, pages 372–373, 2000.

[LLC09] PCI Security Standards Council LLC. Payment Card Industry (PCI) - Hardware Security Module - Security Requirements - Version 1.0. `https://www.pcisecuritystandards.org/documents/PCI%20HSM%20Security%20Requirements%20v1.0%20final.pdf`, 2009. Online, accessed February 22, 2022.

[LLG$^+$04] Jae W Lee, Daihyun Lim, Blaise Gassend, Gookwon Edward Suh, Marten Van Dijk, and Srinivas Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No. 04CH37525)*, pages 176–179. IEEE, 2004.

[LLG$^+$05] Daihyun Lim, Jae W Lee, Blaise Gassend, Gookwon Edward Suh, Marten Van Dijk, and Srinivas Devadas. Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(10):1200–1205, 2005.

[LT15] Michael Le and Yuval Tamir. Fault injection in virtualized systems—challenges and applications. *IEEE Transactions on Dependable and Secure Computing*, 12(3):284–297, 2015.

[LVY12] Dong Li, Jeffrey S. Vetter, and Weikuan Yu. Classifying soft error vulnerabilities in extreme-scale scientific applications using a binary instrumentation tool. In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2012.

[LXW13] Yi Li, Ping Xu, and Han Wan. A Fault Injection System Based on QEMU Simulator and Designed for BIT Software Testing. *Applied Mechanics and Materials*, 347-350, February 2013.

[Mac90] Hugh MacPherson. Security enclosures, 1990. US Patent 4972175, filed in 1989, published in 1990.

[Mac93] Hugh MacPherson. Security enclosure manufacture, 1993. US Patent 5539379, filed in 1993, published in 1996.

[Mae12] Roel Maes. *Physically unclonable functions: Constructions, properties and applications.* PhD thesis, Katholieke Universiteit Leuven, Belgium, 2012.

[Mas69] James Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, 1969.

[MB17] Sven Mueelich and Martin Bossert. A New Error Correction Scheme for Physical Unclonable Functions. In *SCC 2017; 11th International ITG Conference on Systems, Communications and Coding*, pages 1–6, 2017.

[MBC16] Cédric Marchand, Lilian Bossuet, and Abdelkarim Cherkaoui. Design and characterization of the TERO-PUF on SRAM FPGAs. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 134–139. IEEE, 2016.

[MCMS10] Abhranil Maiti, Jeff Casarona, Luke McHale, and Patrick Schaumont. A large scale characterization of RO-PUF. In *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 94–99. IEEE, 2010.

[MGDW$^+$18] Daniel Mueller-Gritschneder, Martin Dittrich, Josef Weinzierl, Eric Cheng, Subhasish Mitra, and Ulf Schlichtmann. ETISS-ML: A multi-level instruction set simulator with RTL-level fault injection support for the evaluation of cross-layer resiliency techniques. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 609–612, 2018.

[MGS13] Abhranil Maiti, Vikash Gunreddy, and Patrick Schaumont. A systematic method to evaluate and compare the performance of physical unclonable functions. In *Embedded systems design with FPGAs*, pages 245–267. Springer, 2013.

[MHK$^+$19] Holger Mandry, Andreas Herkle, Ludwig Kürzinger, Sven Müelich, Joachim Becker, Robert F.H. Fischer, and Maurits Ortmanns. Modular PUF Coding Chain with High-Speed Reed-Muller Decoder. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2019.

[Mic] The Micromanipulator Co. *7X Tungsten Cat Whisker Fine Probe Tip*. `https://micromanipulator.com/products/accessories/probe-tips/7x/`. Online, accessed July 28, 2021.

[MIS13] The MISRA Consortium Limited. *MISRA C:2012 - Guidelines for the use of the C language in critical systems*, March 2013.

[MIT22] MITRE. Common Weakness Enumeration - A Community-Developed List of Software & Hardware Weakness Types. `https://cwe.mitre.org/`, 2022. Accessed (Online): May 25, 2022.

[MKP08] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. Testing techniques for hardware security. In *2008 IEEE International Test Conference*, pages 1–10. IEEE, 2008.

[MKRM10] Mehran Mozaffari-Kermani and Arash Reyhani-Masoleh. Concurrent structure-independent fault detection schemes for the advanced encryption standard. *IEEE Transactions on Computers*, 59(5):608–622, 2010.

[MKRM11] Mehran Mozaffari-Kermani and Arash Reyhani-Masoleh. Reliable Hardware Architectures for the Third-Round SHA-3 Finalist Grostl Benchmarked on FPGA Platform. In *2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pages 325–331, 2011.

[ML08] Paolo Maistri and Régis Leveugle. Double-Data-Rate Computation as a Countermeasure against Fault Analysis. *IEEE Transactions on Computers*, 57(11):1528–1539, 2008.

[MLSW16] Roel Maes, Vincent Leest, Erik Sluis, and Frans Willems. Secure key generation from biased pufs: extended version. *Journal of Cryptographic Engineering*, 6, June 2016.

[MOG+20] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against intel sgx. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1466–1482, 2020.

[MPB18] Sven Müelich, Sven Puchinger, and Martin Bossert. Using Convolutional Codes for Key Extraction in SRAM Physical Unclonable Functions. *arXiv:1704.01306*, 2018.

[MR00] Eliane Martins and Amanda C.A. Rosa. A fault injection approach based on reflective programming. In *Proceeding International Conference on Dependable Systems and Networks. DSN 2000*, pages 407–416, 2000.

[MRL02] Eliane Martins, Cecilia MF Rubira, and Nelson G.M. Leme. Jaca: a reflective fault injection tool based on patterns. In *Proceedings International Conference on Dependable Systems and Networks*, pages 483–487. IEEE, 2002.

[MRMK13] Ahmed Mahmoud, Ulrich Rührmair, Mehrdad Majzoobi, and Farinaz Koushanfar. Combined modeling and side channel attacks on strong PUFs. *Cryptology ePrint Archive*, 2013.

[MRMS01] Henrique Madeira, Mrio Rela, Francisco Moreira, and João Silva. RIFLE: A general purpose pin-level fault injector. *European Dependable Computing Conference*, March 2001.

[MSS13] Dominik Merli, Frederic Stumpf, and Georg Sigl. Protecting PUF error correction by codeword masking. *IACR Cryptology ePrint Archive*, 334, 2013.

[MSSS11a] Dominik Merli, Dieter Schuster, Frederic Stumpf, and Georg Sigl. Semi-invasive em attack on fpga ro pufs and countermeasures. In *Proceedings of the Workshop on Embedded Systems Security (WESS)*, pages 1–9. ACM, March 2011.

[MSSS11b] Dominik Merli, Dieter Schuster, Frederic Stumpf, and Georg Sigl. Side-channel analysis of PUFs and fuzzy extractors. In *TRUST*, number 6740 in Lecture Notes in Computer Science, pages 33–47. Springer Berlin Heidelberg, 2011.

[MSY06] Tal Malkin, François-Xavier Standaert, and Moti Yung. A Comparative Cost/Security Analysis of Fault Attack Countermeasures. In *Fault Diagnosis and Tolerance in Cryptography*, pages 159–172, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[MTV08] Roel Maes, Pim Tuyls, and Ingrid Verbauwhede. Intrinsic PUFs from flip-flops on reconfigurable devices. In *3rd Benelux workshop on information and system security (WISSec 2008)*, volume 17, page 2008, 2008.

[MTV09a] Roel Maes, Pim Tuyls, and Ingrid Verbauwhede. A soft decision helper data algorithm for SRAM PUFs. In *2009 IEEE international symposium on information theory*, pages 2101–2105. IEEE, 2009.

[MTV09b] Roel Maes, Pim Tuyls, and Ingrid Verbauwhede. Low-overhead implementation of a soft decision helper data algorithm for sram pufs. In *International workshop on cryptographic hardware and embedded systems*, pages 332–347. Springer, 2009.

[MV11] Hessam Mahdavifar and Alexander Vardy. Achieving the Secrecy Capacity of Wiretap Channels Using Polar Codes. *IEEE Transactions on Information Theory*, 57(10):6428–6443, 2011.

[MVHV12] Roel Maes, Anthony Van Herrewege, and Ingrid Verbauwhede. PUFKY: A Fully Functional PUF-Based Cryptographic Key Generator. In Emmanuel Prouff and Patrick Schaumont, editors, *Proceedings of the 14th International Conference on Cryptographic Hardware and Embedded Systems*, pages 302–319, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MW79] Timothy C. May and Murray H. Woods. Alpha-particle-induced soft errors in dynamic memories. *IEEE Transactions on Electron Devices*, 26(1):2–9, 1979.

[Nat02] National Institute of Standards and Technology (NIST), Gaithersburg, MD. *FIPS PUB 140-2 – Security Requirements for Cryptographic Modules*, March 2002.

[Nat10] National Institute of Standards and Technology (NIST), Gaithersburg, MD. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, April 2010.

[Nat19] National Institute of Standards and Technology (NIST), Gaithersburg, MD. *FIPS PUB 140-3 – Security Requirements for Cryptographic Modules*, March 2019.

[Nat20] National Institute of Standards and Technology (NIST), Gaithersburg, MD. *NIST Special Publication 800-53 – Rev. 5 – Security and Privacy Controls for Information Systems and Organizations*, September 2020.

[NCDM13] Roberto Natella, Domenico Cotroneo, Joao A. Duraes, and Henrique S. Madeira. On fault representativeness of software fault injection. *IEEE Transactions on Software Engineering*, 39(1):80–96, 2013.

[NIS22a] Stopping-power & range tables for electrons, protons, and helium ions. `https://www.nist.gov/pml/stopping-power-range-tables-electrons-protons-and-helium-ions`, 2022. Accessed (Online): April 17, 2022.

[NIS22b] X-ray mass attenuation coefficients. `https://www.nist.gov/pml/x-ray-mass-attenuation-coefficients`, 2022. Accessed (Online): April 17, 2022.

[Noh11] Frank Noha. *Application Report SPNA126 – ECC Handling in TMSx70-Based Microcontrollers.* Texas Instruments Incorporated, February 2011.

[NSCM15] Phuong Ha Nguyen, Durga Prasad Sahoo, Rajat Subhra Chakraborty, and Debdeep Mukhopadhyay. Efficient attacks on robust ring oscillator puf with enhanced challenge-response set. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 641–646. IEEE, 2015.

[NSHB13] Dmitry Nedospasov, Jean-Pierre Seifert, Clemens Helfmeier, and Christian Boit. Invasive PUF analysis. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 30–38. IEEE, 2013.

[NSJ$^+$18] Phuong Ha Nguyen, Durga Prasad Sahoo, Chenglu Jin, Kaleel Mahmood, Ulrich Rührmair, and Marten van Dijk. The interpose PUF: Secure PUF design against state-of-the-art machine learning attacks. *Cryptology ePrint Archive*, 2018.

[Nyb18] Ralph Heinz-Erik Nyberg. *New Techniques for Emulating Fault Attacks*. PhD thesis, Technical University of Munich, Munich, Germany, 2018.

[Obe19] Johannes Obermaier. *Breaking and Restoring Embedded System Security*. PhD thesis, Technische Universität München, 2019.

[OHHS18] Johannes Obermaier, Florian Hauschild, Matthias Hiller, and Georg Sigl. An embedded key management system for PUF-based security enclosures. In *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–6. IEEE, 2018.

[OI18] Johannes Obermaier and Vincent Immler. The Past, Present, and Future of Physical Security Enclosures: From Battery-Backed Monitoring to PUF-Based Inherent Security and Beyond. *Journal of Hardware and Systems Security*, 2(4):289–296, 2018.

[OIHS18] Johannes Obermaier, Vincent Immler, Matthias Hiller, and Georg Sigl. A Measurement System for Capacitive PUF-Based Security Enclosures. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, New York, NY, USA, 2018. Association for Computing Machinery.

[OT17] Johannes Obermaier and Stefan Tatschner. Shedding too much Light on a Microcontroller's Firmware Protection. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. USENIX Association, 2017.

[OW85] Lawrence H. Ozarow and Aaron D. Wyner. Wire-Tap Channel II. In *Advances in Cryptology*, pages 33–50, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[Pap01] Ravikanth Pappu. *Physical One-Way Functions*. PhD thesis, Massachusetts Institute of Technology (MIT), MA, USA, 2001.

[PB12] Woomyoung Park and Alexander Barg. Polar codes for q-ary channels, $q = 2^r$. In *2012 IEEE International Symposium on Information Theory Proceedings*, pages 2142–2146, 2012.

[PBR17] Roberta Piscitelli, Shivam Bhasin, and Francesco Regazzoni. *Fault Attacks, Injection Techniques and Tools for Simulation*, pages 27–47. Springer International Publishing, Cham, 2017.

[PMB$^+$15] Sven Puchinger, Sven Mueelich, Martin Bossert, Matthias Hiller, and Georg Sigl. On Error Correction for Physical Unclonable Functions. In *SCC 2015; 10th International ITG Conference on Systems, Communications and Coding*, pages 1–6, 2015.

[PRTG02] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical one-way functions. *Science*, 297(5589):2026–2030, 2002.

[PSC07] Stefan Potyra, Volkmar Sieh, and M Dal Cin. Evaluating Fault-Tolerant System Designs Using FAUmachine. In *Proceedings of the 2007 Workshop on Engineering Fault Tolerant Systems*, EFTS '07, page 9–es, New York, NY, USA, 2007. Association for Computing Machinery.

[PZRB19] Jingyu Pan, Fan Zhang, Kui Ren, and Shivam Bhasin. One fault is all it needs: Breaking higher-order masking with persistent fault analysis. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, 2019.

[Ram21] Rambus / atsec information security corporation, San Jose, California. *VaultIP - FIPS 140-2 Non-Proprietary Security Policy – Document Revision: E*, May 2021.

[Rau] Stewart E. Rauch. Terrestrial radiation induced soft errors in integrated circuits. `https://eds.ieee.org/images/files/Education/terrestrial_radiation_induced_soft_errors_in_integrated_circuits.pdf`. Online, accessed April 15, 2022.

[RBMK10] Jeyavijayan Rajendran, Hetal Borad, Shyam Mantravadi, and Ramesh Karri. SLICED: Slide-based concurrent error detection technique for symmetric block ciphers. In *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 70 – 75, July 2010.

[RD12] Ulrich Rührmair and Marten van Dijk. Practical security analysis of PUF-based two-player protocols. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 251–267. Springer, 2012.

[RGP22] Jonas Ruchti, Michael Gruber, and Michael Pehl. When the Decoder Has to Look Twice: Glitching a PUF Error Correction. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(3):26–70, June 2022.

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

[RS60] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[RSDT13] Cyril Roscian, Alexandre Sarafianos, Jean-Max Dutertre, and Assia Tria. Fault Model Analysis of Laser-Induced Faults in SRAM Memory Cells. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 89–98. IEEE, 2013.

[RSS$^+$10] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 237–249, 2010.

[RSS$^+$13] Ulrich Rührmair, Jan Sölter, Frank Sehnke, Xiaolin Xu, Ahmed Mahmoud, Vera Stoyanova, Gideon Dror, Jürgen Schmidhuber, Wayne Burleson, and Srinivas Devadas. PUF modeling attacks on simulated and silicon data. *IEEE transactions on information forensics and security*, 8(11):1876–1891, 2013.

[RXS$^+$14] Ulrich Rührmair, Xiaolin Xu, Jan Sölter, Ahmed Mahmoud, Mehrdad Majzoobi, Farinaz Koushanfar, and Wayne Burleson. Efficient power and timing side channels for physical unclonable functions. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 476–492. Springer, 2014.

[Sam86] Johan Samyn. Method and apparatus for checking the authenticity of documents, 1986. US Patent 4820912, filed in 1986, published in 1989.

[SB02] Volkmar Sieh and Kerstin Buchacker. Umlinux - a versatile swifi tool. In *Proceedings of the 4th European Dependable Computing Conference on Dependable Computing*, EDCC-4, page 159–171, Berlin, Heidelberg, 2002. Springer-Verlag.

[SBHS16] Bodo Selmke, Stefan Brummer, Johann Heyszl, and Georg Sigl. Precise Laser Fault Injections into 90 nm and 45 nm SRAM-cells. In Naofumi Homma and Marcel Medwed, editors, *Smart Card Research and Advanced Applications*, pages 193–205. Springer, 2016.

[Sch97] Dieter K. Schroder. Carrier lifetimes in silicon. *IEEE Transactions on Electron Devices*, 44(1):160–170, 1997.

[SD07] Gookwon Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *2007 44th ACM/IEEE Design Automation Conference*, pages 9–14. IEEE, 2007.

[Sea08] Robert C. Seacord. *The CERT C Secure Coding Standard*. Addison-Wesley Professional, 2008.

[SEC09] Standards for efficient cryptography - sec 1: Elliptic curve cryptography. `http://www.secg.org/sec1-v2.pdf`, 2009. Online, published on May 21, 2009.

[SEC10] Standards for efficient cryptography - sec 2: Recommended elliptic curve domain parameters - version 2.0. `www.secg.org/sec2-v2.pdf`, 2010. Online, published on January 27, 2010.

[SF19] Mitsuru Shiozaki and Takeshi Fujino. Simple electromagnetic analysis attacks based on geometric leak on an asic implementation of ring-oscillator puf. In *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*, ASHES'19, pages 13–21, New York, NY, USA, 2019. ACM.

[SFIC14] Merrielle Spain, Benjamin Fuller, Kyle Ingols, and Robert Cunningham. Robust keys from physical unclonable functions. In *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 88–92. IEEE, 2014.

[SGC$^+$21] Yang Su, Yansong Gao, Michael Chesser, Omid Kavehei, Alanson Sample, and Damith C. Ranasinghe. SecuCode: Intrinsic PUF Entangled Secure Wireless Code Dissemination for Computational RFID Devices. *IEEE Transactions on Dependable and Secure Computing*, 18(4):1699–1717, 2021.

[SH07] Jörn-Marc Schmidt and Michael Hutter. Optical and em fault-attacks on crt-based rsa: Concrete results. In *Austrochip 2007, 15th Austrian Workhop on Microelectronics, 11 October 2007, Graz, Austria, Proceedings*, pages 61–67. Verlag der Technischen Universität Graz, 2007. Austrochip 2007 ; Conference date: 11-10-2007 Through 11-10-2007.

[SH14] Dieter Schuster and Robert Hesselbarth. Evaluation of bistable ring PUFs using single layer neural networks. In *International Conference on Trust and Trustworthy Computing*, pages 101–109. Springer, 2014.

[SHO07] Ying Su, Jeremy Holleman, and Brian Otis. A 1.6 pJ/bit 96% stable chip-ID generating circuit using process variations. In *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, pages 406–611. IEEE, 2007.

[SJB$^+$18] Sayandeep Saha, Dirmanto Jap, Jakub Breier, Shivam Bhasin, Debdeep Mukhopadhyay, and Pallab Dasgupta. Breaking redundancy-based countermeasures with random faults and power side channel. In *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 15–22, 2018.

[Sko10] Sergei Skorobogatov. Flash Memory 'Bumping' Attacks. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems*, pages 158–172. Springer, 2010.

[Sko12] Sergei Skorobogatov. *Physical Attacks and Tamper Resistance*, pages 143–173. Springer New York, New York, NY, 2012.

[SMRC09] Dhiman Saha, Debdeep Mukhopadhyay, and Dipanwita Roy Chowdhury. A Diagonal Fault Attack on the Advanced Encryption Standard. *IACR Cryptology ePrint Archive*, 2009:581, 2009.

[SNMC15] Durga Prasad Sahoo, Phuong Ha Nguyen, Debdeep Mukhopadhyay, and Rajat Subhra Chakraborty. A case of lightweight PUF constructions: Cryptanalysis and machine learning attacks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(8):1334–1343, 2015.

[SO19] Marc Schink and Johannes Obermaier. Taking a Look into Execute-Only Memory. In *13th USENIX Workshop on Offensive Technologies (WOOT 19)*. USENIX Association, 2019.

[SOD07] Gookwon Edward Suh, Charles W. O'Donnell, and Srinivas Devadas. Aegis: A single-chip secure processor. *IEEE Design & Test of Computers*, 24(6):570–580, 2007.

[SOSD05] Gookwon Edward Suh, Charles W. O'Donnell, Ishan Sachdev, and Srinivas Devadas. Design and implementation of the AEGIS single-chip secure processor using physical random functions. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 25–36. IEEE, 2005.

[SS99] Joshua R. Smith and Andrew V. Sutherland. Microstructure Based Indicia. In *Proceedings of the Second Workshop on Automatic Identification Advanced Technologies*, pages 79–83, 1999.

[SSF+16] Anju Sharma, Preeth Sivakumar, Andrew Feigel, In Tae Bae, Lawrence P. Lehman, Joseph Gregor, James Cash, and Joseph Kolly. Effects of x-ray exposure on NOR and NAND flash memories during high-resolution 2D and 3D x-ray inspection. *International Symposium on Microelectronics*, 2016(1):000660–000665, October 2016.

[SSHA08] Akashi Satoh, Takeshi Sugawara, Naofumi Homma, and Takafumi Aoki. High-Performance Concurrent Error Detection Scheme for AES Hardware. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, pages 100–112, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[SSK12] Koichi Shimizu, Daisuke Suzuki, and Tomomi Kasuya. Glitch PUF: extracting information from usually unwanted glitches. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 95(1):223–233, 2012.

[STB97] Volkmar Sieh, Oliver Tschache, and Frank Balbach. VERIFY: evaluation of reliability using VHDL-models with embedded fault descriptions. In *Proceedings of IEEE 27th International Symposium on Fault Tolerant Computing*, pages 32–36. IEEE, 1997.

[STM19] STMicroelectronics, United Kingdom. *DS13039 Datasheet Rev 1 – STSAFE-A110 – Authentication, state-of-the-art security for peripherals and IoT devices*, December 2019.

[STM22a] STMicroelectronics. *AN5342 Application note Rev 3 – Error correction code (ECC) management for internal memories protection on STM32H7 Series*, March 2022.

[STM22b] STMicroelectronics. *RM0091 Reference Manual Rev 10 – STM32F0x1/ STM32F0x2/STM32F0x8 advanced Arm-based 32-bit MCUs*, May 2022.

[Sto03] Norbert Stolte. *Recursive Codes with the Plotkin-Construction and Their Decoding.* PhD thesis, Technische Universität Darmstadt, 2003.

[ŠTO05] Boris Škorić, Pim Tuyls, and Wil Ophey. Robust key extraction from physical uncloneable functions. In *International Conference on Applied Cryptography and Network Security*, pages 407–422. Springer, 2005.

[STZP21] Paul Staat, Johannes Tobisch, Christian Zenger, and Christof Paar. Anti-Tamper Radio: System-Level Tamper Detection for Computing Systems. *arXiv preprint arXiv:2112.09014*, 2021.

[SVET10] Rickard Svenningsson, Jonny Vinter, Henrik Eriksson, and Martin Törngren. MODIFI: A MODel-Implemented Fault Injection Tool. In Erwin Schoitsch, editor, *Computer Safety, Reliability, and Security*, pages 210–222, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[SVS$^+$88] Zary Segall, D. Vrsalovic, D. Siewiorek, D. Ysskin, J. Kownacki, J. Barton, R. Dancey, A. Robinson, and T. Lin. FIAT-fault injection based automated testing environment. In *[1988] The Eighteenth International Symposium on Fault-Tolerant Computing. Digest of Papers*, pages 102–107, 1988.

[SZK$^+$18] Bodo Selmke, Kilian Zinnecker, Philipp Koppermann, Katja Miller, Johann Heyszl, and Georg Sigl. Locked out by Latch-up? An Empirical Study on Laser Fault Injection into Arm Cortex-M Processors. In *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 7–14. IEEE, 2018.

[TB15] Johannes Tobisch and Georg T. Becker. On the scaling of machine learning attacks on PUFs with application to noise bifurcation. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 17–31. Springer, 2015.

[TDF$^+$14] Shahin Tajik, Enrico Dietz, Sven Frohmann, Jean-Pierre Seifert, Dmitry Nedospasov, Clemens Helfmeier, Christian Boit, and Helmar Dittrich. Physical Characterization of Arbiter PUFs. In *Proceedings of the 16th International Workshop on Cryptographic Hardware and Embedded Systems — CHES 2014 - Volume 8731*, pages 493–509. Springer Berlin Heidelberg, 2014.

[TDF+17] Shahin Tajik, Enrico Dietz, Sven Frohmann, Helmar Dittrich, Dmitry Nedospasov, Clemens Helfmeier, Jean-Pierre Seifert, Christian Boit, and Heinz-Wilhelm Hübers. Photonic side-channel analysis of arbiter PUFs. *Journal of Cryptology*, 30(2):550–571, 2017.

[TDP20] Lars Tebelmann, Jean-Luc Danger, and Michael Pehl. Self-secured PUF: Protecting the Loop PUF by Masking. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 293–314. Springer International Publishing, 2020.

[TG] Thales Group. *High Assurance Hardware Security Modules.* `https://cpl.thalesgroup.com/encryption/hardware-security-modules/network-hsms`. Online, accessed on January 16, 2022.

[TI95] Timothy K. Tsai and Ravishankar K. Iyer. Measuring Fault Tolerance with the FTAPE fault injection tool. In Heinz Beilner and Falko Bause, editors, *Quantitative Evaluation of Computing and Communication Systems*, pages 26–40, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[TIA01] Telecommunications Industry Association (TIA). *TIA-644 - Electrical Characteristics of Low Voltage Differential Signaling (LVDS) Interface Circuits*, February 2001.

[TMS+13] Nikolaus Theißing, Dominik Merli, Michael Smola, Frederic Stumpf, and Georg Sigl. Comprehensive analysis of software countermeasures against fault attacks. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 404–409, 2013.

[Tol92] Keith M. Tolk. *Reflective particle technology for identification of critical components.* Sandia National Laboratories Albuquerque, NM, 1992.

[Tos15] David Tosenovjan. *Application Note AN5200 Rev. 1 – Error Correcting Codes Implemented on MPC55xx and MPC56xx Devices.* NXP Semiconductors, December 2015.

[TPI19] Lars Tebelmann, Michael Pehl, and Vincent Immler. Side-Channel Analysis of the TERO PUF. In *International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*, pages 43–60. Springer International Publishing, 2019.

[TPS17] Lars Tebelmann, Michael Pehl, and Georg Sigl. EM Side-Channel Analysis of BCH-based Error Correction for PUF-based Key Generation. In *Proceedings of the 2017 Workshop on Attacks and Solutions in Hardware Security*, pages 43–52, New York, NY, USA, 2017. ACM.

[TSŠ+06] Pim Tuyls, Geert-Jan Schrijen, Boris Škorić, Jan van Geloven, Nynke Verhaegh, and Rob Wolters. Read-Proof Hardware from Protective Coatings.

In *Proceedings of the 8th International Conference on Cryptographic Hardware and Embedded Systems*, pages 369–383. Springer Berlin Heidelberg, 2006.

[TV11] Ido Tal and Alexander Vardy. List decoding of polar codes. In *2011 IEEE International Symposium on Information Theory Proceedings*, pages 1–5, 2011.

[TZP20] Johannes Tobisch, Christian Zenger, and Christof Paar. Electromagnetic Enclosure PUF for Tamper Proofing Commodity Hardware and other Applications. In *Workshop on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE)*, 2020.

[VBRE07] Jonny Vinter, L. Bromander, P. Raistrick, and H. Edler. FISCADE - A Fault Injection Tool for SCADE Models. In *2007 3rd Institution of Engineering and Technology Conference on Automotive Electronics*, pages 1–9, 2007.

[VDE19] VDE. IT-Sicherheit und Industrie 4.0: Wo sehen Sie aktuell die größten Bedrohungen? [Graph]. `https://de-statista-com.eaccess.ub.tum.de/statistik/daten/studie/1013718/umfrage/umfrage-zu-den-bedrohungen-in-der-it-sicherheit-und-industrie-40-in-deutschland/`, 2019. In Statista. Online, accessed February 8, 2022.

[VNK$^+$15] Michael Vai, Ben Nahill, Josh Kramer, Michael Geis, Dan Utin, David Whelihan, and Roger Khazan. Secure architecture for embedded systems. In *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–5. IEEE, September 2015.

[VPK$^+$15] Tomá Vanát, Jan Pospíil, Filip Kríek, Jozef Ferencei, and Hana Kubátová. A system for radiation testing and physical fault injection into the fpgas and other electronics. In *2015 Euromicro Conference on Digital System Design*, pages 205–210, 2015.

[vR95] Rudolph L. van Renesse. 3DAS: a 3-dimensional-structure authentication system. In *European Convention on Security and Detection, 1995.*, pages 45–49, 1995.

[WBG17] Alexander Wild, Georg T. Becker, and Tim Güneysu. A fair and comprehensive large-scale analysis of oscillation-based PUFs for FPGAs. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–7. IEEE, 2017.

[Wei87] Steve H. Weingart. Physical Security for the $\mu$ABYSS System. In *1987 IEEE Symposium on Security and Privacy*, pages 52–52, 1987.

[WFP19] Florian Wilde, Christoph Frisch, and Michael Pehl. Efficient bound for conditional min-entropy of physical unclonable functions beyond iid. In *2019 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, 2019.

[Whi87] Steve R White. ABYSS: A Trusted Architecture for Software Protection. In *1987 IEEE Symposium on Security and Privacy*, pages 38–38. IEEE, 1987.

[Wil21] Florian K. A. Wilde. *Metrics for Physical Unclonable Functions.* PhD thesis, Technische Universität München, 2021.

[WKKG04] Kaijie Wu, Ramesh Karri, Grigori Kuznetsov, and Michael Goessel. Low cost concurrent error detection for the advanced encryption standard. In *2004 International Conferce on Test*, pages 1242–1248. IEEE, 2004.

[WMT+79] R. C. Wyatt, Peter J. McNulty, P. Toumbas, P. L. Rothwell, and R. C. Filz. Soft Errors Induced by Energetic Protons. *IEEE Transactions on Nuclear Science*, 26(6):4905–4910, 1979.

[WTLP14] Jiesheng Wei, Anna Thomas, Guanpeng Li, and Karthik Pattabiraman. Quantifying the accuracy of high-level fault injection techniques for hardware faults. In *Proceedings - 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014*, pages 375–382, June 2014.

[WTSS13] Stefan Winter, Michael Tretter, Benjamin Sattler, and Neeraj Suri. simfi: From single to simultaneous software fault injections. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12, 2013.

[Wyn75] Aaron D. Wyner. The wire-tap channel. *The Bell System Technical Journal*, 54(8):1355–1387, 1975.

[XRHB15] Xiaolin Xu, Ulrich Rührmair, Daniel E Holcomb, and Wayne Burleson. Security evaluation and enhancement of bistable ring PUFs. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 3–16. Springer, 2015.

[XX12] Jun Xu and Ping Xu. The research of memory fault simulation and fault injection method for bit software test. In *2012 Second International Conference on Instrumentation, Measurement, Computer, Communication and Control*, pages 718–722, 2012.

[YAL+03] Pedro Yuste, David Andrés, Lenin Lemus, Juan Serrano Martín, and Pedro Gil-Vicente. INERTE: Integrated NExus-based Real-Time fault injection tool for Embedded systems. In *2003 International Conference on Dependable Systems and Networks*, page 669, January 2003.

[YD10] Meng-Day Yu and Srinivas Devadas. Secure and robust error correction for physical unclonable functions. *IEEE Design & Test of Computers*, 27(1):48–65, 2010.

[YHD15] Meng-Day Yu, Matthias Hiller, and Srinivas Devadas. Maximum-likelihood decoding of device-specific multi-bit symbols for reliable key generation. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 38–43. IEEE, 2015.

[YLZ$^+$20] Weitao Yang, Yonghong Li, Weidong Zhang, Yaxin Guo, Haoyu Zhao, Jianan Wei, Yang Li, Chaohui He, Kesheng Chen, Gang Guo, Boyang Du, and Sterpone Luca. Electron inducing soft errors in 28 nm system-on-Chip. *Radiation Effects and Defects in Solids*, 175(7-8):745–754, 2020.

[YS18] Peihong Yuan and Fabian Steiner. Construction and Decoding Algorithms for Polar Codes based on $2 \times 2$ Non-Binary Kernels. In *2018 IEEE 10th International Symposium on Turbo Codes Iterative Information Processing (ISTC)*, pages 1–5, 2018.

[YW06] Chih-Hsu Yen and Bing-Fei Wu. Simple error detection methods for hardware implementation of Advanced Encryption Standard. *IEEE Transactions on Computers*, 55(6):720–731, 2006.

[ZHW$^+$21] Yin Zhang, Zhangqing He, Meilin Wan, Jiuyang Liu, Haoshang Gu, and Xuecheng Zou. A SC PUF standard cell used for key generation and anti-invasive-attack protection. *IEEE Transactions on Information Forensics and Security*, pages 1–1, 2021.

[ZLZ$^+$18] Fan Zhang, Xiaoxuan Lou, Xinjie Zhao, Shivam Bhasin, Wei He, Ruyi Ding, Samiya Qureshi, and Kui Ren. Persistent fault analysis on block ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018, Issue 3:150–172, 2018.

[ZOW$^+$16] Shaza Zeitouni, Yossef Oren, Christian Wachsmann, Patrick Koeberl, and Ahmad-Reza Sadeghi. Remanence decay side-channel: The PUF case. *IEEE Transactions on Information Forensics and Security*, 11(6):1106–1116, June 2016.